CADIA: COMBINATORIAL AUCTION WINNER DETERMINATION USING ITEM ASSOCIATION

by

Chi Hong Law

B.Sc. (Hons) Computer Science, Acadia University, 1992
Master of Computer Science, Dalhousie University, 1993

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY, INFORMATION TECHNOLOGY

In the School of
Interactive Arts and Technology

SIMON FRASER UNIVERSITY

Fall 2005

APPROVAL

Name: LAW, Chi Hong

Degree: PhD, Information Technology

Title of Thesis: CADIA: COMBINATORIAL AUCTION WINNER DETERMINATION USING ITEM ASSOCIATION

Examining Committee:

Chair: _____

Dr. Tom Calvert
Professor Emeritus, Interactive Arts & Technology

_____

Dr. Marek Hatala
Senior Supervisor
Assistant Professor, Interactive Arts & Technology

_____

Dr. Vive Kumar
Supervisor
Assistant Professor, Interactive Arts & Technology

_____

Dr. Rob Woodbury
Supervisor
Professor, Interactive Arts & Technology

_____

Dr. Michael Brydon
External Examiner
Assistant Professor, Business Administration
Simon Fraser University

_____

Dr. Holger H. Hoos
External Examiner
Assistant Professor, Computer Science
University of British Columbia

Date Approved: 04 August 2005

ABSTRACT

In combinatorial auctions (CAs), bidders are allowed to bid on any combination of items. Although CAs are economically efficient mechanisms for resources allocation, most auctioneers are hesitant to adopt them due to the fact that the CA winner determination process is a non-deterministic polynomial hard (NP-hard) problem. If an exhaustive search technique is used to solve the problem realistically, the number of auctioned items and bids must be small enough to be handled by the technique due to the constraints of today's computation power. Arising from the demand for CAs, this thesis presents a novel but also practical combinatorial auction winner determination approach. Such an approach has been designed and implemented into a system called CADIA. CADIA is able to generate results with high accuracy and good performance in CAs of hundreds of items and thousands of bids. CADIA's knowledge for winner determination is discovered from a process of mining the auction data using item association. Such knowledge is then used to identify particular bids as winners. Both potential winners and possible losers identified during the auctions are used as additional knowledge to further improve the results. Empirical evaluation shows that CADIA is more efficient than brute-force technique based systems in terms of running time when searching for the optimal revenue. In situations where obtaining the optimal revenue becomes unrealistic to be handled by the brute-force technique, as in auctions of hundreds of items and thousands of bids, CADIA finds better approximate revenue than greedy search based systems.

DEDICATION

To my parents who have given me the greatest love and care, and especially to my wife Sisie and my son Gabriel who have given me their warmest support.

ACKNOWLEDGEMENTS

I take this opportunity to express my greatest thanks to Dr. Marek Hatala who has, in various ways, helped me to attain my objective of preparing this thesis. I also thank Dr. Rob Woodbury, Dr. Vive Kumar, Dr. Holger Hoos, Dr. Robert Holte and Finnegan Southey for their valuable suggestions. The thesis would not have been completed without the above mentioned people.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER ONE:
INTRODUCTION

In combinatorial auctions (CAs), bidders are allowed to bid on any combination of items with the constraint that each item can be allocated to no more than one bidder. CAs are important in situations where the value of an item to a bidder strongly depends on other items he wins. Economists believe that combinatorial auctions (CAs) allow resources to be allocated in a more efficient way due to the exhibition of complementarity and substitutability when buyers valuate a set of items [Huberman et al., 1997; Boutilier et al., 1999; Huberman et al., 2000; Krishna, 2002]. From the auctioneer's point of view, the ultimate goal of a CA is to maximize the revenue from selecting some winning bids. Since the winner determination process in CAs is a very complex optimization problem, the number of auctioned items and bids must be small[1] enough if an exhaustive search technique is used to obtain the optimal revenue. Airport slots allocation, resources allocation by NASA space station, Sears transportation acquisition auction, supply chain formulation, and spectrum auctions by the US Federal Communications Commission (FCC) are examples of real-world CA [Rothkopf, et al., 2000; Rassenti et al., 1982]. Due to the increasing demand for CAs, the winner determination problem has recently received considerable attention in the fields of economics and computer science.

[1] An exhaustive search algorithm belongs to the complexity class of 2^n. It takes 1 hour and 1 decade to solve problems of size n=51 and n=68 respectively on a supercomputers performing a single floating-point operation in 10^{-12} seconds [Johnsonbaugh and Schaefer, 2004]. The term "small" is used throughout the thesis to refer to problem size n <= 50, i.e., 50 items in the case of CAs.

Unfortunately, CA winner determination belongs to the class of non-deterministic polynomial hard (NP-hard) problem [Rothkopf, 1998; Fujishima et al., 1999; Bjorndal and Jornsten, 2000]. The number of items and bids directly impact the time of finding the winners. For M items, there are $(2^M - 1)$ combinations of items; and for B bids, there are $(2^B - 1)$ combinations of candidate winners. The CA winner determination problem becomes computationally intractable when the number of items and number of bids are large[2]. For instance, a system based on the brute-force, exhaustive search technique running on a Pentium personal computer is able to find the optimal revenue for an auction of 9 items and 10 bids in 3 minutes, but will take about 4 minutes and 30 minutes when the number of items is increased by 1 and 2 respectively. Systems based on the exhaustive search technique are impractical in real world auctions when there are hundreds of items and thousands of bids.

In recent years, some techniques were proposed to find the optimal or approximate solution for CA winner determination. Integer programming (IP) technique [Andersson et al., 2000], which is able to obtain the optimal revenue, can practically handle CA winner determination in auctions of hundreds of items and thousands of bids. However, one of the leading commercial implementations based on IP called CPLEX states in its user's guide that some common difficulties are encountered when solving IP problems [ILOG, 2005]. These difficulties are "running out of memory", "failure to prove optimality". A few heuristics techniques [Sandholm, 1999; de Vries and Vohra, 2000; Hoos and Boutilier, 2000; Nisan, 2000; Sandholm et al., 2000; Sandholm, 2002] were proposed and proved to be able to solve the problems with hundreds of items and

[2] The term "large" is used throughout the thesis to refer to problem size $n > 100$. In recent publications, sample auctions of hundred items and thousands of bids are used when evaluating proposed techniques.

thousands of bids. However, these techniques include one of or a combination of greedy search, depth-first search, and branch-and-bound tree search strategies. The drawbacks of these techniques are that the results may or may not be optimal if the algorithms implementing the techniques are terminated prematurely.

CADIA, which is a CA winner determination system developed upon the proven knowledge discovery technique in data mining called item association, provides a novel and practical approach to solve the problem. Since auction is a real world business process, it is worthwhile to use the item association pattern as knowledge in problem solving. CADIA applies such knowledge discovered from the auction data in winner determination which has been overlooked in any published techniques. In addition, CADIA uses a tactical-bid-elimination technique to further improve its result.

In an evaluation where the goal is to obtain the optimal revenue, CADIA is compared to a brute-force (exhaustive search) technique based system and is concluded empirically to be a practical system and runs at least 20 times faster than a brute-force technique. When there are hundreds of items and thousands of bids, a comparison of the brute-force technique and CADIA becomes unrealistic due to the large size of problem instance. Thus, in another evaluation, CADIA is compared to an approximation system that is based on the greedy and depth-first search technique. Empirical results show that CADIA always find better revenue. The current implementation of CADIA uses in-memory storage and search techniques that can practically handle up to five hundred items and two thousands bids running on a Pentium based personal computer. Such a limitation can be overcome when external memory storage and search techniques are employed in the trade off of speed.

In Chapter 2, I review the major characteristics and benefits of CAs followed by the definition of the CA format. I also present the CA winner determination problem and survey the state of knowledge about techniques that are able to find optimal or approximate solutions. Chapter 3 describes item association, which is the core technique adopted by CADIA, and its importance in association rule mining applications. Chapter 4 presents CADIA's hypothesis, core structure, and algorithms. An auction example of 10 items and 10 bids will be used to illustrate the core concepts of CADIA. Chapter 5 describes how CADIA is evaluated in terms of data collection, setup, and experimental results. Chapter 6 and 7 presented an improved version of CADIA and its evaluation respectively. Chapter 8 discusses CADIA's practicality and the shortcomings of CADIA and other evaluated techniques. Chapter 9 presents the conclusion and possible future research directions.

CHAPTER TWO:
COMBINATORIAL AUCTIONS

2.1 Social Benefits

In CAs, the bidders' valuations in most cases are not additive [Bichler, 1999] because the value of a combination of items may not be equal to the sum of the values of the same items unbundled. A bidder considers a set of items as a complement bundle of items when he values the bundle higher than the sum of the single item values. Contrarily, a bidder considers a set of items as a substitute bundle of items when he values the bundle lower than the sum of the single item values [Krishna, 2002]. Because combinations of items in bids generally overlap, the CA winner determination becomes an optimization problem.

Even though the CA winner determination is a NP-hard problem [Rothkopf, 1998; Fujishima et al., 1999], CA is believed to be an efficient way to allocate resources to buying agents whose preferences exhibit complex structure with respect to complementarity and substitutability [Rassenti et al., 1982; Rothkopf et al., 1998; Wellman et al., 2001]. If complementarities and substitutability exist among auctioned items, evidence suggests that it is more appropriate to permit bidders to bid for combinations, rather than on individual item because bidders do not get stuck with partial bundles of low value [Banks et al., 1989]. If an exhaustive search technique is used to

solve the problem realistically, the number of auctioned items and bids must be small enough to be handled by the technique due to the constraints of today's computation power.

Combinatorial auctions were first proposed by Jackson [1976] for radio spectrum rights. Later, Rassenti et al. [1982] proposed such auctions to allocate airport time slots. Strevell and Chong [1985] described the use of an auction to allocate vacation time slots. Banks et al. [1989] proposed a combinatorial auction for selecting projects on the space shuttle, but the prototype was tested experimentally and was never implemented due to political reasons. Olson et al. [2000] described the design and use of a combinatorial auction that was employed by Sears in 1993 to select carriers. In this auction, delivery routes were bid upon. Since bidders were allowed to bid on combinations of routes, they had the opportunity to construct routes that utilized their trucks as efficiently as possible. Graves et al. [1993] described the auction of seats in a course that was executed regularly at the University of Chicago's Business School. Srinivasan et al. [1998] proposed a mechanism for trading financial securities that allowed buyers and sellers to offer bundles of financial instruments. In 1994, Federal Communications Commission (FCC) planned to use a CA auction to allocate spectrum rights [Cramton, 1997; Cramton and Schwartx, 2000] because bidders were interested in different collections of spectrum licenses.

In recent years, a number of logistics consulting firms offered CA software [Case, 2001]. For example, SAITECH-INC offers a software product called SBIDS that allows trucking companies to bid on bundles of lanes. In 1998, OptiMark Technologies offered an automated trading system that allowed bidders to submit price-quantity-stock triples along with a priority list. The Securities and Exchange Commission (SEC) approved

Pacific Stock Exchange's proposal to implement this electronic trading system. In 1998, the NASDAQ announced plans to introduce this technology to its dealers and investors. Logistics.com claims that by January 2000 more than $5 billion in transportation contracts had been bid on using a CA system called OptiBid by Ford Motor Company, Wal-Mart, and Kmart [de Vries, S. and Vohra, R., 2000]. CombineNet claims that its Rev technology runs much faster than the state-of-art general purpose mixed integer programming solver. Its customer includes some of the Fortune 100 and Global 1000 companies.

2.2 Auction Format

The three major issues one must deal with in designing a CA are bidding protocol, allocation, and payment [Nisan, 2000]. Each bidder must be able to express bids on combinations of items. Each bid may be interpreted as the maximum amount of money that the bidder is willing to pay for. The bidding protocol determines how this bidding communication is done. The items in the auction must be allocated among the different bidders. The allocation will attempt to optimize some objective function, usually the auctioneer's revenue. Each winner of a set of items will pay according to the payment rules. A well-designed auction will ensure that the intended goals of the auction are met when all bidders act according to their chosen strategies.

CA design becomes an interdisciplinary study and has received considerable attention in the fields of economics and computer science. Bidding protocol, allocation, and payment have been treated as independent research topics by economists and

computer scientists in recent years. Resembling many other CA related computer science research [Gonen and Lehmann, 2000; Sandholm, 2002], my research focuses only on the allocation problem. Bidders are assumed to act non-strategically and bids are sealed and assumed to be simply the bidders' valuations. As stated, the allocation attempts to optimize the auctioneer's revenue according to the declared bids. Although the auctioneer's ultimate goal is to attain maximum revenue, he will find it computationally intractable when the numbers of items and bids are large due to the NP-hard nature of the winner determination problem. For M items, there are $(2^M - 1)$ combinations of items for a bidder who fully expresses its preferences must bid on all these combinations. This is definitely undesirable because it is computationally intractable to determine one's valuation for any given combination [Parkes, 1999].

2.3 Winner Determination Problem

The CA winner determination problem can be represented mathematically by the following notation [Sandholm et al., 2001a]. Let M={0, 1, 2, ...m} be the set of auctioned items, and B={b_0, b_1, b_2, ...b_n} be the set of bids, and each bid is a tuple b_j = (S_j, p_j), where $S_j \subseteq M$ is a subset of M and $p_j \geq 0$ for all $j \in \{0, 1, 2, \ldots, n\}$ is a price offered by b_j.

$$\max \sum_{j=0}^{n} p_j x_j \quad \text{Subject to} \quad \sum_{j|i \in Sj} x_j \leq 1, \quad i = 1,2,\ldots m \qquad x_j \in \{0,1\}$$

x_j is called a decision variable and its value is 1 when b_j is a winner, 0 otherwise. The CA winner determination is to identify the bids as winners or losers with the aim to maximize auctioneers' revenue under the constraint that each item can be allocated to at most one bid. For example, when M = {0, 1, 2, ..., 9} and B = {b_0, b_1, b_2, ..., b_9}, we may have the following bidding pattern.

{bid}	{a set of items}	{bidding price}
{b_0}	{0, 4, 6, 7}	{206.28}
{b_1}	{0, 1, 3, 4}	{207.28}
{b_2}	{0, 6}	{205.00}
{b_3}	{0, 4, 5, 9}	{208.28}
{b_4}	{2, 4, 5, 8}	{108.28}
{b_5}	{1, 2, 7}	{55.74}
{b_6}	{1, 2, 3, 6}	{55.74}
{b_7}	{2, 9}	{152.00}
{b_8}	{0, 4, 8}	{154.74}
{b_9}	{0, 4, 6, 7, 8}	{205.50}

Figure 1 A file containing the bidding pattern of 10 items and 10 bids.

A typical CA winner determination system, as described in Figure 1, accepts a number of items and a number of bids as inputs and identifies a subset of all bids as outputs. The output bids become the winners.

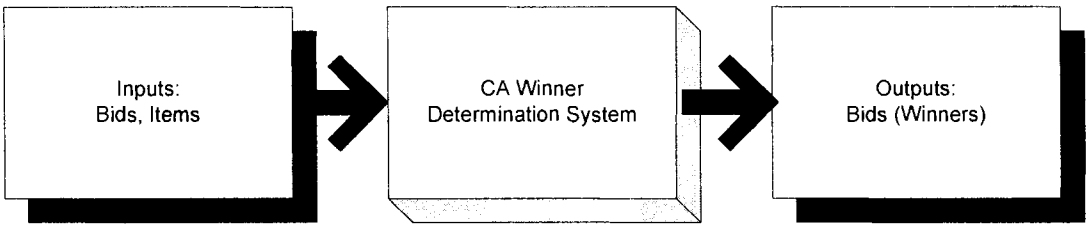

Figure 2 A combinatorial auction winner determination system.

The constraint for the system is that each item can be allocated to at most one bid. Since the CA winner determination is a NP-hard problem, a more realistic system should possess the following characteristics:

1. **the system should generate approximate revenue that is close to the optimal revenue, and**
2. **the system should be capable to handle a large number of items and bids in a single auction.**

The CA winner determination can be translated to another NP-hard problem such as the weighted set-packing problem [Rothkopf, et al., 1998; Karp, 1972]. A problem is assigned to the NP-hard class if it can be solved by a NP algorithm. A NP algorithm is a two-phase procedure. During the "nondeterministic" phase, a candidate solution is generated. In the "verification" phrase, the candidate solution is verified using a deterministic algorithm. For example, if a problem is known to be NP, and a solution to

the problem is somehow known, then demonstrating the correctness of the solution can always be reduced to polynomial (P) time verification. The proof of a problem is a NP problem can be summarized in the following three steps [Sipser, 1997]. The CA winner determination problem is used here to illustrate these steps.

1. **A subset of bids B can be selected from all bids non-deterministically.**
2. **B can be verified using a deterministic algorithm. That is, B must contain no conflicted bids to claim itself as a solution.**
3. **If the verification test passes, the solution is accepted; otherwise, it is rejected.**

If an instance of the NP-hard problem in question is small, we might be able to solve it by the brute-force search algorithm described in Section 2.5.1. Even though this approach works in principle, its practicality is very limited because the number of instance parameters is usually very large in real-world problems [Levitin, 2003, Johnsonbaugh and Schaefer, 2004].

2.4 Feasible, Approximate or Optimal Solution

The CA winner determination problem is in fact an optimization problem because it aims to maximize auctioneers' revenue as the objective function subject to the constraint that each item can be allocated to at most one bid. The terms feasible, approximate, and optimal solutions have been used very often in the research of optimization problems, and thus a formal definition for each is needed here. In computer

science terminology [Levitin, 2003], a feasible solution to an optimization problem is a point in the problem's search space that satisfies all the problem's constraints, while the optimal solution is a feasible solution with the best value of the objective function. An approximate solution is also a feasible solution with good but not necessarily the best value. When obtaining the optimal solution is unrealistic, an approximate solution is preferred to a feasible solution. The three different kinds of solutions to the CA winner determination problem can be represented mathematically [Papadimitriou and Steiglitz, 1998] by the following notation.

An instance of the CA optimization problem is a pair (F, r), where F is the set that contains all feasible solutions; r is the revenue function.

When searching for a feasible solution, we need to find solution $s_f \in F$ for which $r(s_f) \geq 0$.

When searching for the optimal solution, we need to find solution $s_{opt} \in F$ for which s_{opt} must be a feasible solution and $r(s_{opt}) \geq 0$ and $r(s_{opt}) \geq r(s_f)$ for all $s_f \in F$.

When searching for an approximate solution, we need to find solution $s_a \in F$ for which s_a must be a feasible solution and $r(s_a) \leq r(s_{opt})$. In addition, the bid set B_a used to obtain s_a must not be a subset of bid set B_f of any other feasible solution $s_f \in F$.

The brute-force based techniques are able to obtain the optimal solution, but these techniques are impractical and are used only when the size of the problem instance in

question is small. When the instance size is large, the approach of finding an approximate solution becomes more attractive. The generic greedy search based techniques are guaranteed to obtain a feasible solution. However, there is no guarantee whether such a solution is a good approximate solution or not. Thus, a formal approach of evaluating a proposed technique should measure both its accuracy ratio and performance ratio. Such an approach has been adopted in CADIA's evaluation as described in Section 5.

2.5 Optimization Problem Solving Techniques

Because of the intractability nature [Papadimitriou and Steiglitz, 1998; Sipser, 1992] of the CA winner determination problem, much research has focused on sub-cases of the problem that are tractable [Rothkopf et al., 1998; Lehmann et al., 1999; Tennenholtz, 2000; Ronen, 2001]. For example, both the number of auctioned items and bids can be restricted to be small enough to be handled by an optimal revenue search technique within the constraints of today's computation power. Unfortunately, these sub-cases are very restrictive and therefore are not applicable to many CA domains. In fact, there is no substitute for a CA if an auctioneer aims to allocate resources efficiently. Thus, many researchers have begun to propose heuristic techniques for winner determination in CAs.

All proposed heuristic techniques can be classified into exact methods and approximate methods [de Vries and Vohra, 2000]. An exact method for solving the CA problem is one that is guaranteed to produce an optimal solution if run to completion.

With approximate methods, one seeks a feasible solution fast and hopes that it is near optimal. This raises the obvious question of how close to optimal the solution is.

2.5.1 Brute-Force Technique

If problem solving is seen as a search in the state space [Russell and Norvig, 2003], the brute-force technique would be described as an exhaustive search for all possible states in the problem space with an aim to optimize some objective function. The implementation of a winner determination based on the brute-force technique is quite straightforward (Algorithm 1). First, all bid combinations based on the available bids are generated. Second, a bid combination will be removed if it has conflicts among its bids. A conflict is found when an item is wanted by more than one bid. Last, the bid combination which has the highest total price becomes the set of winners.

```
Algorithm:  CA winner determination based on the
            brute-force technique
Input:      all bids B, b_i ∈B and bid tuples (S_i,p_i),
            where S_i is a subset of wanted items and p_i is
            the bid price, and i∈{0,1,…n}.
Output:     a set of winners, B_winners
Begin
      Generate all bid combinations B_c from B
      B_candidate ← B_c
      for each bid combination B_a ∈ B_c
         for each bid b_i ∈ B_a
            for each bid b_j ∈ B_a and j ≠ i
               if (S_i ∩ S_j ≠ ∅)
                  B_candidate ← B_candidate - B_a
      B_winners ← highest_price_bid_comb (B_candidate)
End

Function highest_price_bid_comb (B_candidate)
Begin
      p_h ← 0, B_h ← ∅
      for each bid combination B_i ∈ B_candidate
          P_i = total_price (B_i)
          if (P_i > p_h){
            p_h ← P_i
            B_h ← B_i
          }
      return B_h
End

Function total_price (B)
Begin
      p_t = 0
      for each bid b_i ∈ B
          p_t = p_t + p_i
      return p_t
End
```

Algorithm 1 CA winner determination based on the brute-force technique.

As an illustration, suppose we have two auctioned items, M = {0, 1}, three bids, B = $\{b_0, b_1, b_2\}$ and the bid data as shown in Figure 3. The number of all possible item combinations is $(2^2 - 1)$ or 3. That is, a bidder who fully expresses its preferences may bid on all three combinations.

{bid}	{a set of items}	{bidding price}
$\{b_0\}$	{0}	{10.00}
$\{b_1\}$	{0, 1}	{18.00}
$\{b_2\}$	{1}	{10.00}

Figure 3 An example of bid data.

The number of all possible bid combinations is $(2^3 - 1)$ or 7. The brute-force technique will search for all bid combinations with an aim to maximize the revenue. According to Algorithm 1, all combinations of bids are generated. They are $\{b_0\}$, $\{b_1\}$, $\{b_2\}$, $\{b_0, b_1\}$, $\{b_0, b_2\}$, $\{b_1, b_2\}$ and $\{b_0, b_1, b_2\}$. A conflict exists when an item is wanted by more than one bid. Since b_0 conflicts with b_1, b_1 conflicts with b_2, and b_1, b_2, b_3 are in conflict with each other, the bid combinations $\{b_0, b_1\}$, $\{b_1, b_2\}$ and $\{b_0, b_1, b_2\}$ are discarded. At last, the bid combination that generates the highest revenue among all remaining combinations becomes the set of winners. Since the remaining combinations are $\{b_0\}$, $\{b_1\}$, $\{b_2\}$, $\{b_0, b_2\}$, the winner goes to $\{b_0, b_2\}$ because it generates the highest revenue of $20.

For the CA winner determination problem solving, the brute-force technique leads to an algorithm that is extremely inefficient because it has a running time complexity of an exponential order of magnitude of $(2^{|B|})$ when generating all bid combinations. When $|B| = 100$, for example, the order of magnitude becomes 2^{100} or 1.3×10^{30}. As a result, if the brute-force technique is applied, the winner determination problem can be solved in polynomial time only if there are an infinite number of processors and if conflicts among all bids can be identified at once.

2.5.2 Greedy Search Technique

Greedy search makes use of a heuristic function to order the searched nodes within the search space [Lawler, 1985; Lawler et al., 1992]. Thus, the search technique will choose the searched node that appears to be the best based on the function, regardless of its position in the state space. When the technique is applied to the CA winner determination problem, all bids can be treated as nodes within the search space. Since the goal is to maximize the revenue, a greedy search will start expanding the node that is estimated to be closest to the goal state, that is, a bid with the highest bidding price. Algorithm 2 describes the technique when applied to the CA winner determination problem.

When the algorithm is applied to the same data as described in Figure 3, b_1 will become the first winner because it offers the highest bidding price. It is also the only winner because b_0 and b_2 have conflicts with b_1. As a result, the revenue generated (i.e. \$18) is not optimal. Although the greedy technique can always find a solution quickly, its revenue may not be optimal and sometimes far from approximate.

```
Algorithm:  CA winner determination based on the
            greedy search technique
Input:      all bids b_i ∈B and bid tuples (S_i,p_i),
            where S_i is a subset of wanted items and p_i is
            the bid price, and i∈{0,1,…n}.
Output:     a set of winners, B_winners
Begin
      B_winners ← {∅}
      Loop until each bid b_i ∈ B has been identified as a
      winner or loser {
         b_h ← highest_price_bid (B)
         B_winners ← B_winners + b_h
         B ← B - b_h
         B ← B - all_conflicted_bids (b_h)
      }
End

Function highest_price_bid (B)
Begin
      p_h ← 0
      for each bid b_i ∈ B
         if (p_i > p_h){
            p_h ← p_i
            b_h ← b_i
         }
      return b_h
End

Function all_conflicted_bids (b_h)
Begin
      B_losers ← {∅}
      for each bid b_i ∈ B
         if (S_h ∩ S_i ≠ ∅)
            B_losers ← B_losers + b_i
      return B_losers
End
```

Algorithm 2 CA winner determination based on the greedy search technique

However, the greedy technique can be used to determine the lower bound revenue during a CA winner determination algorithm and may be considered part of a CA system design. Some recently proposed CA winner determination systems that use the greedy technique with random starts have shown that the revenue can be improved significantly [Holte, 2001].

2.5.3 Integer Programming

Integer programming (IP) [Nemhauser and Wolsey, 1999; Miller, 2000] has been used to solve optimization problems. It is a technique that aims to maximize an objective function subject to the constraint that the solution values of the variables be integers.

IP problem can be illustrated graphically with the following simple example. Suppose we would like to maximize the objective function based on values of two variables x_1 and x_2. A common approach for solving integer programming problems is to start by relaxing IP problems to linear programming (LP) problems. A LP problem is a problem of optimizing a linear function of several variables subject to constraints in the form of linear equations and linear inequalities. The LP problem can then be solved by the simplex method, which was developed by George B. Danzig in 1947 [Cooper and Steinberg, 1974; Haeussler et al., 2002].

With a LP, we may have the graph depicted in Figure 4. Since only integer values for the variables are allowed, we may have a solution (shown as ×) bounded by the feasible region. Intuitively, we may be tempted to round up or down the values of x_1 and

x_2 as final solutions, but such an approach may end up with an infeasible solution. By using the cutting-plane method, we can derive the region that connects the "outermost" feasible lattice points. As a result, the integer optimum will be interior to the region bounded by the dotted lines, and the lines when x_1=0, and x_2=0 (Figure 5).

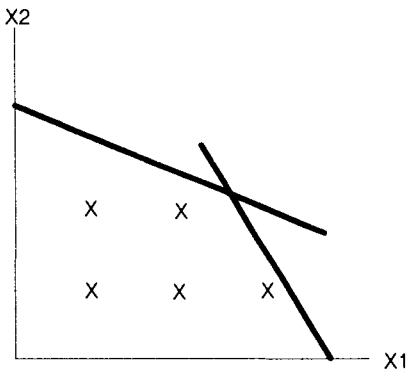

Figure 4 An IP problem is relaxed as a LP problem.

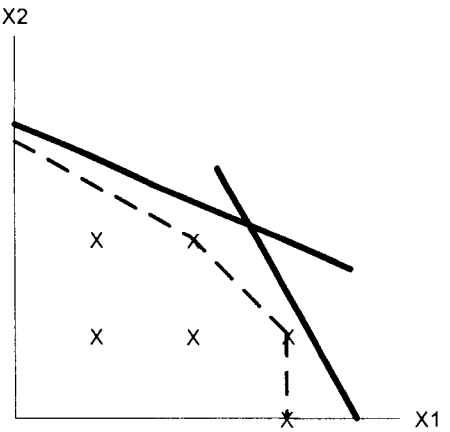

Figure 5 The optimal integer solution can be derived with cutting-plane method.

Thus, the allocation problem in CA winner determination can be formalized as an integer programming (IP) [Andersson et al., 2000] problem since its aim is to maximize the revenue as the goal subject to the constraint that the solution values of the variables

(i.e. bids) be whole numbers (i.e. 0 or 1). As an illustration, the auction example described in 2.5.1 can be formalized as the following IP problem:

$$\begin{aligned} \textbf{Maximize} \quad & z = 10b_0 + 18b_1 + 10b_2 \\ \textbf{subject to} \quad & b_0 + b_1 \leq 1 \\ & b_1 + b_2 \leq 1 \\ \textbf{with} \quad & b_0, b_1, b_2 = 0 \text{ or } 1 \end{aligned}$$

The IP program is then relaxed as a LP problem.

$$\begin{aligned} \textbf{Maximize} \quad & \text{revenue} = 10b_0 + 18b_1 + 10b_2 \\ \textbf{subject to} \quad & b_0 + b_1 + b_3 = 1 \\ & b_1 + b_2 + b_4 = 1 \\ \textbf{with} \quad & b_0, b_1, b_2, b_3, b_4 \geq 0 \text{ and } \leq 1 \end{aligned}$$

Since the problem involves more than two variables, the simplex method is recommended because the graphical method is usually too inefficient. The optimal solution is found when b_0 and $b_2 = 1$, and b_1, b_3 and $b_4 = 0$. The maximum revenue is \$20.

Theoretically, integer programming is guaranteed to find an optimal solution [Rothkopf et al., 1998; Pekeč et al., 2000]. However, one of the leading commercial implementations based on IP called CPLEX states in its user's guide that some common difficulties are encountered when solving IP program [ILOG, 2005]. These difficulties include "running out of memory" and "failure to prove optimality".

2.5.4 Branch and Bound

Nisan [2000] suggests a branch-and-bound technique based on integer programming (IP) relaxation. The technique is able to return the optimal revenue. In essence, the technique [Hoffman and Padberg, 1993; Gonen and Lehmann, 2000] generates a treelike structure to identify and solve a set of increasingly constrained sub-problems, derived from the original integer linear program. In branch and bound, the technique first explores the most promising directions as is done by the greedy search technique. This will hopefully provide very good lower bounds quickly. It is expected that the upper bounds obtained using the IP relaxation will usually be close enough to the optimal. Combined with good lower bounds, further search can be reduced. As an illustration (Figure 6), instead of testing for each possible candidate solution (shown as •), the technique picks a candidate (e.g. when $x_1 = c$) as a temporary solution and then tests for the possibility of branching out to improve the solution further. The idea is to divide the feasible solution space for this problem into two sub-spaces during each branch until no further improvement is possible.

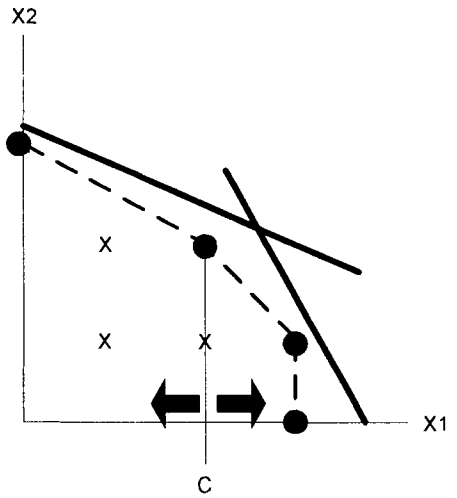

Figure 6 The solution space is divided during the branching process.

The branch-and-bound technique is sensitive to the upper and lower bound values obtained during the process. CPLEX, which also uses branch-and bound technique, is known to "run out of memory" when the branch and bound tree is large [ILOG, 2005]. That is, the search tree can grow so large that there are too many sub-spaces to be investigated. Besides, in a problem with a hundred variables such as the CA, it becomes arguable why a particular variable is chosen for initial branching. In addition, it is difficult to know from the beginning which branches are better than others.

2.5.5 Constraint Programming

Constraint programming (CP) is an alternative approach for solving combinatorial optimization problems [Smith et al., 1997; Lustig and Puget, 2001] because the problems can be formulated as constraint satisfaction problems (CSPs). A CSP consists of a set of variables, each with a finite set of possible values (its domain), and a set of constraints which the values assigned to the variables must satisfy. In a CSP that is also an optimization problem, there is an additional variable representing the objective; each time a solution to the CSP is found, a new constraint is added to ensure that any future solution must have an improved value of the objective, and this continues until the problem becomes infeasible, when the last solution found is known to be optimal.

As an illustration, the problem in Section 2.5.1 can be formulated as a CSP and can be solved with CP. The problem is to determine the winners in an auction of two auctioned items, $M = \{0, 1\}$ and three bids, $B = \{b_0, b_1, b_2\}$ and the bid data as shown in Figure 3. First, we define the variable to be the bids: b_0, b_1, and b_2. Each variable's

domain is the set {{0}, {1}, {0, 1}} and its value is determined based on the bid content. The constraint requires that each item can be allocated to at most one bid.

Since the number of variables in our example is small, the CSP can be visualized as a constraint graph [Russell and Norvig, 2003], as shown in Figure 7. The nodes of the graph correspond to variables of the problem and the arcs correspond to constraints.

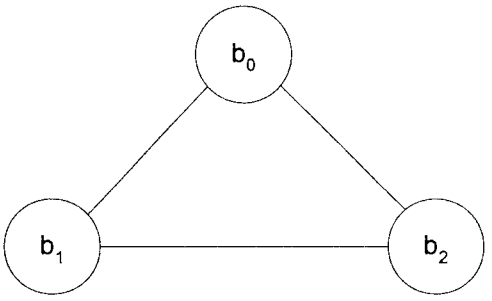

Figure 7 The auction problem represented as a constraint graph.

The most common CP algorithm for solving CSP is backtracking. Backtracking [Levitin, 2003] can be seen as a more intelligent variation of the brute-force technique. The idea is to construct a depth-first search tree one node at a time and evaluate if a partially constructed tree is a feasible solution. Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution; the nodes of the second level represent the choices for the second component, and so on. Leaves represent either dead ends (shown as ×) or feasible solutions (shown as √). The search tree for the auction problem is shown in Figure 8, where we have assigned variables in the order b_0, b_1, and b_2.

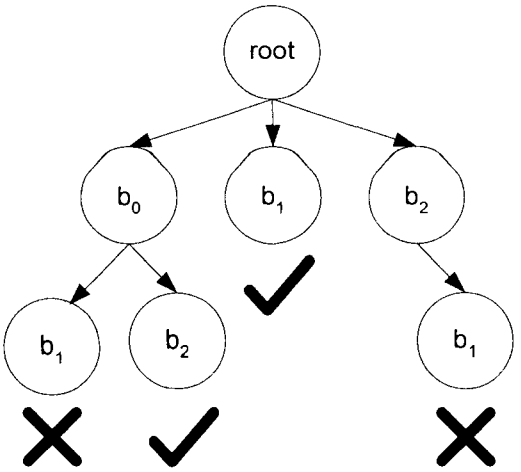

Figure 8 The search tree for the auction problem

Whenever a partially constructed tree violates a constraint (shown as ×), backtracking is able to eliminate a subspace during the search. For example, once we assign item-0 to b_0, we can infer that now only item-1 is available and any bids that request item-0 will violate the constraint. Thus, the construction of the tree $b_0 \rightarrow b_1$ is not feasible. As a result, the constraints can help us reduce the problem search space. Besides, the bid ordering in an auction is commutative because the order of the nodes has no effect on the outcome. That is, the solutions given by the trees $b_2 \rightarrow b_0$ and $b_0 \rightarrow b_2$ are the same. Once the former has been explored, the latter becomes redundant. As a result, the inclusion of the commutativity property can further eliminate the search space.

CP can further be improved through constraint propagation. The most popular technique is forward checking [Russell and Norvig, 2003]. In forward checking, whenever a current variable X is assigned, it looks at each future variable Y that is connected to X and removes temporarily from Y's domain any value that conflicts with

this assignment. The technique is able to conclude a partial solution is not feasible if the domain of Y is empty. As a result, the technique allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking. When forward checking is used in the auction winner determination problem, it simply checks if future variables conflict with the current variable since the values of all variables have been assigned.

Although backtracking is better than the brute-force technique, its running complexity for most nontrivial problems is still exponential. A paper by Smith et al. [1997] concludes that CP is useful only if the assignment of a value to a variable can trigger the pruning of a significant amount of problem search space. He also added that CP is less useful when the problem involves large numbers of variables.

2.5.6 Other Techniques and Commercial Implementations

Sandholm [2001b, 2002] described an algorithm called CABOB (Combinatorial Auction Branch On Bids), and had run tests on randomly generated instances, the largest of which involved 400 items and 2000 of bids. CABOB is in fact a depth-first, branch-and-bound tree search technique. In addition, the algorithm addresses several special cases during the search, and uses LP for upper bounding and a relatively simple greedy algorithm for lower bounding. The algorithm performs a short dynamic analysis of the underlying LP problem, and then uses the most suitable bid ordering heuristic. Empirical

results indicate that CABOB solves CA problems within seconds for auction size of hundred of items and thousands of bids, but also show that it cannot guarantee a polynomial running time for every input.

Fujishjima et al. [1999] proposed a set of approximate methods called CASS (Combinatorial Auction Structured Search). CASS uses "binning" where the bids are grouped into mutually exclusive bins or subsets. The maximal revenue comes either from a single bid or from the sum of the maximal revenues of two disjoint exhaustive subsets. The time saving comes from the fact that the number of bids to be dealt with is much smaller in the subsets as compared to a set containing all bids. It is obvious that the technique does not scale well because it requires an exhaustive search for all mutually exclusive bids. To overcome this issue, CASS applies pruning to reduce the search space.

Hoos and Boutilier [2000] described a stochastic local search approach to solve the CA problem, and characterized its performance with a focus on time-limited situations. Since it is a local search approach, it uses a goal test to estimate the distance to the goal state. The test involves ranking bids according to expected revenue. One obvious problem is that a local search algorithm can get caught in local maxima. Once at the top of the locally best solution, moving to any other node would lead to a node with less optimal results. Another possibility is that a plateau or flat spot exists in the problem space. Once the search algorithm gets up to this area all moves would have the same result and so progress would be halted. However, the stochastic nature of the search will randomly choose moves to avoid the problem.

CA winner determination algorithms have been implemented commercially [de Vries, S. and Vohra, R., 2000]. Logistics.com's OptiBid software has been used in

situations where the number of items averages 500. OptiBid does not limit the number of distinct subsets that bidders bid on or the number of items allowed within a bid. OptiBid is based on the integer programming technique with a series of proprietary formulations and heuristic algorithms. SAITECH-INC's SBID is also based on the integer programming and proprietary techniques. SAITECH-INC reports that SBID is able to handle problems of a similar size as OptiBid. CombineNet's Rev technology is based on the tree searching algorithm, combining with branch and bound, cutting planes, and a series of proprietary algorithms.

CHAPTER THREE:
ITEM ASSOCIATIONS

3.1 Concepts

CADIA's core algorithm is based on the item association [Han and Kamber, 2001; Dunham, 2003] technique which has been widely adopted in many real-world data mining applications. For instance, a sales manager may wonder, "Which sets of items are customers likely to purchase together?" A more specific question may be "How likely is item x_1 to be purchased after item x_2 is purchased?" The answers to these questions, which become the domain knowledge, can help decision makers to strategically encourage the overall sale. CADIA uses the technique to discover knowledge from the auction data which has been overlooked in any published techniques. Such knowledge is used in an informed search to identify winners. The concept of item association, which depends on identifying all frequent itemsets [Pasquier et al, 1999; Pei et al., 2000] in transactions, can be defined by the following characteristics as:

Let $I = \{1, 2,\ldots, m\}$ be a set of items and $T = \{t_1, t_2,\ldots, t_n\}$ be a database of transactions where $t_i \subseteq I$, and $Support_{min}$ be an expert-defined minimum support count.

1. **A set of items is called an itemset. An itemset that contains k items is a k-itemsets.**

2. **A candidate k-itemset's frequency count is the number of transactions that contains k-itemset.**
3. **A candidate k-itemset C_k becomes a frequent k-itemset F_k if its frequency count is greater than or equal to $Support_{min}$.**
4. **The set of candidate 1-itemsets C_1 is I. The set of candidate -itemsets C_k ,where k = 2, 3, ...m, is generated by joining F_{k-1} with itself as $F_{k-1} \triangleright\triangleleft F_{k-1}$.**

The following algorithm, which is based on the Apriori property [Agrawal et al., 1993], is used to identify all frequent itemsets.

```
Algorithm:  Identifying all Frequent Itemsets
Input:      transaction database T={t1, t2,..., tn},
            a list of Items I={1,...,m}, and Supportmin
Output:     all frequent itemsets F1, F2, ..., Fm
Begin
      candidate 1-itemsets C1 ← I
      for each itemset cj ∈ C1
            if (frequency(cj) ≥ Supportmin)
               F1 ← F1 ∪ cj
      for i ∈ {2..m} {
         Generate candidate i-itemsets Ci by Fi-1 ▷◁ Fi-1
         for each itemset cj ∈ Ci
            if (frequency(cj) ≥ Supportmin)
               Fi ← Fi ∪ cj
      }
End

Function frequency (c)
Begin
      return (number of transactions that contain c)
End
```

Algorithm 3 Identify all frequent itemsets.

As an illustration, given a transaction $T=\{t_1, t_2, t_3, t_4, t_5\}$ containing five transactions, a list of items $I=\{1,2,3,4,5\}$, and an expert-assigned $Support_{min}=2$, the sets of candidate itemsets C_1, C_2, C_3 and the sets of frequent itemsets F_1, F_2 are generated (Figure 9).

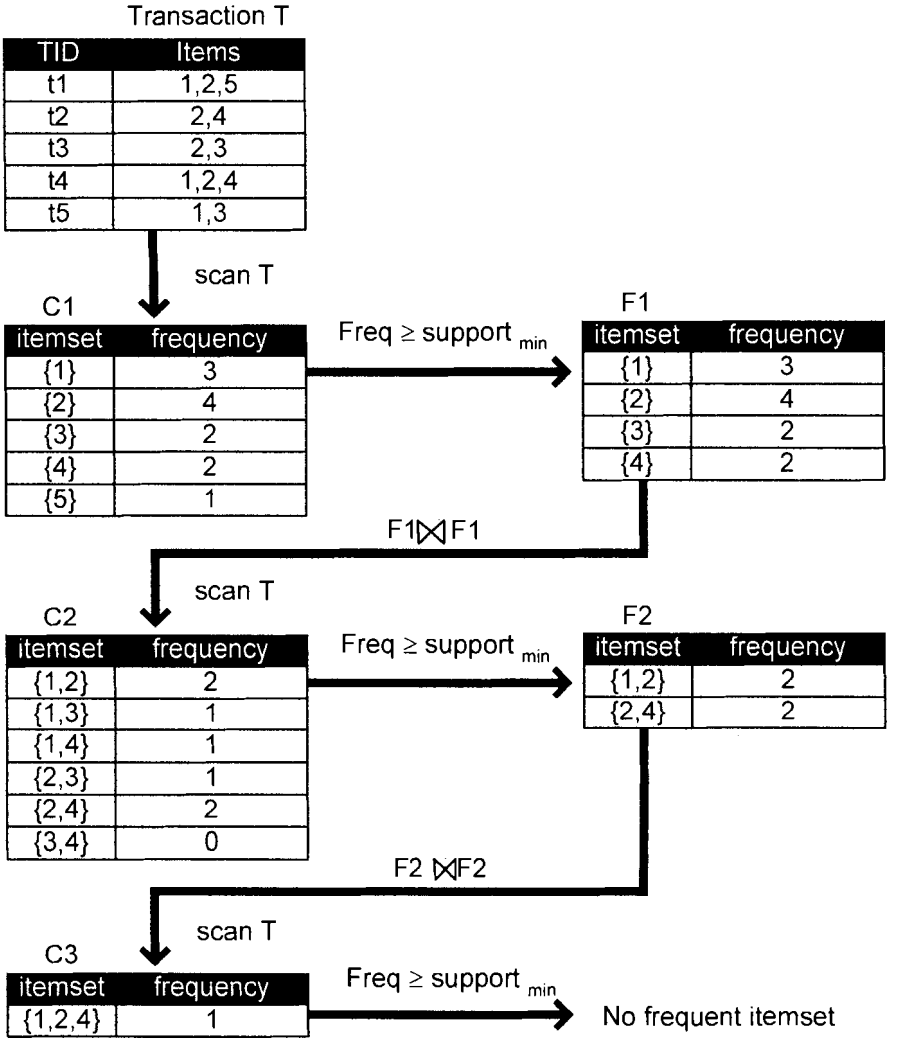

Figure 9 Generation of candidate itemsets and frequent itemsets from transaction.

As a result, the frequent itemsets that have been identified are {1}, {2}, {3}, {4}, {1, 2}, and {2, 4}.

3.2 Association Rule Mining

Item association has been used for years in market basket analysis [Brin et al., 1997a]. If every item in a store is treated as a Boolean variable, each shopping basket can then be represented by a Boolean vector of values assigned to these variables [Hans and Kamber, 2001]. Knowledge of buying patterns can then be obtained through analyzing these Boolean vectors in the form of association rules. In fact, item association is a technique inspired by the association rule data mining model [Agrawal et al., 1993; Mannila et al., 1994; Agrawal et al., 1996]. Very often, association rules are used to uncover the relationships between data items in a database with huge amounts of data. Combining the item association property, the concept of association rule mining can be represented mathematically as:

Let $I = \{1, 2, \ldots, m\}$ be a set of items and $T = \{t_1, t_2, \ldots, t_n\}$ be a database of transactions where $t_i \subseteq I$, and $Support_{min}$ be an expert-defined minimum support count. An association rule is an implication of the form $A \Rightarrow B$ where A, B are itemsets and $A \cap B = \varnothing$. The support(s) for an association rule $A \Rightarrow B$ is the percentage of transactions in the database that contains $A \cup B$. The confidence for an association rule $A \Rightarrow B$ is the ratio of the number of transactions that contain $A \cup B$ to the number of transactions that contain A.

The confidence for an association rule is simply a measure of the rule interestingness and reflects the certainty of discovered rules [Agrawal and Srikant, 1995; Agrawal et al.,1997]. The item association algorithm can then be enhanced to include the formation of association rules (Algorithm 4) as follows:

```
Algorithm:  Association Rules Mining
Input:      transaction database T={t1, t2,…, tn},
            a list of Items I={1,…,m}, and Supportmin
Output:     a set association rules with confidence R
Begin
      All frequent itemsets {F1,…,Fm} =
                  Algorithm-3 (T, I, Supportmin)
      for each frequent i-itemset Fi ∈ {F1,…,Fm}{
         // A ⊆ I and B ⊆ I
         // e.g. if Fi ={1,2,3},
            //A = {{1},{1,2},{1,3},{1,2,3}; same as B
         if (A ⊆ Fi  and B ⊆ Fi and A ∩ B = ∅){
            R ← R ∪ R(A ⇒ B)
            support(A ⇒ B) ← P(A ∪ B)
            confidence(A ⇒ B) ← P(B|A)
         }
      }
      //P(A ∪ B) is the percentage of transactions in
      //T that contain (A ∪ B).

      //P(B|A) is the percentage of transactions in
      //T containing A that also contain B.
End
```

Algorithm 4 Association rule mining.

The example in the previous section can be used here to illustrate association rule mining. A frequent itemset that contains only a single item cannot be used to form an association rule. Since {1}, {2}, {3} are single-itemset, we can translate only {1, 2} and

{2, 4} into the rules. The resulting association rules are as shown below, each listed with its confidence:

First rule : $1 \Rightarrow 2$, **confidence = 2/3 = 66.67%**
Second rule: $2 \Rightarrow 1$, **confidence = 2/4 = 50%**
Third rule: $2 \Rightarrow 4$, **confidence = 2/4 = 50%**
Fourth rule: $4 \Rightarrow 2$, **confidence = 2/2 = 100%**

If we are interested only in rules that have confidence greater than 50% (minimum confidence threshold), the knowledge discovered from the transaction data can be interpreted as: (first rule) when a customer purchases item 1, the chance of him purchasing also item 2 is 66.67%; (last rule) when a customer purchases item 4, the chance of him purchasing also item 2 is 100%.

3.3 Implementation Problems

It can be seen that the association rule mining algorithm suffers from two major costs: space and time [Agrawal and Srikant, 1994; Han and Fu, 1995; Park et al., 1995; Savasere et al., 1995; Toivonen, 1996; Brin et al., 1997b; Silverstein et al., 1998; Aggarwal and Yu, 1999; Agrawal et al., 2000; Han et al., 2000; Park et al., 2000]. The algorithm will generate a huge number of candidate sets before it can identify the frequent itemsets and then the association rules. Besides, the algorithm will scan all transactions repeatedly to perform the frequency counts. There are no trivial solutions

because the algorithm has a running time complexity of an exponential order of growth (2^n) where n is the number of available items. If there are three items a, b, and c, the total number of itemsets will be 2^3-1. The 1-itemsets, 2-itemsets, and 3-itemsets are "{a},{b},{c}"; "{a,b},{a,c},{b,c}"; and "{a,b,c}" respectively. If there are 20 items, the total number of itemsets will be 2^{20}-1.

Although CADIA depends on the item association technique to form its knowledge base, it will not suffer from the exponential growth problem. It is because CADIA identifies only the smallest and least frequent itemsets instead of all frequent itemsets as described in Section 4.1. CADIA's structure and core algorithms are presented in detail in the next chapter.

CHAPTER FOUR:
COMBINATORIAL AUCTION WINNER DETERMINATION USING ITEM ASSOCIATION (CADIA)

4.1 Hypothesis

In many cases, auctions are used to sell items when the auctioneer is unsure about the value of the item being sold. Such an uncertainty regarding values facing both auctioneers and bidders is an inherent feature of auctions [Kelly and Steinberg, 2000; Klemperer, 2000; Lavi and Nisan, 2000; Leyton-Brown et al., 2000b]. The word "auction" itself is derived from the Latin "augere", which means "augment" [Krishna, 2002]. In an open-bid, single-item, first-price auction, the sale is conducted by an auctioneer who begins by calling out a low price and raises it. It continues as long as there are at least two interested bidders and stops when there is only one. In a sealed-bid, single-item, first-price auction, bidders submit bids in sealed envelopes. The person who submits the highest valuation as the bid price for the item will win the item and pays what he bid. When there is a large enough number bidders, we can safely assume that the most wanted item that attracts the largest number of bidders will be sold at the highest valuation among all its valuations. CAs, on the other hand, sell items in bundles instead of one item at a time. Such a form of auction has been believed to be an efficient way for resource allocation. However, the larger numbers of auctioned items and bidders have led to a very complex decision problem. CADIA, which is a CA winner determination

system, is proposed to solve such a problem. The motivation to adopt the item association technique in CADIA is based on the following fact:

> **CA winner determination is a real-world, complex decision problem that involves a large amount of auction data. Item association is good at discovering interesting patterns from large amounts of data. Domain knowledge discovered from auction data in the form of item association can help to solve the problem.**

The following hypothesis is proposed and has been adopted by CADIA when identifying winners. In a single-item auction, if item x is wanted by most bidders, x will be included in most bids. On the contrary, x will be included in very few or no bids if it is not wanted by most bidders. Thus, the number of bids containing the least frequently wanted item (or least frequent item in short) must be less than that containing the most frequently wanted item (or most frequent item in short). Intuitively, an auctioneer may want to sell the least frequent item as early as possible because an unsold item will induce further cost (e.g. storage and handling). Since a higher valuation of an item always implies higher revenue, it is then expected that an auctioneer may sell the most frequent item as late as possible because such an item can always attract the highest possible valuation. Such a belief is relaxed and applied when designing CADIA. In a combinatorial auction, items are valuated as sets. If itemset S is wanted by most bidders, S will be included in most bids. On the contrary, S will be included in very few or no bids if it is not wanted by most bidders. Thus, the number of bids containing the least frequent itemset must be less than that containing the most frequent itemset. An auctioneer may be

tempted to sell the least frequent itemset as early as possible and the most frequent itemset as late as possible. The least frequent itemset is defined as follows:

> **A set of items is called an itemset. An itemset that contains k items is a k-itemset (Section 3.2). The least frequent k-itemset is an itemset whose number of occurrences is the smallest among all frequent k-itemsets.**

In a single-item auction, a tie happens when the item is wanted by more than one bid. The resolution strategy is simply to assign the bid with a highest bidding price as the winner for that item. When itemsets are considered in the design of CADIA, an assumption that a bid containing the least frequent itemset is having less conflict than one containing the most frequent itemset is made. It is understood that such an assumption may not be justified theoretically and may cause error in winner determination. However, adjustments have been made to minimize such errors and are described in Chapter 6.

Another issue that must be considered when designing CADIA is the identification of the least frequent itemsets. If frequency count is the only measure used, it is possible to have more than one itemset whose frequency counts are the same. In such a situation, additional measures such as the degree of confidence and the degree of conflict, which are defined below, will be applied.

Suppose item x and y are in the same itemset; the degree of confidence is either the ratio of the number of bids that contain x and y to the number of bids that contain x, or the ratio of the number of bids that contain x and y to the number of bids that contain y, whichever is higher.

Bid b_i conflicts with any other bids if there is an item wanted by b_i that is also wanted by other bids. Suppose S_i is the set of items wanted by bid b_i and C is the itemset of maximum confidence where $C \subseteq S_i$. The degree of conflict of C is the total number of bids that conflicts with b_i.

The least frequent itemsets are used in winner determination. It is possible to have tied bids (candidate winners) if the least frequent itemset is wanted by more than one bid. The tie resolution strategy adopted by CADIA is to award the candidate winner that offers the highest bidding price as the final winner. If there is a further tie on the bidding price, the candidate winner that is submitted at the earliest time becomes the final winner. Bids are assigned with a number based on their submission time in CADIA. The lower the number, the earlier the time the bid was submitted. Thus, b_0 is submitted at an earlier time than b_1. The hypothesis can be summarized as:

To maximize revenue in a CA, the bids containing the least frequent itemset should be processed first and are declared as candidate winners during each iteration of the winner determination process. When identifying the least frequent itemsets, the measures of frequency count, degree of confidence, and degree of conflict are

compared. The bid among all candidate winners that offers the highest bidding price becomes the winner. In the case of a tie on the bidding price, the bid that was submitted at the earliest time becomes the winner.

4.2 Structure

Figure 10 depicts the structure of CADIA, which is composed of two major components. They are:

1. Item Association Generation Unit (IAG)
2. Winner Determination Unit (WIN)

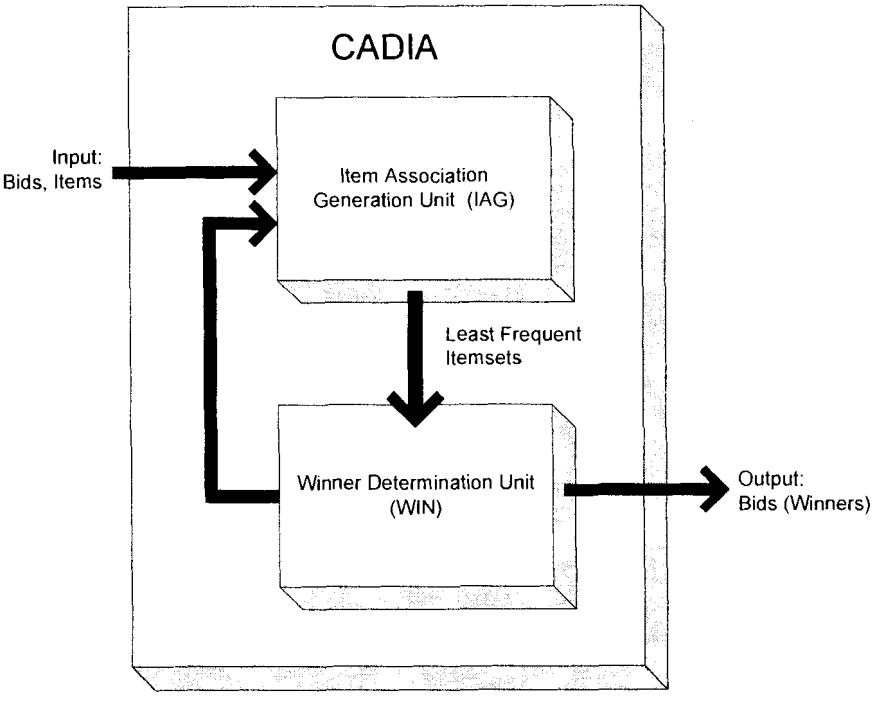

Figure 10 Structure of CADIA.

4.2.1 Item Association Generation Unit (IAG)

The Item Association Generation Unit (IAG) is used to form the knowledge in the form of item association on which the winner determination strategy is based. Instead of looking for all frequent itemsets that satisfy the minimum support count as in most association rule mining applications, IAG attempts to identify only the least frequent itemsets according to the hypothesis stated in section 4.1. Consequently, IAG does not have the inherent problem of time and space complexities as discussed in Section 3.3. That is, even if there are many itemsets whose counts are greater than the minimum support count, IAG looks for only those that have frequency count equal to the minimum support count. In addition, IAG identifies only the smallest frequent itemsets instead of all frequent itemsets. That is, even if there are hundreds of items, IAG always start counting frequent 1-itemsets, followed by 2-itemsets, 3-itemsets, and so on. IAG's default value for the minimum support is assigned to 1, which is the smallest non-zero integer. Our empirical results show that IAG always returns the least frequent itemsets before it generates the 3-itemsets. Whenever there is a tie, the degrees of confidence and conflict of all least frequent itemsets are compared. In other words, CADIA will identify the smallest and least frequent itemset with the highest degree of confidence and the lowest degree of conflict at IAG. Algorithm 5 and 6 are the core algorithms adopted by IAG.

```
Algorithm:  Identifying the smallest and least frequent
            itemsets during each iteration
Input:      all available items M={1,2,…,m},
            all available bids B={b0,b1,…,bn},
            minimum support count (Support_min)
Output:     the smallest and least frequent itemsets F_sl
Begin
      found ← FALSE
      candidate 1-itemsets C1 ← M
      for each itemset cj ∈ C1
         if (frequency(cj) = Support_min){
            C_sl ← C_sl ∪ cj
            C_sl = highestConfidenceItemset(C_sl)
            F_sl = lowestConflictItemset(C_sl)
            found ← TRUE
          }
      if (found = TRUE) //if LFI is found in 1-itemset
         return F_sl

      //2- or higher level itemsets must be generated
      //before frequency counting
      for i ∈ {2..m} {
         Generate candidate i-itemsets Ci by F_i-1 × F_i-1
         for each itemset cj ∈ Ci {
            if (frequency(cj) = Support_min) {
               C_sl ← C_sl ∪ cj
               C_sl = highestConfidenceItemset(C_sl)
               F_sl = lowestConflictItemset(C_sl)
               found ← TRUE
             }
             if (found = TRUE)
                return F_sl
         }
      }
End


Function frequency (c)
Begin
      return (number of transactions that contain c)
End
```

Algorithm 5 Identify the smallest and least frequent itemset

```
Function highestConfidenceItemset (C)
Begin
      highestConfidence ← 0
      highestConfdItemset ← ∅
      for each itemset c_i ∈ C
          // A ⊆ I and B ⊆ I
         if (A ⊆ c_i  and B ⊆ c_i and A ∩ B = ∅){
            confidence(A,B) ← P(B|A)
            if (highestConfidence <= confidence(A,B)){
               highestConfidence ← confidence(A,B)
               highestConfdItemset ← highestConfdItemset ∪ c_i
            }
         }
      return highestConfdItemset
End


Function leastConflictItemset (C)
Begin
      leastConflict ← very large constant
      leastConflictItemset ← ∅
      for each itemset c_i ∈ C {
         conflictCount ← 0
         find b_i ∈ B where c_i ⊆ S_i
         for each bid b_j ∈ B where j≠i {
            if (S_j ∩ S_i ≠ ∅)
                conflictCount = conflictCount + 1
         }
         if (leastConflict >= conflictCount){
            leastConflict ← conflictCount
            leastConflictItemset ← c_i
         }
      }
      return leastConflictItemset
End
```

Algorithm 6 Functions highestConfidenceItemset and leastConflictItemset

4.2.2 Winner Determination Unit (WIN)

CADIA can be treated as an informed search system because its Winner Determination Unit (WIN) uses problem-specific knowledge in the form of item association to look for solutions. Since it is possible to have multiple winners during a CA, WIN uses the least frequent itemset output from IAG in its candidate winner determination process. Given the least frequent itemset, WIN identifies those bids containing the itemset as candidate winners. A conflict exists when there is more than one bid containing the least frequent itemset. Conflicts among bids are resolved by WIN using the following measures:

1. **The bid which offers the highest bidding price becomes a winner.**
2. **If there is a tie on bidding price, the bid which is submitted at the earliest time becomes a winner.**

Algorithm 7 iterates over all bids to identify the candidate winners. Algorithm 8 identifies the winner from all candidate winners based on the bidding price and bid submission time.

```
Algorithm:  Identify all candidate winners
Input:      the least frequent itemset, Ss ,
            all bids bi ∈B and bid tuples {Si,pi}, i∈{0,1,…n}.
Output:     a list of all candidate winners, L.

Begin
      for each bid bi∈B {
         if Si ⊇ Ss
            L ← L ∪ bi
      }
      return L;
End
```

Algorithm 7 Identify all candidate winners.

```
Algorithm:  Identify a winner
Input:      all candidate winners, Bc
            all bids bi ∈B and bid tuples {Si,pi}, i∈{0,1,…n}
Output:     a winner, winneri

Begin
      highestPrice ← 0
      for each bid bi∈Bc {
         if (pi>highestPrice){
            winneri ← bi
            highestPrice ← pi
         }
      }
      return winneri
End
```

Algorithm 8 Identify a winner

4.3 Example

A simple example is presented here to illustrate the CADIA's core concept and algorithms. The sample data contains 10 auctioned items and 10 bids as presented in Figure 11. At the beginning, CADIA will read the bid data as inputs and organize them into a matrix in the memory as described in Figure 12.

{bid}	{a set of items}	{bidding price}
$\{b_0\}$	{0, 4, 6, 7}	{206.28}
$\{b_1\}$	{0, 1, 3, 4}	{207.28}
$\{b_2\}$	{0, 6}	{205.00}
$\{b_3\}$	{0, 4, 5, 9}	{208.28}
$\{b_4\}$	{2, 4, 5, 8}	{108.28}
$\{b_5\}$	{1, 2, 7}	{55.74}
$\{b_6\}$	{1, 2, 3, 6}	{55.74}
$\{b_7\}$	{2, 9}	{152.00}
$\{b_8\}$	{0, 4, 8}	{154.74}
$\{b_9\}$	{0, 4, 6, 7, 8}	{205.50}

Figure 11 Bid data.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	
b4			x		x	x			x		108.28	
b5		x	x					x			55.74	
b6		x	x	x			x				55.74	
b7			x							x	152.00	
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	

Figure 12 Auction data is represented internally as a matrix in CADIA.

During the first step, IAG of CADIA sets the minimum support count to one (a non-zero least support count), starts generating frequent 1-itemsets (Figure 13), and checks if there are itemsets whose frequency counts are equal to but not greater than the minimum support count. That is, IAG identifies the smallest but also least frequent itemset from all bids as described in Section 4.2.2. In this example, all ten frequent 1-itemsets have frequency counts greater than the minimum support count. Thus, IAG is required to generate frequent 2-itemsets (Figure 14). Now, twenty-one out of forty-five itemsets have frequency counts equal to the minimum support count. Additional measures such as the degree of confidence and the degree of conflict will then be applied according to algorithm 5 and 6 to reduce the total number of itemsets. It is shown in Figure 15 and 16 that the highest degree of confidence and the lowest degree of conflict are found to be 50% and 5 respectively. As a result, the itemset {2,9} is determined as the least frequent itemset, which will be used by WIN to determine candidate winners.

itemset	frequency
{0}	6
{1}	3
{2}	4
{3}	2
{4}	6
{5}	2
{6}	4
{7}	3
{8}	3
{9}	2

Figure 13 Frequent 1-itemsets during the first iteration.

itemset	frequency	itemset	frequency	itemset	frequency
{0,1}	1	{1,8}	0	{4,5}	2
{0,2}	0	{1,9}	0	{4,6}	2
{0,3}	1	{2,3}	1	{4,7}	2
{0,4}	5	{2,4}	1	{4,8}	3
{0,5}	1	{2,5}	1	{4,9}	1
{0,6}	3	{2,6}	1	{5,6}	0
{0,7}	2	{2,7}	1	{5,7}	0
{0,8}	2	{2,8}	1	{5,8}	1
{0,9}	1	{2,9}	1	{5,9}	1
{1,2}	2	{3,4}	1	{6,7}	2
{1,3}	2	{3,5}	0	{6,8}	1
{1,4}	1	{3,6}	1	{6,9}	0
{1,5}	0	{3,7}	0	{7,8}	1
{1,6}	1	{3,8}	0	{7,9}	0
{1,7}	1	{3,9}	0	{8,9}	0

Figure 14 Frequent 2-itemsets during the first iteration

itemset	confidence(%)	itemset	confidence(%)	itemset	confidence(%)
{0,1}	33.33	{2,3}	50.00	{3,4}	50.00
{0,3}	50.00	{2,4}	25.00	{3,6}	50.00
{0,5}	50.00	{2,5}	50.00	{4,9}	50.00
{0,9}	50.00	{2,6}	25.00	{5,8}	50.00
{1,4}	33.33	{2,7}	33.33	{5,9}	50.00
{1,6}	33.33	{2,8}	33.33	{6,8}	33.33
{1,7}	33.33	{2,9}	50.00	{7,8}	33.33

Figure 15 Degrees of confidence for itemsets with the least support count

itemset	confidence(%)	conflict	itemset	confidence(%)	conflict
{0,3}	50.00	9	{3,4}	50.00	9
{0,5}	50.00	8	{3,6}	50.00	8
{0,9}	50.00	8	{4,9}	50.00	8
{2,3}	50.00	8	{5,8}	50.00	9
{2,5}	50.00	9	{5,9}	50.00	8
{2,9}	50.00	5			

Figure 16 Degrees of conflict for itemsets with highest confidence.

In the next step, WIN of CADIA will identify all candidate winners. WIN starts looking for those bids that include the least frequent itemset {2,9}. In this example, only b_7 contains itemset {2,9}. Consequently, b_7 is determined as a winner (Figure 17). After the winner is declared, those bids that conflict with it are labelled as losers (Figure 18). Since we can have multiple winners in an auction, we can only say that b_7 is one of the winners and b_3, b_4, b_5 and b_6 are losers during the first iteration.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	
b4			x		x	x			x		108.28	
b5		x	x					x			55.74	
b6		x	x	x			x				55.74	
b7			x							x	152.00	winner
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	

Figure 17 Bid b_7 becomes a winner after the first iteration.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	

Figure 18 Bid b_3, b_4, b_5, b_6 become losers after the first iteration.

In the second iteration, the available qualified bids are b_0, b_1, b_2, b_8 and b_9. IAG will update the item association pattern based these bids. IAG again sets the minimum support count to one, generates frequent 1-itemsets (Figure 19), and checks if there are itemsets whose frequency counts are equal to the minimum support count. In addition, the degree of confidence and the degree of conflict are checked for each itemset. It is shown in Figure 20 that the highest degree of confidence and the lowest degree of conflict are found to be 100% and 5 respectively. Since both itemsets {1} and {3} have satisfied the condition of being the least frequent itemsets, they are used for candidate winners determination.

itemset	frequency
{0}	5
{1}	1
{2}	0
{3}	1
{4}	4
{5}	0
{6}	3
{7}	2
{8}	2
{9}	0

Figure 19 Frequent 1-itemsets during the second iteration.

itemset	confidence(%)	conflict
{1}	100.00	5
{3}	100.00	5

Figure 20 Degrees of conflict for itemsets with maximum confidence during the second iteration.

In candidate winners determination, WIN looks for those bids that include the least frequent itemset {1} or {3}. As a result, b_1 is determined to be a winner during the second iteration (Figure 21). After the winner has been declared, all bids that conflict with it are determined as losers (Figure 22). After the second iteration, WIN stops because all bids have been processed. As a result, b_1 and b_7 are the winners which generate the total revenue of $359.28. In fact, such revenue is the optimal revenue for this sample problem.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	winner
b2	x						x				205.00	
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	

Figure 21 Bid b_1 becomes a winner after the second iteration.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	loser
b1	x	x		x	x						207.28	winner
b2	x						x				205.00	loser
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	loser
b9	x				x		x	x	x		205.50	loser

Figure 22 Bids b_0, b_2, b_8, b_9 become losers after the second iteration.

CHAPTER FIVE:
EVALUATION (I)

The following plan is used for CADIA evaluation:

1. Understand the evaluation purpose.
2. Set up the experiments.
3. Select a sample of inputs.
4. Implement a prototype CADIA.
5. Run CADIA on the sample's input and record the results.
6. Summarize and analyze the results.

5.1 Purpose

The purpose of the evaluation is to evaluate CADIA's accuracy and performance. The evaluation of CADIA's accuracy is straightforward. To conclude that CADIA is a technique capable of finding the optimal solution, the revenue generated by CADIA during an auction must be equal to that generated by an optimal revenue search system such as the brute-force technique based system. In CADIA's performance evaluation, we may be tempted to use mathematical analysis. Though mathematical analysis can be applied to many simple algorithms, the power of mathematics is still far from limitless. Most heuristic techniques that solve the class NP problems are believed to be very

difficult to analyze with mathematical precision and certainty [Goodrich and Tamassia, 2002; Johnsonbaugh and Schaefer, 2004]. Thus, empirical analysis [Levitin, 2003] is adopted in CADIA's evaluation.

The two major approaches of analyzing an algorithm empirically are:

1. **Count the number of times the algorithm's basic operation is executed by inserting a counter in the algorithm.**
2. **Time the algorithm.**

CADIA's core algorithm requires an update of its item association knowledge in each of the iterations of the winner determination process. The process is so dynamic that the first approach of counting the number of operations becomes inappropriate. In addition, CADIA's implementation is in fact a combination of many algorithms. Thus, the second approach of timing the prototype of CADIA is used. Since CADIA is implemented in the C programming language, the built-in system function "clock( )[3]" has been used to return the start time T_{start} and the finish time T_{finish}. The running time required, which has been converted into seconds, is equal to the difference between T_{start} and T_{finish}.

[3] clock() returns wall-clock time used by the calling process.
[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_clock.asp]

5.2 Combinatorial Auction Testing Suite

Many researchers have recently begun to propose algorithms for determining the winners of CAs, with encouraging results. This wave of research has given rise to a new problem, however. In the absence of real world CA data, the only option is to generate auction data artificially. However, it is necessary to use a standard test suite with simulated data to test and improve the proposed algorithms. A test suite called Combinatorial Auction Test Suite (CATS) for testing combinatorial auction algorithms has been proposed and developed by Leyton-Brown et al. [2000a]. CATS includes the ability to generate bids according to all previous published test distributions and has been used in a number of recent papers [Sandholm et al., 2001b; Sandholm, 2002]. In CADIA's evaluation, CADIA is tested on CATS's arbitrary distribution. All sample auctions are generated using CATS instance generators with default parameters.

5.3 Experimental Setup

The test implementation of CADIA in the C programming language running on a 1GHz Pentium PC with 512MB RAM was evaluated on auction data generated by CATS. Five hundreds sample auctions were generated using CATS. Two different tests were performed in the evaluation. The objective of the first test was to justify the conclusion that CADIA is capable of finding the optimal revenue. Thus, CADIA was compared with the brute-force technique (BFT) based system in terms of revenue generation and running time. The objective of the second test is to justify the conclusion that CADIA is a good

approximation system that always guarantees a better than or equal to the lower bound revenue. Thus, CADIA was compared with an implementation that is based on the greedy search technique (GST). The bidding price is adopted as the objective function for the GST.

The software implementation of CADIA takes four parameters during its execution. They are:

1. **a file containing the list of auctioned items,**
2. **a file containing the bid data,**
3. **an integer corresponding to the number of items in the auction,**
4. **an integer corresponding to the number of bids in the auction**

CADIA can be executed from the command line as shown in Figure 23. The command says that CADIA will determine the winners in a CA of 500 items and 1000 bids.

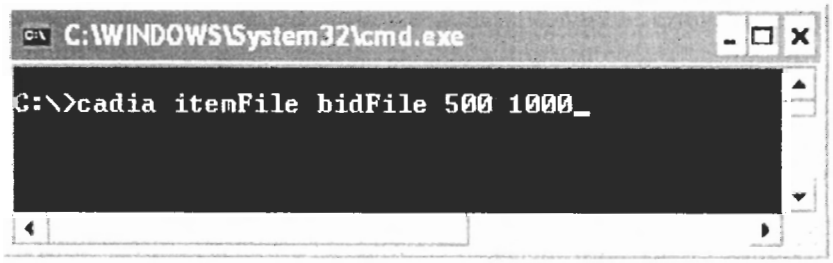

Figure 23 Execute CADIA with 4 arguments from command line.

5.3.1 Comparison of CADIA and BFT

The objective of the first test is to justify the conclusion that CADIA is capable of finding the optimal revenue. Due to the fact that the CA winner determination problem is

a NP-hard problem, the test becomes realistic only if the sample auctions contain a reasonably small number of bids items. Thus, CADIA was compared with BFT in terms of revenue generation and running time on two hundred sample auctions with ten items and ten bids each. BFT is also implemented in the C programming language based on the algorithm described in Algorithm 1.

5.3.2 Comparison of CADIA and GST

The objective of the second test was to justify the conclusion that CADIA is a good approximation system that always guarantees a better than or equal to the lower bound revenue. Since GST is based on greedy search technique, it can handle more items and bids that cannot be handled by BFT and always returns feasible results in reasonable time. Thus, CADIA was compared with GST in terms of revenue generation on two hundred sample auctions with twenty items and one thousand bids each. GST is also implemented in the C programming language based on the algorithm described in Algorithm 2.

5.4 Empirical Results and Analysis

In this section, the results of the tests described in Section 5.3.1 and 5.3.2 are documented in Section 5.4.1 and 5.4.2 respectively. The results will be used to justify the hypothesis in designing CADIA.

5.4.1 Comparison of CADIA and BFT

Even BFT works in principle, however it is practically limited by the number of items and bids it can process. CADIA may or may not find the optimal revenue. Thus, it is interesting to know how accurate CADIA is. The accuracy of CADIA can be quantified by the size of the accuracy ratio $R_{accuracy}$ [Levitin, 2003] of CADIA where S_{CADIA} and S_{BFT} represent the solutions of CADIA and BFT respectively to the objective function f of the winner determination problem (i.e. the revenue). The closer $R_{accuracy}$ is to 1, the better the proposed technique is.

$$R_{accuracy} = \frac{f(S_{CADIA})}{f(S_{BFT})}$$

Table 1 reports the results for all two hundred sample auctions. The accuracy ratio of BFT in all sample auctions is always one because BFT always finds the optimal revenue and thus is used as the standard for comparison. CADIA has an average accuracy ratio of 0.979 and is not able to find the optimal revenue in 36 out of 200 auctions. The results are sorted and plotted as a line chart (Figure 24). In addition, the average running time of BFT and CADIA are 211.722 seconds and 0.199 seconds respectively.

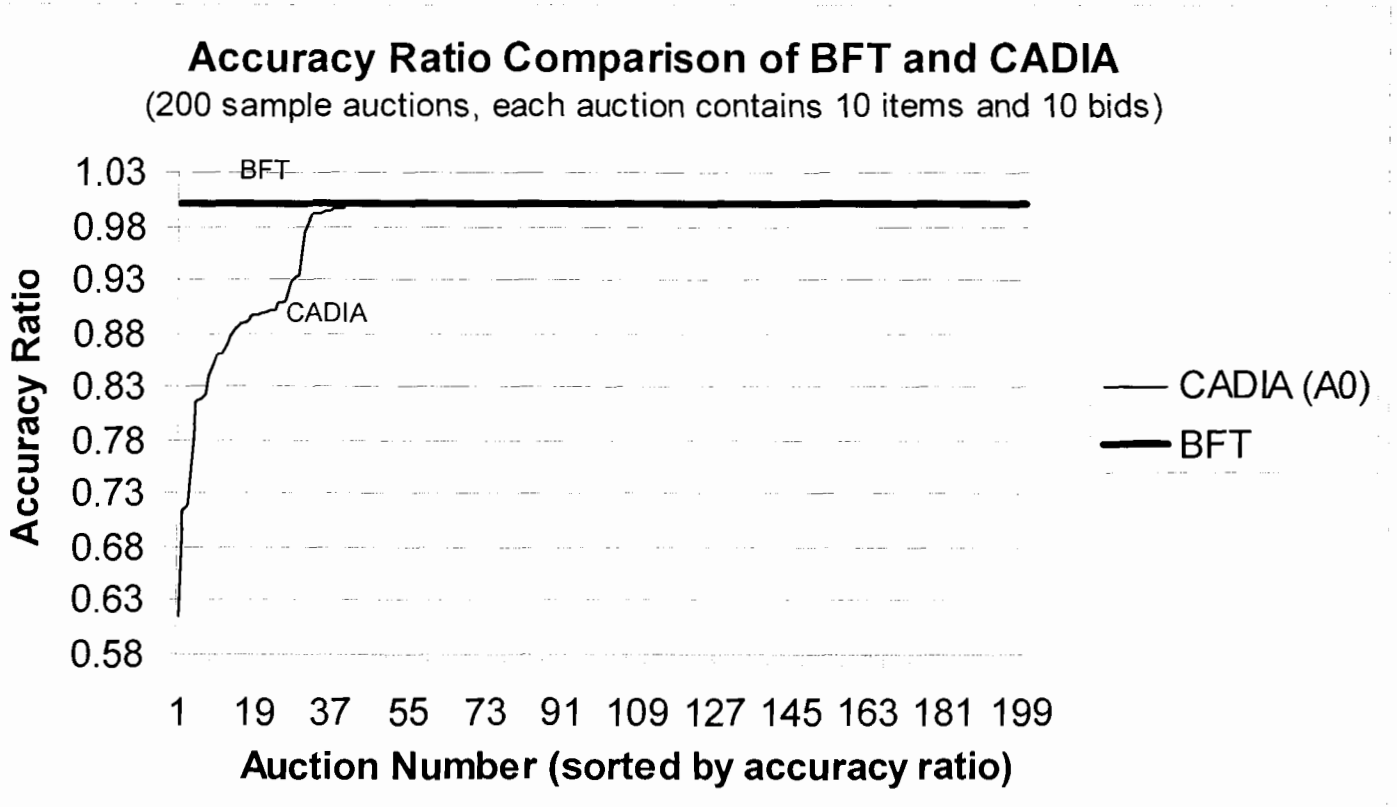

Figure 24 Accuracy ratio comparison of BFT and CADIA.

AUCTION NUMBER	CADIA	AUCTION NUMBER	CADIA	AUCTION NUMBER	CADIA	AUCTION NUMBER	CADIA
1	1.000	51	1.000	101	1.000	151	0.840
2	1.000	52	1.000	102	1.000	152	1.000
3	0.719	53	1.000	103	1.000	153	0.859
4	1.000	54	1.000	104	1.000	154	1.000
5	1.000	55	0.996	105	0.928	155	1.000
6	1.000	56	1.000	106	1.000	156	1.000
7	1.000	57	1.000	107	1.000	157	1.000
8	0.715	58	1.000	108	1.000	158	1.000
9	1.000	59	1.000	109	1.000	159	0.899
10	0.877	60	1.000	110	0.986	160	1.000
11	1.000	61	1.000	111	1.000	161	1.000
12	1.000	62	1.000	112	0.908	162	1.000
13	0.819	63	1.000	113	0.934	163	1.000
14	1.000	64	1.000	114	1.000	164	0.614
15	1.000	65	1.000	115	0.933	165	1.000
16	1.000	66	1.000	116	1.000	166	1.000
17	1.000	67	1.000	117	1.000	167	1.000
18	1.000	68	1.000	118	0.901	168	1.000
19	1.000	69	1.000	119	1.000	169	1.000
20	0.817	70	1.000	120	1.000	170	1.000
21	1.000	71	1.000	121	1.000	171	1.000
22	1.000	72	1.000	122	1.000	172	1.000
23	1.000	73	0.777	123	0.991	173	1.000
24	1.000	74	0.990	124	1.000	174	1.000
25	1.000	75	0.910	125	1.000	175	1.000
26	1.000	76	1.000	126	0.897	176	1.000
27	0.890	77	1.000	127	0.889	177	0.898
28	1.000	78	0.900	128	1.000	178	1.000
29	1.000	79	1.000	129	0.885	179	0.994
30	1.000	80	1.000	130	1.000	180	1.000
31	0.870	81	1.000	131	1.000	181	1.000
32	1.000	82	1.000	132	1.000	182	1.000
33	1.000	83	1.000	133	0.892	183	1.000
34	1.000	84	1.000	134	1.000	184	0.992
35	0.824	85	1.000	135	1.000	185	1.000
36	1.000	86	0.908	136	1.000	186	1.000
37	1.000	87	1.000	137	1.000	187	1.000
38	1.000	88	1.000	138	1.000	188	1.000
39	1.000	89	0.995	139	1.000	189	1.000
40	1.000	90	1.000	140	0.859	190	0.992
41	1.000	91	1.000	141	1.000	191	1.000
42	1.000	92	1.000	142	0.896	192	1.000
43	0.974	93	1.000	143	1.000	193	0.996
44	1.000	94	1.000	144	1.000	194	1.000
45	1.000	95	1.000	145	1.000	195	1.000
46	0.851	96	1.000	146	1.000	196	1.000
47	1.000	97	1.000	147	1.000	197	1.000
48	1.000	98	1.000	148	1.000	198	1.000
49	1.000	99	1.000	149	1.000	199	1.000
50	1.000	100	1.000	150	1.000	200	1.000

Table 1 Accuracy ratio comparison of BFT and CADIA (sample 1-200).

5.4.2 Comparison of CADIA and GST

When there are too many items and bids, it becomes impractical to compare BFT and CADIA. BFT takes more than 200 seconds to process an auction of ten items and ten bids, but requires about 1800 seconds to process an auction with one additional item. Since it is worthwhile to measure CADIA's performance when there are hundreds of items and thousands of bids, CADIA is compared with GST in terms of revenue generation. The performance of CADIA in revenue generation can be quantified by the size of the performance ratio $R_{performance}$ of CADIA where S_{CADIA} and S_{GST} represent the solutions of CADIA and GST respectively to the objective function f of the winner determination problem (i.e. the revenue). The higher the value of $R_{performance}$, the better the performance of the proposed technique.

$$R_{performance} = \frac{f(S_{CADIA})}{f(S_{GST})}$$

In this test, each sample auction contains twenty items and one thousand bids. The number of items has been selected in such a way that it cannot be handled realistically by BFT, but it is still small enough as compared to the number of bids. The purpose of such a setup is to simulate realistic CAs in which there are always conflicts among bids. Table 2 summarizes the performance ratios of GST and CADIA for two hundred sample auctions. The accuracy ratio of GST in all sample auctions is always one because it is used as the standard for comparison.

AUCTION NUMBER	CADIA	AUCTION NUMBER	CADIA	AUCTION NUMBER	CADIA	AUCTION NUMBER	CADIA
1	1.238	51	1.481	101	1.117	151	1.057
2	1.053	52	1.658	102	1.108	152	1.204
3	1.000	53	1.339	103	1.060	153	1.152
4	1.126	54	1.171	104	1.322	154	1.000
5	1.000	55	1.236	105	1.202	155	1.145
6	1.167	56	1.273	106	1.448	156	1.191
7	1.116	57	1.000	107	1.053	157	1.110
8	1.043	58	1.359	108	1.167	158	1.272
9	1.000	59	1.202	109	1.218	159	1.255
10	1.041	60	1.118	110	1.057	160	1.000
11	1.095	61	1.269	111	1.051	161	1.057
12	1.323	62	1.110	112	1.283	162	1.105
13	1.130	63	1.168	113	1.300	163	1.158
14	1.044	64	1.225	114	1.057	164	1.239
15	1.168	65	1.102	115	1.284	165	1.046
16	1.113	66	1.121	116	1.410	166	1.160
17	1.179	67	1.102	117	1.105	167	1.152
18	1.113	68	1.214	118	1.225	168	1.159
19	1.200	69	1.207	119	1.168	169	1.218
20	1.048	70	1.306	120	1.061	170	1.179
21	1.106	71	1.221	121	1.172	171	1.759
22	1.128	72	1.255	122	1.228	172	1.304
23	1.120	73	1.388	123	1.105	173	1.000
24	1.301	74	1.114	124	1.342	174	1.256
25	1.157	75	1.319	125	1.061	175	1.137
26	1.000	76	1.671	126	1.342	176	1.274
27	1.276	77	1.298	127	1.048	177	1.108
28	1.053	78	1.182	128	1.317	178	1.192
29	1.121	79	1.191	129	1.238	179	1.268
30	1.048	80	1.221	130	1.193	180	1.255
31	1.056	81	1.190	131	1.299	181	1.118
32	1.178	82	1.256	132	1.360	182	1.048
33	1.255	83	1.495	133	1.105	183	1.401
34	1.129	84	1.048	134	1.111	184	1.159
35	1.040	85	1.113	135	1.195	185	1.131
36	1.099	86	1.240	136	1.056	186	1.118
37	1.321	87	1.048	137	1.556	187	1.179
38	1.189	88	1.154	138	1.228	188	1.480
39	1.229	89	1.169	139	1.189	189	1.134
40	1.168	90	1.100	140	1.226	190	1.202
41	1.179	91	1.256	141	1.323	191	1.274
42	1.000	92	1.272	142	1.217	192	1.092
43	1.086	93	1.255	143	1.040	193	1.293
44	1.229	94	1.359	144	1.659	194	1.051
45	1.094	95	1.051	145	1.181	195	1.095
46	1.044	96	1.111	146	1.171	196	1.191
47	1.054	97	1.169	147	1.051	197	1.226
48	1.105	98	1.171	148	1.146	198	1.180
49	1.301	99	1.242	149	1.147	199	1.273
50	1.118	100	1.239	150	1.131	200	1.111

Table 2 Performance ratio comparison of GST and CADIA (sample 1-200).

The data in the table is then used to form a line chart (Figure 25). Results show that CADIA has an average performance ratio of 1.186 and outperforms GST in 191 out of 200 auctions. In other words, CADIA on average outperforms GST by 18.6% in our evaluation.

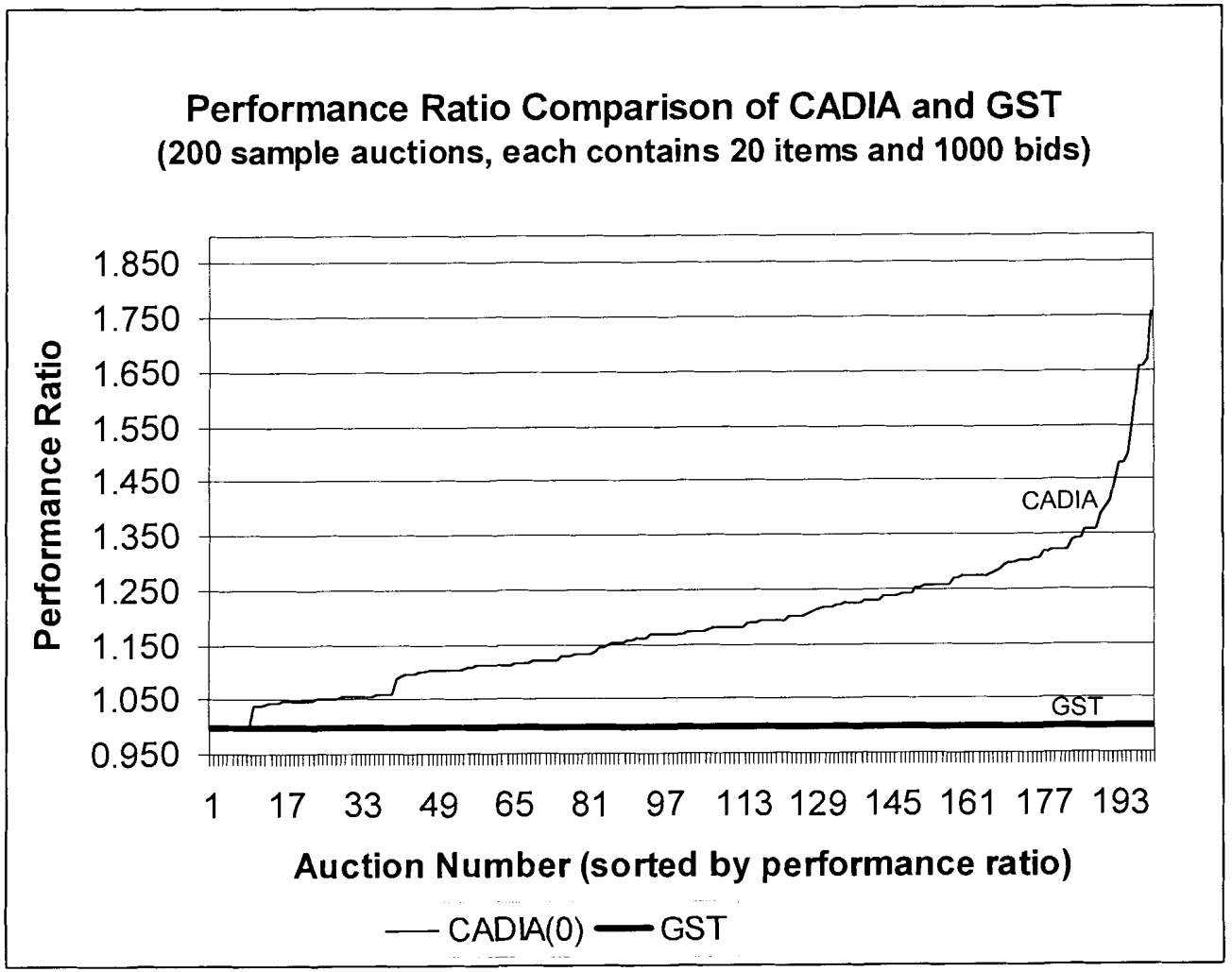

Figure 25 Performance ratio comparison of GST and CADIA.

CHAPTER SIX:
IMPROVING CADIA

6.1 Motivation

It has been shown in Section 5.4.1 that CADIA has an average accuracy ratio of 0.979 for the two hundred sample auctions. That is, CADIA may or may not find the optimal revenue. Such a defect is due to the fact that the comparison of the valuation of an itemset wanted by bids is relaxed to a comparison of the valuation of all the items wanted by bids. Suppose the bid tuples of bids b_i and b_j are (S_i, p_i) and (S_j, p_j) respectively and $S_i \subseteq M$, $S_j \subseteq M$ and $p_i > p_j$. If there is a least frequent itemset F where $F \subseteq S_i$ and $F \subseteq S_j$, b_i will conflict with b_j. Based on the hypothesis in Section 4.1, b_i will become the winner because it offers a higher valuation on S_i. The comparison is actually based on the prices for S_i and S_j but not on F offered by b_i and b_j. The ideal situation would be for each bidder to submit a valuation for each subset of auctioned items in order to attain true valuation comparison as suggested by the Vickrey-Clarke-Grooves (VCG mechanism) [Klemperer, 2000; Krishna, 2002]. However, the VCG mechanism is impractical and rarely used because no bidder is willing to valuate all subsets of auctioned items [Pekeč and Rothkopf, 2000]. Even if there are only 20 auctioned items, it is not likely every bidder is willing to work out 2^{20}-1 or 1048574 valuations. In spite of the relaxation on valuation, CADIA is able to minimize or even correct the error via an adjustment. Instead of immediately declaring a bid b_i as a winner based on the item association technique,

CADIA first declares b_i as a potential winner. It then measures the revenue generated when b_i is not a winner. Comparing the revenue generated with and without b_i, it is possible to improve the revenue via the selection of a new set of winners. Such an adjustment procedure is performed at the Tactical Bids Elimination (TBE) component of CADIA, which is described in detail in Section 6.2.2. TBE becomes an additional component to make CADIA a system capable of finding the optimal revenue.

When improving CADIA, the performance affected by having redundant bids has been taken into consideration. For instance, a bid that bids on the same combination of items as others but offers a lower bidding price should be removed. The additional component Pre-Processing Unit (PRE), which is described in detail in Section 6.2.1, is thus added to CADIA to remove redundant bids.

6.2 New Structure

With the additional components, CADIA can be described as an "aggressive" system because it improves its results on successive iterations during the winner determination process.

Figure 26 depicts the new structure of CADIA, which is composed of four major components. They are:

1. **Pre-Processing Unit (PRE)**
2. **Item Association Generation Unit (IAG)**
3. **Winner Determination Unit (WIN)**
4. **Tactical Bids Elimination Unit (TBE)**

The four components form the four consecutive phases of the winner determination process. Outputs from one component may flow back to a previous component during the process. For example, outputs from TBE will flow back to PRE to further improve the results.

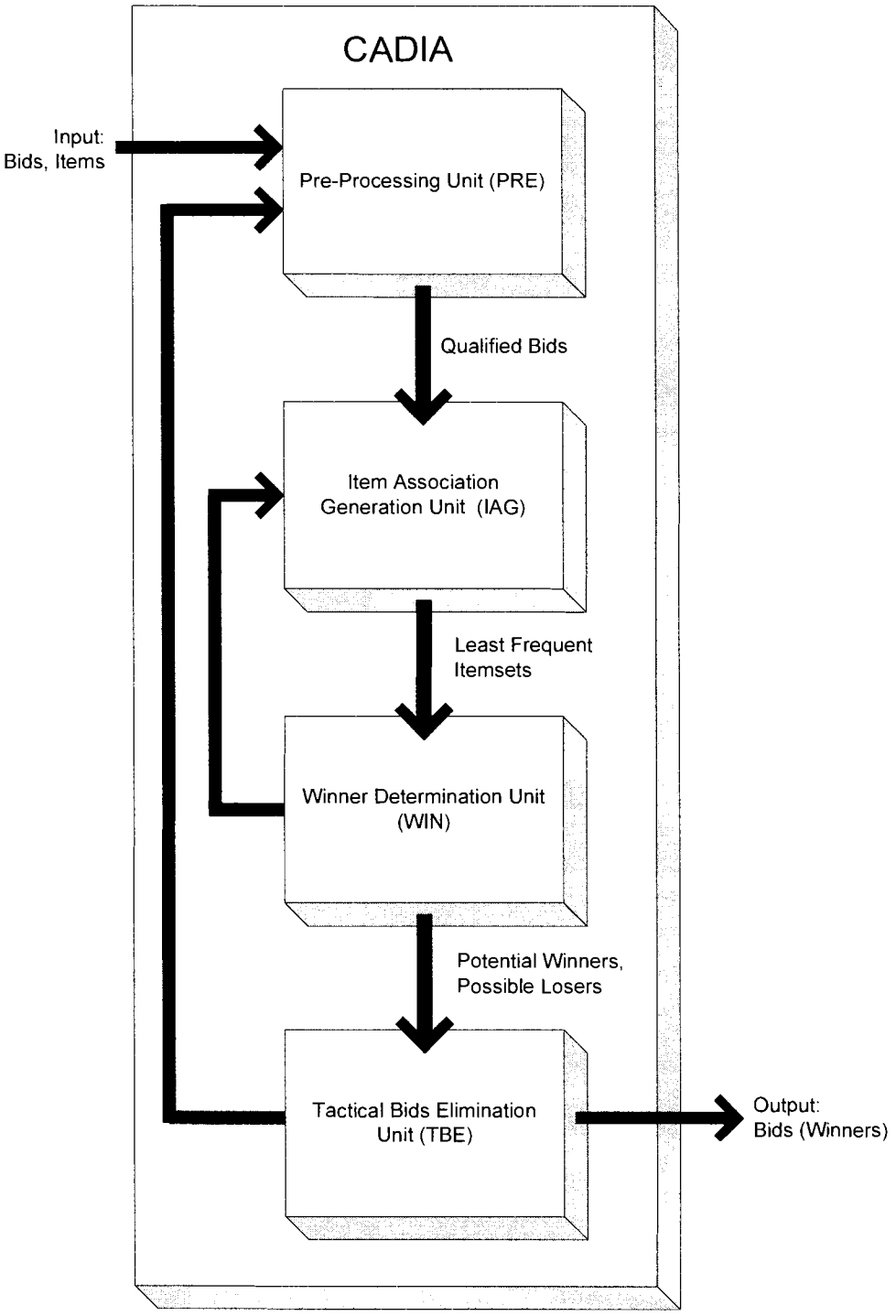

Figure 26 Structure of CADIA.

6.2.1 Pre-Processing Unit (PRE)

The Pre-Processing Unit (PRE) is used to remove redundant bids. A bid is considered as redundant and removed if

1. **it bids on the same combination of items as others and offers a lower bidding price. Mathematically, let the bid tuples of b_x and b_y be (S_x, p_x) and (S_y, p_y) respectively. b_i is removed if $S_x=S_y$, but $p_x<p_y$.**
2. **it bids on the same combination of items as others and offers the same bidding price, but it is submitted at a later time. Mathematically, let t_x and t_y be the time of bids submitted by b_x and b_y respectively. b_x is removed if $S_x=S_y$ and $p_x=p_y$, but $t_x>t_y$.**
3. **its bidding set of items is a superset of another bid's, but it offers a lower bidding price. Mathematically, b_x is removed if $S_x \supseteq S_y$ and $p_x<p_y$.**
4. **its bidding price is less than a lower bound price which is determined by Algorithm 9.** (This criterion was not used when evaluating CADIA against other techniques. The inclusion of it is to improve CADIA's practicality which will be discussed in Chapter 8).

5. **its bidding set of items is a superset of that of the union of two or more mutually exclusive bids, but it offers a lower bidding than the total price of these bids. Mathematically, b_x is removed if $S_x \supseteq S_{y1} \cup S_{y2} \cup \ldots \cup S_{yn}$ and $S_i \cap S_j = \varnothing$ where $i, j \in \{y1, y2, \ldots, yn\}$ and $i \neq j$ and $p_x < (p_{y1} + p_{y2} + .. + p_{yn})$.**

The implementation of criteria 1, 2 and 3 is straightforward. Criterion 4 will filter out those bids whose bidding prices are lower than a reference value called the lower bound price. Such a lower bound price is determined based on a greedy search algorithm. Thus, the higher the lower bound price, the less the number of bids will be qualified for the next processing phase. Algorithm 9 describes how the lower bound price is calculated. A scale factor C_{sf}, whose value between 0 and 1 is selected by the auctioneer, can be used to scale down the lower bound price to allow more bids to be qualified for the next phase. C_{sf} has been set to 0.5 by default. Chapter 8 will discuss the impact of the lower bound price on the running time of the results.

```
Algorithm:  Calculate the lower bound price for each bid
Input:      all bids b_i ∈ B and bid tuples {S_i,p_i},
            i∈{0,1,…n},
            all items M = {1, 2,…, m},
            scale factor C_sf = {0..1},
Output:     the lower bound price b_i.lbPrice for all bids b_i ∈B
Begin
      //use Algorithm 2, a set of winning bids B_greedy is obtained
      B_greedy ← Greedy_Search_Winner_Determination(B)


      greedyRevenue ← 0
      for each bid b_i ∈ B_greedy
         greedyRevenue ← greedyRevenue + p_i
      lbPricePerItem ← greedyRevenue / |M|

      for each bid b_i ∈ B
         b_i.lbPrice ← lbPricePerItem × |S_i| × C_sf
End
```

Algorithm 9 Determine the lower bound price for each bid.

Criterion 5 can be met if an exhaustive search technique is adopted because it requires a search for all bidding sets of items to determine if it is a superset of the union of any other bidding sets. CADIA has delayed the implementation of the criterion until WIN. That is, CADIA checks if a bid is a redundant bid just before it is about to be declared as a candidate winner. The objective of such a delay is to perform the task only when it is needed in order to reduce the overall running time. Nevertheless, the larger the number of items and bids in an auction, the more time is required to perform the search. To further reduce the running time, WIN has been customized to perform a partial search instead. That is, it searches for all bidding sets of items to determine if it is a superset of

the union of any two or three sets only. The partial search is recommended especially for auctions of more than 500 items and 1000 bids. Algorithm 10 depicts the algorithm for determining if a bid's set of items is a superset of the union of any other three bids', but offers a lower bidding price than the sum of that of the three.

```
Algorithm: Test if bid b_x's bidding set of items S_x is a
superset of the union of any other 3 mutually exclusive bids,
but offers a lower bidding price than the total price of the 3
bids.
Input:      all bids b_i ∈ B and bid tuples {S_i,p_i}, i∈{0,1,…n}.
Output:     TRUE if S_x is a superset with lower price,
            FALSE otherwise.
Begin
      for each bid b_i∈B {
         for each bid b_j∈B {
            for each bid b_k∈B {
               if (i ≠ j ≠ k){
                  if (isSupersetOf3(x,i,j,k)=TRUE){
                     if p_x<(p_i+p_j+p_k)
                        return TRUE;
                     else
                        return FALSE;
                  }
               }
            }
         }
      }
      return FALSE;
End

Function isSupersetOf3 (x,i,j,k)
//test if S_x is a superset of the union of S_i ,S_j and S_k
Begin
      if (S_x ⊇ (S_i ∪ S_j ∪ S_k)
            return TRUE;
      else
            return FALSE;
End
```

Algorithm 10 Determine if a bid is a superset of others.

6.2.2 Tactical Bids Elimination Unit (TBE)

It has been discussed in the Section 4.1 that the biggest concern in CADIA's design is the completeness of search for the optimal solution based on the least frequent itemset. A bid which attempts to be granted the winner status may tactically bid on the least wanted itemset. In fact, such a worry is unnecessary because the least wanted itemset is not obvious in auctions of hundreds or even thousands of bids.

Nevertheless, the TBE of CADIA has eased the above concern because it provides further analysis on all potential winners and possible losers that are output from WIN. Suppose b_i is a potential winner, TBE will first assume b_i to be a tactical bid and test the revenue generated if b_i is removed for the auction. An improvement on the revenue may or may not conclude if b_i is a tactical bid, but it will definitely suggest that b_i should not be a winner and be removed from the auction. In CADIA's design, TBE will improve the revenue during the winner determination process using either one of the two different strategies.

In the first strategy, TBE tests the expected revenue when N potential winners and possible losers, which are identified from the previous round of winner determination, are removed from the auction. The integer N, which is specified as an argument when executing CADIA, is referred to as the number of analysis bids. Suppose 2 analysis bids are specified in auction, CADIA will iterate 3 times to search for better revenue. The first iteration does not remove any bids. The second iteration removes the first winner and last loser determined during the first iteration, and the third iteration removes the first two winners and last two losers determined during the second iteration.

TBE in the second strategy also uses both the potential winners and possible losers to improve the revenue but requires a slight complex implementation and thus a detailed description is given next. Suppose N analysis bids are specified in an auction, TBE tests the expected revenue when $^{N}/_{2}$ potential winners and $^{N}/_{2}$ possible losers are removed from the auction. CADIA will iterate 2^{N} times to search for better revenue. The algorithm for testing if a combination of potential winners and losers should be removed is described in Algorithm 11.

The lists of potential winners and possible losers can be seen as additional knowledge discovered during the process. The incorporation of TBE and such knowledge has made CADIA able to search for better or even the optimal revenue. TBE with the second strategy applied a more extensive search than that with the first one, and thus able to obtain better results. Thus, TBE with the second strategy is used when evaluating CADIA with an optimal revenue search technique such as the BFT. The drawback of the second strategy is that it requires more time to complete the search process. In an auction of 20 items, 1000 bids and 6 analysis bids, the second strategy requires 130 seconds but the first strategy requires only 27 seconds for the whole process. For faster result in auction size of hundred items and thousands of bids, TBE with the first strategy may be used. Thus, TBE with the first strategy is used when evaluating CADIA with other approximation techniques.

```
Algorithm:  Identify the combination of potential winners
            b_i∈L_winners and possible losers b_j∈L_losers to be removed
            to further improve the revenue
Input:      all bids b_i ∈ B, and bid tuples {S_i,p_i}, i∈{0,1,…n}.
            lists of potential L_winners and possible L_losers ,
            Number of analysis bids C_a.
Output:     the combination of potential winners and possible
            losers to be removed from the auction
            CmbOfBidToBeRemoved.
Begin
      //make up a list of analysis bids from winners and losers
      Loop until c >= C_a {            //c=0,1,2…
         L_analysis ← L_analysis + L_winners [c]
         L_analysis ← L_analysis + L_losers [c]
         c ← c + 2
      }

      highestRevenue ← CADIA (B)
      for each combination of bids B_i ⊆ L_analysis {  //i=0,1,..,C_a-1
         revenue = CADIA (B-B_i)
         if (highestRevenue < revenue){
            highestRevenue ← revenue
            CmbOfBidsToBeRemoved = B_j
         }
      }
End

Function CADIA (B)
//return the revenue generated for an auction containing the
//set of bids B
Begin
      //use Algorithm 8 - find a winner
      For each winner b_w ∈ B
            revenue ← revenue + p_w
      return revenue
End
```

Algorithm 11 Remove potential winners and losers for result improvement.

6.3 Example

The improved CADIA is demonstrated with the same data (Figure 27) used in the Section 4.3. At the beginning, CADIA will read the bid data as inputs and organize them into a matrix in the memory as described in Figure 28.

{bid}	{a set of items}	{bidding price}
$\{b_0\}$	{0 4 6 7}	{206.28}
$\{b_1\}$	{0 1 3 4}	{207.28}
$\{b_2\}$	{0 6}	{205.00}
$\{b_3\}$	{0 4 5 9}	{208.28}
$\{b_4\}$	{2 4 5 8}	{108.28}
$\{b_5\}$	{1 2 7}	{55.74}
$\{b_6\}$	{1 2 3 6}	{55.74}
$\{b_7\}$	{2 9}	{152.00}
$\{b_8\}$	{0 4 8}	{154.74}
$\{b_9\}$	{0 4 6 7 8}	{205.50}

Figure 27 Bid data.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	
b4			x		x	x			x		108.28	
b5		x	x					x			55.74	
b6		x	x	x			x				55.74	
b7			x							x	152.00	
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	

Figure 28 Auction data is represented internally as a matrix in CADIA.

During the first step at PRE, any redundant bids will be removed. Since the bid tuples of b_9 and b_0 are $(S_9,p_9)=\{\{0,4,6,7,8\}, 205.50\}$ and $(S_0,p_0)=\{\{0,4,6,7\}, 206.28\}$respectively, b_9 will be removed because it is a superset of b_0 but it offers a lower

bidding price (i.e., $S_9 \supseteq S_0$ but $p_9 < p_0$) according to the criterion 4 stated in Section 6.2.1. The qualified bids after the filtering process at PRE are b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, and b_8 (Figure 29).

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	
b4			x		x	x			x		108.28	
b5		x	x					x			55.74	
b6		x	x	x			x				55.74	
b7			x							x	152.00	
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	loser

Figure 29 Bid b_9 becomes a loser after PRE.

During the second step, IAG of CADIA sets the minimum support count to one (a non-zero least support count), starts generating frequent 1-itemsets (Figure 30) and checks if there are itemsets whose frequency counts are equal to but not greater than the minimum support count. That is, IAG identifies the smallest but also least frequent itemset from all bids as described in Section 4.2.2. In this example, all ten frequent 1-itemsets have frequency counts greater than the minimum support count. Thus, IAG is required to generate frequent 2-itemsets (Figure 31). Now, twenty-four out of forty-five itemsets have frequency counts equal to the minimum support count. Additional measures such as the degree of confidence and the degree of conflict will then be applied according to Algorithms 5 and 6 to reduce the total number of itemsets. Figure 32 and Figure 33 show that the highest degree of confidence and the lowest degree of conflict

are found to be 50% and 5 respectively. As a result, the itemset {2,9} is determined as the least frequent itemset, which will be used by WIN to determine candidate winners.

itemset	frequency
{0}	5
{1}	3
{2}	4
{3}	2
{4}	5
{5}	2
{6}	3
{7}	2
{8}	2
{9}	2

Figure 30 Frequent 1-itemsets during the first iteration.

itemset	frequency	itemset	frequency	itemset	frequency
{0,1}	1	{1,8}	0	{4,5}	2
{0,2}	0	{1,9}	0	{4,6}	1
{0,3}	1	{2,3}	1	{4,7}	1
{0,4}	4	{2,4}	1	{4,8}	2
{0,5}	1	{2,5}	1	{4,9}	1
{0,6}	2	{2,6}	1	{5,6}	0
{0,7}	1	{2,7}	1	{5,7}	0
{0,8}	1	{2,8}	1	{5,8}	1
{0,9}	1	{2,9}	1	{5,9}	1
{1,2}	2	{3,4}	1	{6,7}	1
{1,3}	2	{3,5}	0	{6,8}	0
{1,4}	1	{3,6}	1	{6,9}	0
{1,5}	0	{3,7}	0	{7,8}	0
{1,6}	1	{3,8}	0	{7,9}	0
{1,7}	1	{3,9}	0	{8,9}	0

Figure 31 Frequent 2-itemsets during the first iteration

itemset	confidence(%)	itemset	confidence(%)	itemset	confidence(%)
{0,1}	33.33	{1,7}	50.00	{3,4}	50.00
{0,3}	50.00	{2,3}	50.00	{3,6}	50.00
{0,5}	50.00	{2,4}	25.00	{4,6}	33.33
{0,7}	50.00	{2,5}	50.00	{4,7}	50.00
{0,8}	50.00	{2,6}	33.33	{4,9}	50.00
{0,9}	50.00	{2,7}	50.00	{5,8}	50.00
{1,4}	33.33	{2,8}	50.00	{5,9}	50.00
{1,6}	33.33	{2,9}	50.00	{6,7}	50.00

Figure 32 Degrees of confidence for itemsets with the least support count

itemset	confidence(%)	conflict	itemset	confidence(%)	conflict
{0,3}	50.00	8	{2,8}	50.00	8
{0,5}	50.00	7	{2,9}	50.00	5
{0,7}	50.00	8	{3,4}	50.00	8
{0,8}	50.00	6	{3,6}	50.00	7
{0,9}	50.00	7	{4,7}	50.00	8
{1,7}	50.00	6	{4,9}	50.00	7
{2,3}	50.00	7	{5,8}	50.00	8
{2,5}	50.00	8	{5,9}	50.00	7
{2,7}	50.00	6	{6,7}	50.00	8

Figure 33 Degrees of conflict for itemsets with maximum confidence.

In the next step, WIN of CADIA will identify all candidate winners. WIN starts looking for those bids that include the least frequent itemset {2,9}. In this example, only b_7 contains itemset {2,9}. Consequently, b_7 is determined as a potential winner (Figure 34). After the potential winner is declared, all bids that conflict with it are determined as possible losers (Figure 35). Since winner determination is a multi-round process involving both WIN and IAG, we can only say that b_7 is one potential winner and b_3, b_4, b_5 and b_6 are some possible losers during the first iteration.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	
b4			x		x	x			x		108.28	
b5		x	x					x			55.74	
b6		x	x	x			x				55.74	
b7			x							x	152.00	winner
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	loser

Figure 34 Bid b_7 becomes a winner after the first iteration.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	
b2	x						x				205.00	
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	loser

Figure 35 Bid b_3, b_4, b_5, b_6 become losers after the first iteration.

In the second iteration, the qualified bids are b_0, b_1, b_2 and b_8. IAG will update the item association pattern based on the current available qualified bids. IAG again sets the minimum support count to one, generates frequent 1-itemsets (Figure 36), and checks if there are itemsets whose frequency counts are equal to the minimum support count. In addition, the degree of confidence and the degree of conflict are checked for each itemset. It is shown in Figure 37 that the highest degree of confidence and the lowest degree of conflict are found to be 100% and 4 respectively. Since itemsets {1}, {3}, {7} and {8} all have satisfied the condition of being the least frequent itemsets, they are used for candidate winners determination.

itemset	frequency
{0}	4
{1}	1
{2}	0
{3}	1
{4}	3
{5}	0
{6}	2
{7}	1
{8}	1
{9}	0

Figure 36 Frequent 1-itemsets during the second iteration.

itemset	confidence(%)	conflict
{1}	100.00	4
{3}	100.00	4
{7}	100.00	4
{8}	100.00	4

Figure 37 Degrees of conflict for itemsets with maximum confidence during the second iteration.

In candidate winners determination, WIN looks for those bids that include the least frequent itemset {1}, {3}, {7} or {8}. Bids b_0, b_1, and b_8 become candidate winners because b_0 contains itemset {7}, b_1 contains itemset {1} and {3}, and b_8 contains itemset {8}. According to the criterion 1 in Section 4.2.2, b_0, b_1, and b_8 are conflicted bids and the bidding price comparison strategy must be applied to select only one potential winner. As a result, b_1 is determined to be the potential winner during the second iteration (Figure 39). After the potential winner has been declared, all bids that conflict with it are determined as possible losers (Figure 40). After the second iteration, WIN stops because all bids have been processed. As a result, b_1 and b_7 are the potential winners which generate the total revenue of $359.28.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	candidate
b1	x	x		x	x						207.28	candidate
b2	x						x				205.00	candidate
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	candidate
b9	x				x		x	x	x		205.50	loser

Figure 38 Bids b_0, b_1, b_2, b_8 become candidate winners.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	
b1	x	x		x	x						207.28	winner
b2	x						x				205.00	
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	
b9	x				x		x	x	x		205.50	loser

Figure 39 Bid b_1 becomes the potential winner.

Item / Bid	0	1	2	3	4	5	6	7	8	9	Bid Price	winner / loser
b0	x				x		x	x			206.28	loser
b1	x	x		x	x						207.28	winner
b2	x						x				205.00	loser
b3	x				x	x				x	208.28	loser
b4			x		x	x			x		108.28	loser
b5		x	x					x			55.74	loser
b6		x	x	x			x				55.74	loser
b7			x							x	152.00	winner
b8	x				x				x		154.74	loser
b9	x				x		x	x	x		205.50	loser

Figure 40 Bids b_0, b_2, b_8 become possible losers.

During the fourth step at TBE of CADIA, all potential winners and possible losers will be analysed to further improve the revenue. The second strategy of TBE described in Section 6.2.2 is used in this illustration. In WIN's implementation, all potential winners and possible losers are recorded in two separate lists. The time during which a winner or a loser is identified will determine its order in the lists. According to Algorithm 11, the

order of elements in $L_{winners}$ (list of winners) and L_{losers} (list of losers) become $\{b_7, b_1\}$ and $\{b_3, b_4, b_5, b_6, b_0, b_2, b_8\}$ respectively. TBE will then form the analysis bid list $L_{analysis}$ based on $L_{winners}$ and L_{losers}. Suppose the number of analysis bids specified by the user of CADIA is 3, the content of $L_{analysis}$, which is made up by picking the first winner, followed by the last loser, and then the second winner, becomes $\{b_7, b_8, b_1\}$. Since $|B_{analysis}| = 3$, CADIA will run in the first round without removing any bids, and then additional seven times to search for better revenue because there are seven different ways of combining these analysis bids. TBE will remove each of these combinations and record the revenue generated in each run. The results, which have been summarized in table 3.

Round	The set of Bids	Winners (time order)	Running time (second)	Revenue ($)
1	B	b_7, b_1	0.11	359.28
2	$B - \{b_7, b_8, b_1\}$	b_3, b_5	0.12	264.02
3	$B - \{b_7, b_8\}$	b_3, b_5	0.12	264.02
4	$B - \{b_7, b_1\}$	b_3, b_5	0.11	264.02
5	$B - \{b_7\}$	b_3, b_5	0.12	264.02
6	$B - \{b_8, b_1\}$	b_2, b_4	0.11	313.28
7	$B - \{b_8\}$	b_2, b_4	0.11	313.28
8	$B - \{b_1\}$	b_3, b_6	0.12	264.02

Table 3 CADIA runs eight times for bid and revenue analysis.

In this example, the best result remains the one initially determined by WIN. That is, the winners and revenue remain b_1, b_7, and $359.28 respectively. In this example, $359.28 is in fact the optimal revenue.

CHAPTER SEVEN:
EVALUATION (II)

7.1 Purpose

The purpose of the evaluation is to evaluate the new implementation of CADIA's accuracy and efficiency. The same evaluation plan as described in Chapter 5 is adopted.

7.2 Experimental Setup

Three different tests were performed in the evaluation. The objective of the first test was to justify the conclusion that CADIA is capable of finding the optimal revenue for the generated sample auctions. Thus, CADIA was compared with an optimal revenue search technique in terms of revenue generation and running time. The objective of the second test was to justify the conclusion that CADIA is a good approximation system Thus, CADIA was compared with some approximation techniques. The objective of the third test was to justify the conclusion that CADIA is still an efficient system even though its running time grows exponentially. Thus, CADIA's running time was measured against different numbers of auctioned items and bids.

The software implementation of the extended version of CADIA takes five parameters during its execution. They are:

1. **a file containing the list of auctioned items,**
2. **a file containing the bid data,**
3. **an integer corresponding to the number of items in the auction,**
4. **an integer corresponding to the number of bids in the auction, and**
5. **an integer corresponding to the number of analysis bids, which are composed from the lists of winners and losers, for further revenue improvement.**

CADIA can be executed from the command line as shown in Figure 41. The command says that CADIA will determine the winners in a CA of 500 items and 1000 bids; in addition the number of analysis bids is 10.

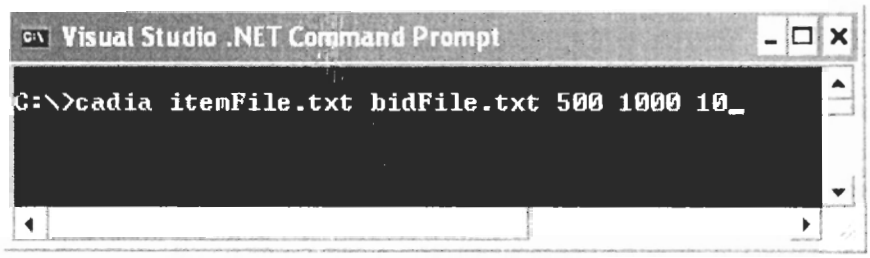

Figure 41 Execute CADIA with 5 arguments from command line.

7.2.1 Comparison of CADIA and BFT

The objective of the first test is to justify the conclusion that CADIA is capable of generate the optimal revenue. Thus, CADIA was compared with BFT in terms of revenue generation and running time on two hundred sample auctions with ten items and ten bids

each. According to the results described in Section 5.4.1, CADIA may not be able to obtain the optimal revenue after its first iteration of the winner determination process. With the adjustments made at the TBE (Section 6.2.2) based on the information of analysis bids, CADIA is able to obtain better revenue at successive iterations. In this evaluation, TBE with the second strategy as described in Second 6.2.2 is adopted. Thus, the revenue generated when the number of analysis bids equals two, four, and six are recorded accordingly. Additionally, the total time required by CADIA to reach the optimal revenue is recorded in each auction. The results are documented and analyzed in Section 7.3.1.

7.2.2 Comparison of CADIA and GST, Four Hill Climbers and ESG

The objective of the second test is to justify the conclusion that CADIA is a good approximation system that always guarantees a better than or equal to the lower bound revenue. Since GST is based on greedy search technique, it can handle more items and bids that cannot be handled by BFT and always returns feasible results in reasonable time. Thus, CADIA was compared with the GST. Besides, CADIA was compared with some approximation systems including the four hill climbers (PRICE, N2NORM, KO, DEMAND) [Holte, 2001] and the Exponential Subgradient (ESG) [Schuurmans et al., 2001]in terms of revenue generation in two hundred sample auctions with twenty items and one thousand bids each.

Each of the hill climbers uses a different objective function. PRICE's function, which is based on the bid prices, selects the search path which results in the greatest

increase in the value of the included bids. N2NORM's function is based on the 2-norm. It divides the bid's price by its "size", where the size of bid j is the square root of the sum of squares of the $f_{i,j}$, the fraction of the remaining quantity of item i that bid j requires. KO's function is based on the division of the bid's price by its "knockout cost", where a bid's knockout cost is the sum of the prices of the available bids that are eliminated if this bid is chosen. DEMAND's function is based on a given "price" that is derived from the sum of the values of all bids referencing that item. A bid is weighted based on how much it is willing to pay versus the amount of money willing to be paid by other bids for the requested items. ESG, which is based on the gradient search method, attempts to find a directional derivative so that the search can proceed in the direction of the steepest ascent in the search space. In ESG, constraints are used to penalize movements that do not approach the optimum or, to reward those that approach the optimum. The idea is to find the right step size to guarantee the best rate of improvement over several iterations.

Since CADIA is able to generate higher revenue at successive iterations, the results with different number of analysis bids are recorded. All results are documented analyzed in Section 7.3.2.

7.2.3 Running Time Measurement of CADIA

The objective of the third test is to justify the conclusion that CADIA is still an efficient system even though its running time grows exponentially. The running time comparisons of CADIA with the approximation techniques have been ignored in the evaluation because CADIA in general runs slower than other approximation techniques.

Such a slower response time is due to the obvious fact that CADIA's core knowledge requires some time to generate.

CADIA's running time is measured against different number of auctioned items and bids. The sample sizes of the auctioned items and bids are selected in the range of 100 to 500 and 200 to 2000 respectively. The results are then plotted as two separate graphs. The first measures the running time when the number of bids is fixed and the number of items varies. The second measures the running time when the number of items is fixed and the number of bids varies. The test plan is outlined in Table 4. The results are documented and analyzed in Section 7.3.3.

Number of Bids	Number of Items
200	100, 200, 300, 400, 500
400	100, 200, 300, 400, 500
600	100, 200, 300, 400, 500
800	100, 200, 300, 400, 500
1000	100, 200, 300, 400, 500
1200	100, 200, 300, 400, 500
1400	100, 200, 300, 400, 500
1600	100, 200, 300, 400, 500
1800	100, 200, 300, 400, 500
2000	100, 200, 300, 400, 500

Table 4 Test plan for measuring CADIA's running time

7.3 Empirical Results and Analysis

In this section, the results of the tests described in Section 7.2.1, 7.2.2, and 7.2.3 are documented in Section 7.3.1, 7.3.2, and 7.3.3 respectively. The results will be used to justify the hypothesis in designing CADIA.

7.3.1 Comparison of CADIA and BFT

Even though BFT works in principle, it is practically limited by the number of items and bids it can process. CADIA may find the optimal revenue with or without the adjustments made by TBE. Thus, it is interesting to know how accurate CADIA is before and after TBE adjustments.

Tables 5, 6, 7, and 8 report the results for all two hundred sample auctions. The abbreviations A0, A2, A4, and A6 after the word CADIA mean that zero, two, four, and six analysis bids are used by CADIA. The accuracy ratio of BFT in all sample auctions is always one because BFT always generates the optimal revenue and thus is used as the standard for comparison. CADIA has an average accuracy ratio of 0.979 (Table 9) and is not able to find the optimal revenue in 36 out of 200 auctions when no analysis is used. The results of these 200 auctions are sorted by accuracy ratio for CADIA (A0) and plotted as a line chart (Figure 42 and Figure 43). The Figure 43, which is an enlarged view of the 36 auctions, shows how the result is improved when more and more analysis bids are included. It is found in the 200 sample auctions that CADIA attains the accuracy ratio of 1.0 and returns the optimal revenue for all sample auctions when six analysis bids are used.

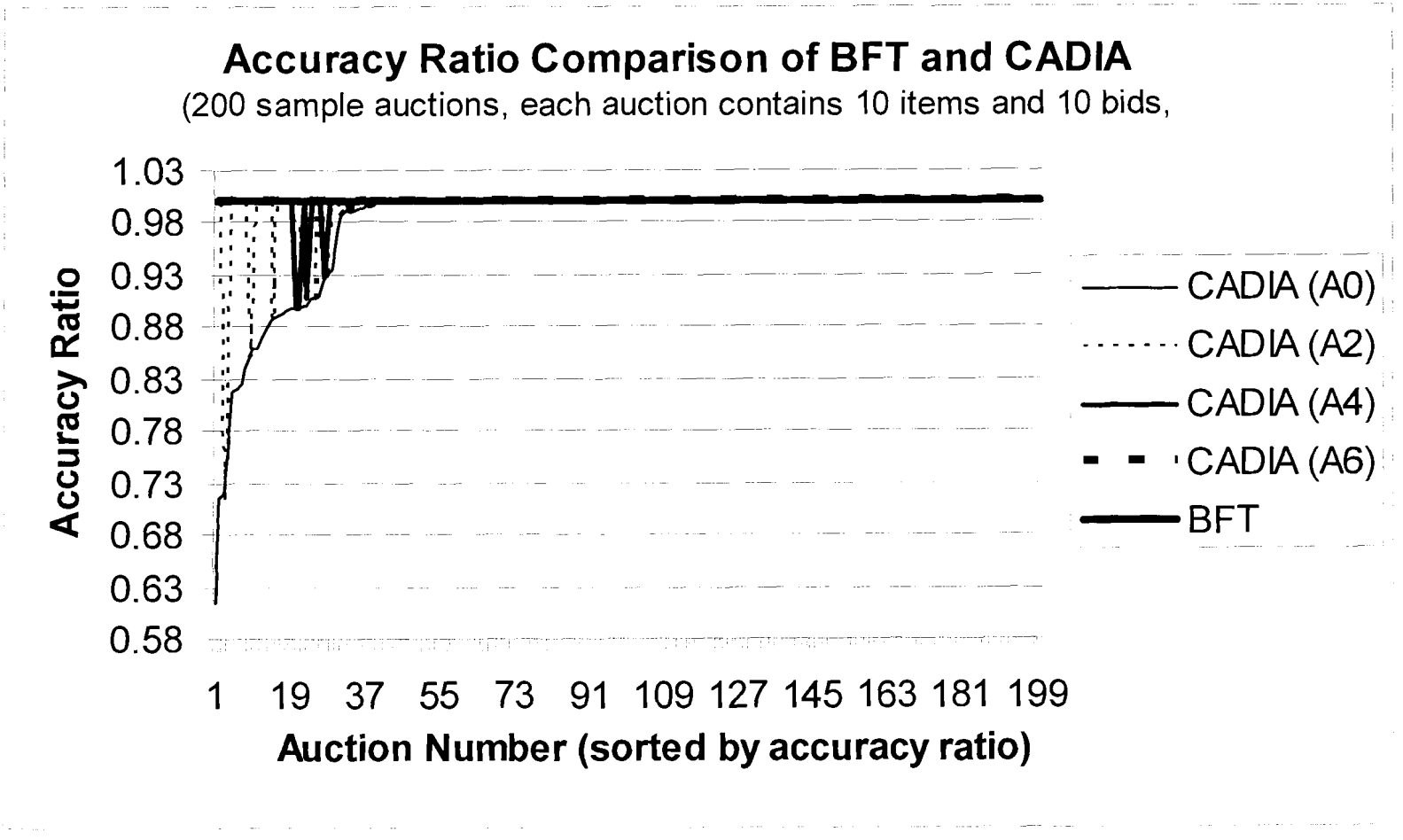

Figure 42 Accuracy ratio comparison of BFT and CADIA (all 200 auctions).

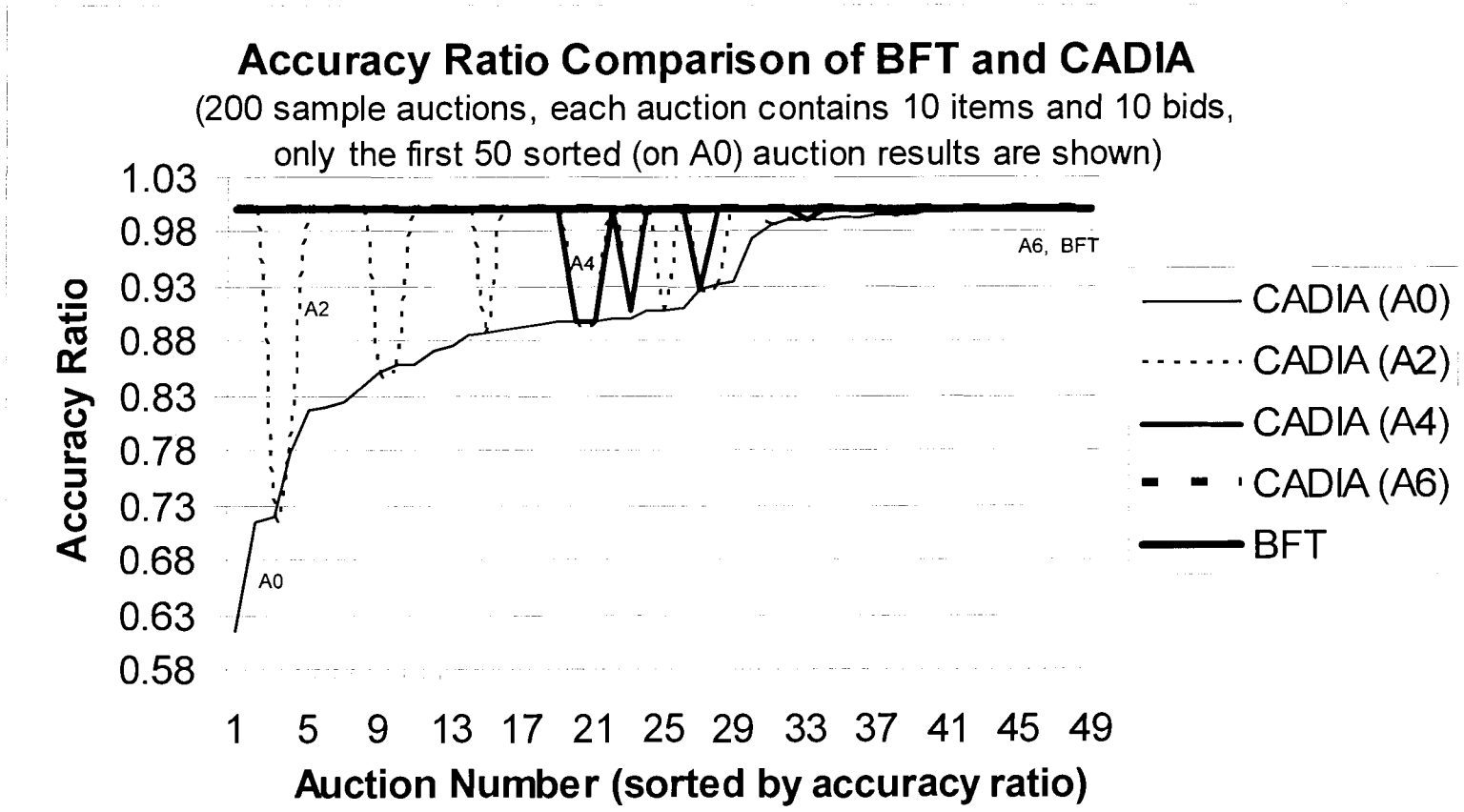

Figure 43 Accuracy ratio comparison of BFT and CADIA (enlarged view of Figure 42).

AUCTION NUMBER	BFT	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
1	1.000	1.000	1.000	1.000	1.000
2	1.000	1.000	1.000	1.000	1.000
3	1.000	0.719	0.719	1.000	1.000
4	1.000	1.000	1.000	1.000	1.000
5	1.000	1.000	1.000	1.000	1.000
6	1.000	1.000	1.000	1.000	1.000
7	1.000	1.000	1.000	1.000	1.000
8	1.000	0.715	1.000	1.000	1.000
9	1.000	1.000	1.000	1.000	1.000
10	1.000	0.877	1.000	1.000	1.000
11	1.000	1.000	1.000	1.000	1.000
12	1.000	1.000	1.000	1.000	1.000
13	1.000	0.819	1.000	1.000	1.000
14	1.000	1.000	1.000	1.000	1.000
15	1.000	1.000	1.000	1.000	1.000
16	1.000	1.000	1.000	1.000	1.000
17	1.000	1.000	1.000	1.000	1.000
18	1.000	1.000	1.000	1.000	1.000
19	1.000	1.000	1.000	1.000	1.000
20	1.000	0.817	1.000	1.000	1.000
21	1.000	1.000	1.000	1.000	1.000
22	1.000	1.000	1.000	1.000	1.000
23	1.000	1.000	1.000	1.000	1.000
24	1.000	1.000	1.000	1.000	1.000
25	1.000	1.000	1.000	1.000	1.000
26	1.000	1.000	1.000	1.000	1.000
27	1.000	0.890	1.000	1.000	1.000
28	1.000	1.000	1.000	1.000	1.000
29	1.000	1.000	1.000	1.000	1.000
30	1.000	1.000	1.000	1.000	1.000
31	1.000	0.870	1.000	1.000	1.000
32	1.000	1.000	1.000	1.000	1.000
33	1.000	1.000	1.000	1.000	1.000
34	1.000	1.000	1.000	1.000	1.000
35	1.000	0.824	1.000	1.000	1.000
36	1.000	1.000	1.000	1.000	1.000
37	1.000	1.000	1.000	1.000	1.000
38	1.000	1.000	1.000	1.000	1.000
39	1.000	1.000	1.000	1.000	1.000
40	1.000	1.000	1.000	1.000	1.000
41	1.000	1.000	1.000	1.000	1.000
42	1.000	1.000	1.000	1.000	1.000
43	1.000	0.974	1.000	1.000	1.000
44	1.000	1.000	1.000	1.000	1.000
45	1.000	1.000	1.000	1.000	1.000
46	1.000	0.851	0.851	1.000	1.000
47	1.000	1.000	1.000	1.000	1.000
48	1.000	1.000	1.000	1.000	1.000
49	1.000	1.000	1.000	1.000	1.000
50	1.000	1.000	1.000	1.000	1.000

Table 5 Accuracy ratio comparison of BFT and CADIA (sample 1-50).

AUCTION NUMBER	BFT	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
51	1.000	1.000	1.000	1.000	1.000
52	1.000	1.000	1.000	1.000	1.000
53	1.000	1.000	1.000	1.000	1.000
54	1.000	1.000	1.000	1.000	1.000
55	1.000	0.996	0.996	0.996	1.000
56	1.000	1.000	1.000	1.000	1.000
57	1.000	1.000	1.000	1.000	1.000
58	1.000	1.000	1.000	1.000	1.000
59	1.000	1.000	1.000	1.000	1.000
60	1.000	1.000	1.000	1.000	1.000
61	1.000	1.000	1.000	1.000	1.000
62	1.000	1.000	1.000	1.000	1.000
63	1.000	1.000	1.000	1.000	1.000
64	1.000	1.000	1.000	1.000	1.000
65	1.000	1.000	1.000	1.000	1.000
66	1.000	1.000	1.000	1.000	1.000
67	1.000	1.000	1.000	1.000	1.000
68	1.000	1.000	1.000	1.000	1.000
69	1.000	1.000	1.000	1.000	1.000
70	1.000	1.000	1.000	1.000	1.000
71	1.000	1.000	1.000	1.000	1.000
72	1.000	1.000	1.000	1.000	1.000
73	1.000	0.777	0.777	1.000	1.000
74	1.000	0.990	0.990	1.000	1.000
75	1.000	0.910	1.000	1.000	1.000
76	1.000	1.000	1.000	1.000	1.000
77	1.000	1.000	1.000	1.000	1.000
78	1.000	0.900	1.000	1.002	1.000
79	1.000	1.000	1.000	1.000	1.000
80	1.000	1.000	1.000	1.000	1.000
81	1.000	1.000	1.000	1.000	1.000
82	1.000	1.000	1.000	1.000	1.000
83	1.000	1.000	1.000	1.000	1.000
84	1.000	1.000	1.000	1.000	1.000
85	1.000	1.000	1.000	1.000	1.000
86	1.000	0.908	0.908	1.000	1.000
87	1.000	1.000	1.000	1.000	1.000
88	1.000	1.000	1.000	1.000	1.000
89	1.000	0.995	0.995	1.000	1.000
90	1.000	1.000	1.000	1.000	1.000
91	1.000	1.000	1.000	1.000	1.000
92	1.000	1.000	1.000	1.000	1.000
93	1.000	1.000	1.000	1.000	1.000
94	1.000	1.000	1.000	1.000	1.000
95	1.000	1.000	1.000	1.000	1.000
96	1.000	1.000	1.000	1.000	1.000
97	1.000	1.000	1.000	1.000	1.000
98	1.000	1.000	1.000	1.000	1.000
99	1.000	1.000	1.000	1.000	1.000
100	1.000	1.000	1.000	1.000	1.000

Table 6 Accuracy ratio comparison of BFT and CADIA (sample 51-100).

AUCTION NUMBER	BFT	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
101	1.000	1.000	1.000	1.000	1.000
102	1.000	1.000	1.000	1.000	1.000
103	1.000	1.000	1.000	1.000	1.000
104	1.000	1.000	1.000	1.000	1.000
105	1.000	0.928	0.928	0.928	1.000
106	1.000	1.000	1.000	1.000	1.000
107	1.000	1.000	1.000	1.000	1.000
108	1.000	1.000	1.000	1.000	1.000
109	1.000	1.000	1.000	1.000	1.000
110	1.000	0.986	0.986	1.000	1.000
111	1.000	1.000	1.000	1.000	1.000
112	1.000	0.908	1.000	1.000	1.000
113	1.000	0.934	1.000	1.000	1.000
114	1.000	1.000	1.000	1.000	1.000
115	1.000	0.933	0.933	1.000	1.000
116	1.000	1.000	1.000	1.000	1.000
117	1.000	1.000	1.000	1.000	1.000
118	1.000	0.901	0.907	0.907	1.000
119	1.000	1.000	1.000	1.000	1.000
120	1.000	1.000	1.000	1.000	1.000
121	1.000	1.000	1.000	1.000	1.000
122	1.000	1.000	1.000	1.000	1.000
123	1.000	0.991	0.991	0.991	1.000
124	1.000	1.000	1.000	1.000	1.000
125	1.000	1.000	1.000	1.000	1.000
126	1.000	0.897	1.000	1.000	1.000
127	1.000	0.889	0.889	1.000	1.000
128	1.000	1.000	1.000	1.000	1.000
129	1.000	0.885	1.000	1.000	1.000
130	1.000	1.000	1.000	1.000	1.000
131	1.000	1.000	1.000	1.000	1.000
132	1.000	1.000	1.000	1.000	1.000
133	1.000	0.892	1.000	1.000	1.000
134	1.000	1.000	1.000	1.000	1.000
135	1.000	1.000	1.000	1.000	1.000
136	1.000	1.000	1.000	1.000	1.000
137	1.000	1.000	1.000	1.000	1.000
138	1.000	1.000	1.000	1.000	1.000
139	1.000	1.000	1.000	1.000	1.000
140	1.000	0.859	0.859	1.000	1.000
141	1.000	1.000	1.000	1.000	1.000
142	1.000	0.896	1.000	1.000	1.000
143	1.000	1.000	1.000	1.000	1.000
144	1.000	1.000	1.000	1.000	1.000
145	1.000	1.000	1.000	1.000	1.000
146	1.000	1.000	1.000	1.000	1.000
147	1.000	1.000	1.000	1.000	1.000
148	1.000	1.000	1.000	1.000	1.000
149	1.000	1.000	1.000	1.000	1.000
150	1.000	1.000	1.000	1.000	1.000

Table 7 Accuracy ratio comparison of BFT and CADIA (sample 101-150).

AUCTION NUMBER	BFT	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
151	1.000	0.840	1.000	1.000	1.000
152	1.000	1.000	1.000	1.000	1.000
153	1.000	0.859	1.000	1.000	1.000
154	1.000	1.000	1.000	1.000	1.000
155	1.000	1.000	1.000	1.000	1.000
156	1.000	1.000	1.000	1.000	1.000
157	1.000	1.000	1.000	1.000	1.000
158	1.000	1.000	1.000	1.000	1.000
159	1.000	0.899	0.899	0.899	1.000
160	1.000	1.000	1.000	1.000	1.000
161	1.000	1.000	1.000	1.000	1.000
162	1.000	1.000	1.000	1.000	1.000
163	1.000	1.000	1.000	1.000	1.000
164	1.000	0.614	1.000	1.000	1.000
165	1.000	1.000	1.000	1.000	1.000
166	1.000	1.000	1.000	1.000	1.000
167	1.000	1.000	1.000	1.000	1.000
168	1.000	1.000	1.000	1.000	1.000
169	1.000	1.000	1.000	1.000	1.000
170	1.000	1.000	1.000	1.000	1.000
171	1.000	1.000	1.000	1.000	1.000
172	1.000	1.000	1.000	1.000	1.000
173	1.000	1.000	1.000	1.000	1.000
174	1.000	1.000	1.000	1.000	1.000
175	1.000	1.000	1.000	1.000	1.000
176	1.000	1.000	1.000	1.000	1.000
177	1.000	0.898	0.898	0.898	1.000
178	1.000	1.000	1.000	1.000	1.000
179	1.000	0.994	1.000	1.000	1.000
180	1.000	1.000	1.000	1.000	1.000
181	1.000	1.000	1.000	1.000	1.000
182	1.000	1.000	1.000	1.000	1.000
183	1.000	1.000	1.000	1.000	1.000
184	1.000	0.992	1.000	1.000	1.000
185	1.000	1.000	1.000	1.000	1.000
186	1.000	1.000	1.000	1.000	1.000
187	1.000	1.000	1.000	1.000	1.000
188	1.000	1.000	1.000	1.000	1.000
189	1.000	1.000	1.000	1.000	1.000
190	1.000	0.992	1.000	1.000	1.000
191	1.000	1.000	1.000	1.000	1.000
192	1.000	1.000	1.000	1.000	1.000
193	1.000	0.996	0.996	1.000	1.000
194	1.000	1.000	1.000	1.000	1.000
195	1.000	1.000	1.000	1.000	1.000
196	1.000	1.000	1.000	1.000	1.000
197	1.000	1.000	1.000	1.000	1.000
198	1.000	1.000	1.000	1.000	1.000
199	1.000	1.000	1.000	1.000	1.000
200	1.000	1.000	1.000	1.000	1.000

Table 8 Accuracy comparison of BFT and CADIA (sample 151-200).

	BFT	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
Average	1.000	0.979	0.993	0.998	1.000

Table 9 Accuracy ratio comparison of BFT and CADIA.

Table 10 summarizes the results of the running time comparison of BFT and CADIA with various numbers of analysis bids. Results show that BFT has an average running time of 211.722 seconds. CADIA's average running time is in the range 0.199 – 10.138 seconds when up to six analysis bids are considered. As expected, the larger the number of analysis bids involved, the longer running time CADIA takes.

	BFT	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
Average	211.722	0.199	0.754	2.749	10.138

Table 10 Running time comparison of BFT and CADIA.

Since each sample auction contains a total number of ten bids, the maximum number of analysis bids becomes ten. When the maximum number of analysis bids is used, CADIA is guaranteed to find the optimal revenue. The more the analysis bids are considered, the better the revenue will be generated. In most cases, CADIA may not use the maximum number of analysis bids before the optimal revenue is found. CADIA was able to return the optimal revenue in all our test cases when six analysis bids were considered. When six analysis bids are considered, CADIA will take only 10.138 seconds, which is about 4.8% of the time required by BFT, to identify the optimal

revenue. From the empirical results and analysis, CADIA can be concluded as a system that is capable of generating the optimal revenue and that runs much faster than BFT based systems.

7.3.2 Comparison of CADIA and GST, Four Hill Climbers and ESG

7.3.2.1 Comparison of CADIA and GST

When the there are too many items and bids, it becomes impractical to compare BFT and CADIA. CADIA is compared with GST in terms of a revenue generation. In this test, each sample auction contains twenty items and one thousand bids. The number of items has been selected in such a way that it cannot be handled realistically by BFT, but it is still small enough as compared to the number of bids. In this evaluation, the ratio of the number of items to the number of bids is 50. The purpose of such a setup is to simulate realistic CAs in which there are always conflicts among bids. Tables 11 to 14 summarize the performance ratios of GST and CADIA with various numbers of analysis bids for two hundred sample auctions. The first strategy of TBE as described in Section 6.2.2 is adopted here to organize the analysis bids. The results are sorted by performance ratio for CADIA (A0) and plotted as a line chart (Figure 44).

AUCTION NUMBER	GST	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
1	1.000	1.238	1.238	1.238	1.238
2	1.000	1.053	1.054	1.054	1.054
3	1.000	1.000	1.000	1.000	1.000
4	1.000	1.126	1.126	1.126	1.126
5	1.000	1.000	1.000	1.000	1.000
6	1.000	1.167	1.167	1.167	1.167
7	1.000	1.116	1.120	1.120	1.120
8	1.000	1.043	1.043	1.043	1.043
9	1.000	1.000	1.000	1.000	1.000
10	1.000	1.041	1.044	1.044	1.044
11	1.000	1.095	1.095	1.095	1.095
12	1.000	1.323	1.323	1.323	1.323
13	1.000	1.130	1.130	1.130	1.130
14	1.000	1.044	1.044	1.044	1.044
15	1.000	1.168	1.168	1.168	1.168
16	1.000	1.113	1.113	1.113	1.113
17	1.000	1.179	1.179	1.179	1.179
18	1.000	1.113	1.113	1.113	1.113
19	1.000	1.200	1.200	1.200	1.200
20	1.000	1.048	1.050	1.050	1.050
21	1.000	1.106	1.106	1.108	1.108
22	1.000	1.128	1.128	1.128	1.128
23	1.000	1.120	1.120	1.120	1.120
24	1.000	1.301	1.301	1.301	1.301
25	1.000	1.157	1.157	1.157	1.157
26	1.000	1.000	1.000	1.000	1.000
27	1.000	1.276	1.276	1.276	1.276
28	1.000	1.053	1.053	1.053	1.053
29	1.000	1.121	1.121	1.121	1.121
30	1.000	1.048	1.050	1.050	1.050
31	1.000	1.056	1.056	1.056	1.056
32	1.000	1.178	1.178	1.178	1.178
33	1.000	1.255	1.255	1.255	1.257
34	1.000	1.129	1.129	1.129	1.129
35	1.000	1.040	1.040	1.040	1.040
36	1.000	1.099	1.099	1.099	1.099
37	1.000	1.321	1.321	1.323	1.323
38	1.000	1.189	1.192	1.193	1.193
39	1.000	1.229	1.229	1.229	1.229
40	1.000	1.168	1.168	1.168	1.170
41	1.000	1.179	1.179	1.181	1.181
42	1.000	1.000	1.000	1.000	1.000
43	1.000	1.086	1.086	1.086	1.086
44	1.000	1.229	1.229	1.229	1.229
45	1.000	1.094	1.094	1.094	1.096
46	1.000	1.044	1.044	1.044	1.046
47	1.000	1.054	1.054	1.054	1.054
48	1.000	1.105	1.105	1.105	1.107
49	1.000	1.301	1.301	1.303	1.303
50	1.000	1.118	1.118	1.118	1.118

Table 11 Performance ratio comparison of GST and CADIA (sample 1-50).

AUCTION NUMBER	GST	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
51	1.000	1.481	1.481	1.481	1.481
52	1.000	1.658	1.658	1.658	1.658
53	1.000	1.339	1.339	1.339	1.339
54	1.000	1.171	1.171	1.171	1.171
55	1.000	1.236	1.236	1.236	1.236
56	1.000	1.273	1.273	1.273	1.273
57	1.000	1.000	1.000	1.000	1.000
58	1.000	1.359	1.359	1.359	1.362
59	1.000	1.202	1.204	1.205	1.205
60	1.000	1.118	1.121	1.121	1.121
61	1.000	1.269	1.269	1.269	1.269
62	1.000	1.110	1.113	1.113	1.113
63	1.000	1.168	1.168	1.168	1.168
64	1.000	1.225	1.225	1.225	1.225
65	1.000	1.102	1.102	1.102	1.102
66	1.000	1.121	1.121	1.122	1.122
67	1.000	1.102	1.102	1.102	1.102
68	1.000	1.214	1.217	1.217	1.217
69	1.000	1.207	1.207	1.207	1.207
70	1.000	1.306	1.306	1.306	1.306
71	1.000	1.221	1.221	1.222	1.222
72	1.000	1.255	1.255	1.258	1.258
73	1.000	1.388	1.388	1.388	1.388
74	1.000	1.114	1.114	1.114	1.114
75	1.000	1.319	1.319	1.319	1.319
76	1.000	1.671	1.671	1.671	1.671
77	1.000	1.298	1.298	1.298	1.301
78	1.000	1.182	1.182	1.182	1.182
79	1.000	1.191	1.191	1.191	1.191
80	1.000	1.221	1.221	1.221	1.221
81	1.000	1.190	1.192	1.192	1.192
82	1.000	1.256	1.256	1.256	1.256
83	1.000	1.495	1.495	1.495	1.495
84	1.000	1.048	1.050	1.051	1.051
85	1.000	1.113	1.113	1.113	1.113
86	1.000	1.240	1.243	1.243	1.243
87	1.000	1.048	1.048	1.048	1.048
88	1.000	1.154	1.154	1.154	1.154
89	1.000	1.169	1.169	1.172	1.172
90	1.000	1.100	1.102	1.102	1.102
91	1.000	1.256	1.256	1.256	1.256
92	1.000	1.272	1.275	1.275	1.275
93	1.000	1.255	1.255	1.255	1.255
94	1.000	1.359	1.359	1.359	1.359
95	1.000	1.051	1.053	1.053	1.053
96	1.000	1.111	1.111	1.111	1.111
97	1.000	1.169	1.169	1.169	1.169
98	1.000	1.171	1.171	1.171	1.171
99	1.000	1.242	1.243	1.243	1.243
100	1.000	1.239	1.239	1.242	1.242

Table 12 Performance ratio comparison of GST and CADIA (sample 51-100).

AUCTION NUMBER	GST	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
101	1.000	1.117	1.117	1.117	1.117
102	1.000	1.108	1.108	1.108	1.108
103	1.000	1.060	1.060	1.060	1.060
104	1.000	1.322	1.322	1.322	1.324
105	1.000	1.202	1.205	1.205	1.205
106	1.000	1.448	1.448	1.451	1.451
107	1.000	1.053	1.053	1.053	1.053
108	1.000	1.167	1.167	1.167	1.169
109	1.000	1.218	1.218	1.218	1.221
110	1.000	1.057	1.059	1.059	1.059
111	1.000	1.051	1.051	1.051	1.054
112	1.000	1.283	1.283	1.283	1.286
113	1.000	1.300	1.300	1.300	1.300
114	1.000	1.057	1.060	1.060	1.060
115	1.000	1.284	1.284	1.286	1.286
116	1.000	1.410	1.410	1.410	1.410
117	1.000	1.105	1.105	1.105	1.105
118	1.000	1.225	1.225	1.225	1.225
119	1.000	1.168	1.168	1.170	1.170
120	1.000	1.061	1.061	1.128	1.128
121	1.000	1.172	1.172	1.172	1.172
122	1.000	1.228	1.228	1.229	1.229
123	1.000	1.105	1.105	1.105	1.105
124	1.000	1.342	1.346	1.346	1.346
125	1.000	1.061	1.061	1.064	1.064
126	1.000	1.342	1.344	1.345	1.345
127	1.000	1.048	1.048	1.048	1.048
128	1.000	1.317	1.317	1.317	1.317
129	1.000	1.238	1.238	1.240	1.240
130	1.000	1.193	1.193	1.193	1.193
131	1.000	1.299	1.299	1.299	1.299
132	1.000	1.360	1.360	1.360	1.360
133	1.000	1.105	1.105	1.105	1.105
134	1.000	1.111	1.111	1.111	1.113
135	1.000	1.195	1.195	1.195	1.195
136	1.000	1.056	1.056	1.056	1.056
137	1.000	1.556	1.556	1.556	1.556
138	1.000	1.228	1.228	1.228	1.228
139	1.000	1.189	1.189	1.189	1.189
140	1.000	1.226	1.226	1.226	1.226
141	1.000	1.323	1.323	1.323	1.323
142	1.000	1.217	1.217	1.217	1.217
143	1.000	1.040	1.040	1.040	1.040
144	1.000	1.659	1.659	1.659	1.663
145	1.000	1.181	1.181	1.181	1.181
146	1.000	1.171	1.171	1.171	1.171
147	1.000	1.051	1.051	1.051	1.051
148	1.000	1.146	1.146	1.146	1.146
149	1.000	1.147	1.147	1.147	1.148
150	1.000	1.131	1.131	1.131	1.131

Table 13 Performance ratio comparison of GST and CADIA (sample 101-150).

AUCTION NUMBER	GST	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
151	1.000	1.117	1.117	1.117	1.117
152	1.000	1.108	1.108	1.108	1.108
153	1.000	1.060	1.060	1.060	1.060
154	1.000	1.322	1.322	1.322	1.324
155	1.000	1.202	1.205	1.205	1.205
156	1.000	1.448	1.448	1.451	1.451
157	1.000	1.053	1.053	1.053	1.053
158	1.000	1.167	1.167	1.167	1.169
159	1.000	1.218	1.218	1.218	1.221
160	1.000	1.057	1.059	1.059	1.059
161	1.000	1.051	1.051	1.051	1.054
162	1.000	1.283	1.283	1.283	1.286
163	1.000	1.300	1.300	1.300	1.300
164	1.000	1.057	1.060	1.060	1.060
165	1.000	1.284	1.284	1.286	1.286
166	1.000	1.410	1.410	1.410	1.410
167	1.000	1.105	1.105	1.105	1.105
168	1.000	1.225	1.225	1.225	1.225
169	1.000	1.168	1.168	1.170	1.170
170	1.000	1.061	1.061	1.128	1.128
171	1.000	1.172	1.172	1.172	1.172
172	1.000	1.228	1.228	1.229	1.229
173	1.000	1.105	1.105	1.105	1.105
174	1.000	1.342	1.346	1.346	1.346
175	1.000	1.061	1.061	1.064	1.064
176	1.000	1.342	1.344	1.345	1.345
177	1.000	1.048	1.048	1.048	1.048
178	1.000	1.317	1.317	1.317	1.317
179	1.000	1.238	1.238	1.240	1.240
180	1.000	1.193	1.193	1.193	1.193
181	1.000	1.299	1.299	1.299	1.299
182	1.000	1.360	1.360	1.360	1.360
183	1.000	1.105	1.105	1.105	1.105
184	1.000	1.111	1.111	1.111	1.113
185	1.000	1.195	1.195	1.195	1.195
186	1.000	1.056	1.056	1.056	1.056
187	1.000	1.556	1.556	1.556	1.556
188	1.000	1.228	1.228	1.228	1.228
189	1.000	1.189	1.189	1.189	1.189
190	1.000	1.226	1.226	1.226	1.226
191	1.000	1.323	1.323	1.323	1.323
192	1.000	1.217	1.217	1.217	1.217
193	1.000	1.040	1.040	1.040	1.040
194	1.000	1.659	1.659	1.659	1.663
195	1.000	1.181	1.181	1.181	1.181
196	1.000	1.171	1.171	1.171	1.171
197	1.000	1.051	1.051	1.051	1.051
198	1.000	1.146	1.146	1.146	1.146
199	1.000	1.147	1.147	1.147	1.148
200	1.000	1.131	1.131	1.131	1.131

Table 14 Performance ratio comparison of GST and CADIA (sample 151-200).

	GST	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
Average	1.000	1.186	1.186	1.187	1.187

Table 15 Performance ratio comparison of GST and CADIA.

Table 15 summarizes the average performance ratios. The larger the number of analysis bids used, the better CADIA's performance will be. For instance, in Auction No. 126, CADIA has a performance ratio of 1.342 when no analysis bid is used; it has successfully improved the ratio to 1.344 and 1.345 when two and four analysis bids are used respectively at successive iterations during the winner determination process. Results show that CADIA has an average performance ratio of 1.186, 1.186, 1.187 and 1.187 when zero, two, four and six analysis bids are used respectively. In other words, CADIA outperforms GST by 18.6%, 18.6%, 18.7% and 18.7% when zero, two, four and six analysis bids are used in 200 sample auctions.

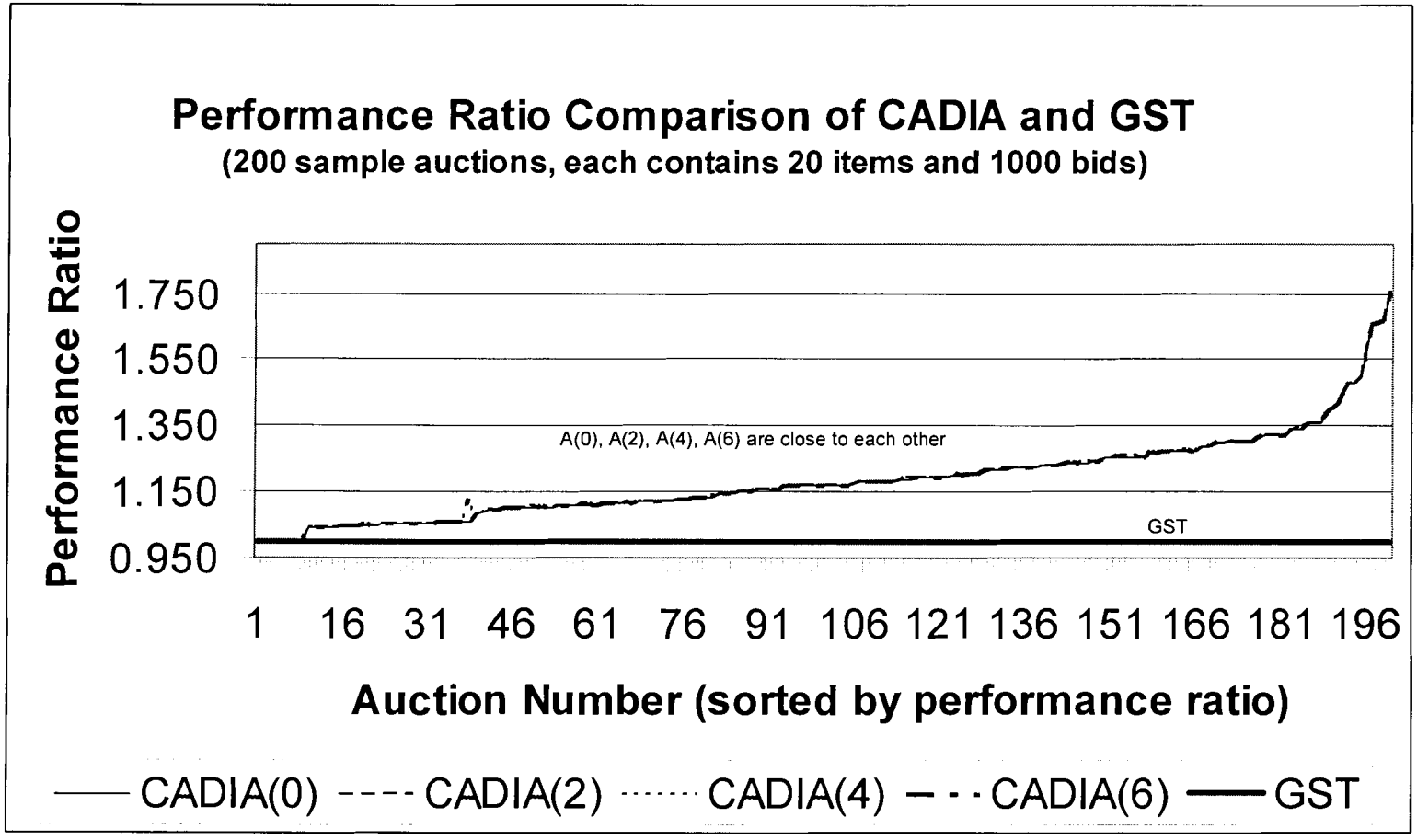

Figure 44 Performance ratio comparison of GST and CADIA.

7.3.2.2 Comparison of CADIA and Four Hill Climbers

Tables 16, 17, 18, and 19 report the results of comparing CADIA with the four climbers, which have also been plotted as line charts (Fig. 45, 46, 47, and 48) for all two hundred sample auctions. The abbreviations A0, A2, A4, and A6 after the word CADIA mean that zero, two, four, and six analysis bids are used. The first strategy of TBE as described in Section 6.2.2 is adopted here to organize the analysis bids. The performance ratio of PRICE is used as a reference for the comparison and is thus set to one.

AUCTION NUMBER	PRICE	N2NORM	KO	DEMAND	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
1	1.000	1.121	1.121	1.239	1.238	1.238	1.238	1.238
2	1.000	1.054	1.054	1.052	1.053	1.054	1.054	1.054
3	1.000	1.000	0.999	0.997	1.000	1.000	1.000	1.000
4	1.000	1.065	1.129	1.127	1.126	1.126	1.126	1.126
5	1.000	1.000	1.000	0.998	1.000	1.000	1.000	1.000
6	1.000	0.999	1.000	1.167	1.167	1.167	1.167	1.167
7	1.000	1.000	1.000	1.117	1.116	1.120	1.120	1.120
8	1.000	1.047	1.047	1.044	1.043	1.043	1.043	1.043
9	1.000	0.999	1.000	0.997	1.000	1.000	1.000	1.000
10	1.000	1.000	1.000	1.041	1.041	1.044	1.044	1.044
11	1.000	1.000	1.000	1.095	1.095	1.095	1.095	1.095
12	1.000	1.108	1.162	1.324	1.323	1.323	1.323	1.323
13	1.000	1.000	1.000	1.130	1.130	1.130	1.130	1.130
14	1.000	1.044	1.044	1.042	1.044	1.044	1.044	1.044
15	1.000	1.170	1.170	1.168	1.168	1.168	1.168	1.168
16	1.000	0.999	1.000	1.111	1.113	1.113	1.113	1.113
17	1.000	1.181	1.122	1.179	1.179	1.179	1.179	1.179
18	1.000	1.113	1.056	1.111	1.113	1.113	1.113	1.113
19	1.000	1.101	1.000	1.201	1.200	1.200	1.200	1.200
20	1.000	0.998	1.051	1.048	1.048	1.050	1.050	1.050
21	1.000	1.107	1.054	1.106	1.106	1.106	1.108	1.108
22	1.000	1.127	1.000	1.126	1.128	1.128	1.128	1.128
23	1.000	1.060	1.059	1.118	1.120	1.120	1.120	1.120
24	1.000	1.242	1.302	1.301	1.301	1.301	1.301	1.301
25	1.000	1.157	1.158	1.154	1.157	1.157	1.157	1.157
26	1.000	0.951	1.000	0.997	1.000	1.000	1.000	1.000
27	1.000	1.276	1.277	1.273	1.276	1.276	1.276	1.276
28	1.000	1.000	1.000	1.051	1.053	1.053	1.053	1.053
29	1.000	1.121	1.121	1.119	1.121	1.121	1.121	1.121
30	1.000	1.051	1.000	1.049	1.048	1.051	1.051	1.051
31	1.000	1.057	1.056	1.054	1.056	1.056	1.056	1.056
32	1.000	1.122	1.122	1.179	1.178	1.178	1.178	1.178
33	1.000	1.206	1.206	1.255	1.255	1.255	1.255	1.257
34	1.000	1.065	1.064	1.127	1.129	1.129	1.129	1.129
35	1.000	1.040	1.000	1.039	1.040	1.040	1.040	1.040
36	1.000	1.102	1.051	1.100	1.099	1.099	1.099	1.099
37	1.000	1.258	1.322	1.321	1.320	1.320	1.323	1.323
38	1.000	1.128	1.128	1.190	1.189	1.192	1.193	1.193
39	1.000	1.172	1.171	1.227	1.229	1.229	1.229	1.229
40	1.000	1.000	1.000	1.168	1.168	1.168	1.168	1.170
41	1.000	1.182	1.181	1.179	1.179	1.179	1.181	1.181
42	1.000	1.000	1.000	0.998	1.000	1.000	1.000	1.000
43	1.000	1.044	0.999	1.086	1.086	1.086	1.086	1.086
44	1.000	1.171	1.228	1.227	1.229	1.229	1.229	1.229
45	1.000	0.951	1.000	1.094	1.094	1.094	1.094	1.096
46	1.000	1.000	1.000	1.044	1.044	1.044	1.044	1.046
47	1.000	0.999	1.057	1.112	1.114	1.114	1.114	1.114
48	1.000	0.999	1.108	1.105	1.105	1.105	1.105	1.107
49	1.000	1.242	1.242	1.301	1.301	1.301	1.303	1.303
50	1.000	1.060	1.060	1.118	1.118	1.118	1.118	1.118

Table 16 Performance ratio comparison of 4 Hill Climbers and CADIA (sample 1-50).

AUCTION NUMBER	PRICE	N2NORM	KO	DEMAND	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
51	1.000	1.483	1.483	1.481	1.481	1.481	1.481	1.481
52	1.000	1.658	1.000	1.655	1.658	1.658	1.658	1.658
53	1.000	1.229	1.171	1.340	1.339	1.339	1.339	1.339
54	1.000	1.171	1.171	1.169	1.171	1.171	1.171	1.171
55	1.000	1.060	1.060	1.237	1.236	1.236	1.236	1.236
56	1.000	1.276	1.069	1.273	1.273	1.273	1.273	1.273
57	1.000	0.951	1.000	0.997	1.000	1.000	1.000	1.000
58	1.000	1.302	1.000	1.359	1.359	1.359	1.359	1.362
59	1.000	1.206	1.207	1.203	1.202	1.204	1.205	1.205
60	1.000	1.121	1.120	1.119	1.118	1.121	1.121	1.121
61	1.000	1.163	1.163	1.269	1.269	1.269	1.269	1.269
62	1.000	1.054	1.057	1.110	1.110	1.113	1.113	1.113
63	1.000	1.171	0.999	1.168	1.168	1.168	1.168	1.168
64	1.000	1.228	1.228	1.225	1.225	1.225	1.225	1.225
65	1.000	1.101	1.000	1.100	1.102	1.102	1.102	1.102
66	1.000	1.121	1.000	1.119	1.121	1.121	1.122	1.122
67	1.000	1.000	1.051	1.100	1.102	1.102	1.102	1.102
68	1.000	1.162	1.054	1.214	1.214	1.217	1.217	1.217
69	1.000	1.068	1.206	1.204	1.207	1.207	1.207	1.207
70	1.000	1.206	1.206	1.306	1.306	1.306	1.306	1.306
71	1.000	1.137	1.136	1.135	1.137	1.137	1.138	1.138
72	1.000	1.193	1.257	1.255	1.255	1.255	1.257	1.257
73	1.000	1.388	1.193	1.385	1.388	1.388	1.388	1.388
74	1.000	1.061	1.121	1.180	1.182	1.182	1.182	1.182
75	1.000	1.257	1.064	1.319	1.319	1.319	1.319	1.319
76	1.000	1.596	1.671	1.668	1.671	1.671	1.671	1.671
77	1.000	1.240	1.000	1.298	1.298	1.298	1.298	1.301
78	1.000	1.121	1.000	1.179	1.182	1.182	1.182	1.182
79	1.000	1.000	1.064	1.191	1.191	1.191	1.191	1.191
80	1.000	1.221	1.219	1.218	1.221	1.221	1.221	1.221
81	1.000	1.128	1.191	1.190	1.190	1.192	1.192	1.192
82	1.000	1.259	1.257	1.256	1.256	1.256	1.256	1.256
83	1.000	1.464	1.037	1.533	1.495	1.495	1.495	1.495
84	1.000	1.050	1.051	1.048	1.048	1.050	1.051	1.051
85	1.000	1.000	1.056	1.111	1.113	1.113	1.113	1.113
86	1.000	1.243	1.244	1.240	1.240	1.243	1.243	1.243
87	1.000	0.948	1.000	1.048	1.048	1.048	1.048	1.048
88	1.000	1.000	1.051	1.152	1.154	1.154	1.154	1.154
89	1.000	1.114	1.171	1.169	1.169	1.169	1.171	1.171
90	1.000	1.000	1.000	1.100	1.100	1.102	1.102	1.102
91	1.000	1.258	1.194	1.256	1.256	1.256	1.256	1.256
92	1.000	1.205	1.274	1.273	1.272	1.275	1.275	1.275
93	1.000	1.206	1.257	1.255	1.255	1.255	1.255	1.255
94	1.000	1.362	1.301	1.360	1.359	1.359	1.359	1.359
95	1.000	1.054	1.054	1.052	1.051	1.053	1.053	1.053
96	1.000	1.000	1.000	1.111	1.111	1.111	1.111	1.111
97	1.000	1.057	1.171	1.169	1.169	1.169	1.169	1.169
98	1.000	1.172	1.171	1.169	1.171	1.171	1.171	1.171
99	1.000	1.181	1.181	1.240	1.242	1.243	1.243	1.243
100	1.000	1.242	1.242	1.239	1.239	1.239	1.242	1.242

Table 17 Performance ratio comparison of 4 Hill Climbers and CADIA (sample 51-100).

AUCTION NUMBER	PRICE	N2NORM	KO	DEMAND	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
101	1.000	1.119	1.060	1.118	1.117	1.117	1.117	1.117
102	1.000	1.000	1.107	1.106	1.108	1.108	1.108	1.108
103	1.000	1.000	0.999	1.061	1.060	1.060	1.060	1.060
104	1.000	1.217	1.271	1.323	1.322	1.322	1.322	1.324
105	1.000	1.206	1.206	1.202	1.202	1.205	1.205	1.205
106	1.000	1.385	1.449	1.448	1.448	1.448	1.451	1.451
107	1.000	1.053	1.054	1.051	1.053	1.053	1.053	1.053
108	1.000	1.113	1.114	1.168	1.167	1.167	1.167	1.169
109	1.000	1.221	1.222	1.219	1.218	1.218	1.218	1.221
110	1.000	1.059	1.059	1.057	1.057	1.059	1.059	1.059
111	1.000	1.000	1.000	1.052	1.051	1.051	1.051	1.054
112	1.000	1.286	1.057	1.284	1.283	1.283	1.283	1.286
113	1.000	1.242	1.122	1.301	1.300	1.300	1.300	1.300
114	1.000	1.060	1.059	1.057	1.057	1.060	1.060	1.060
115	1.000	1.229	1.229	1.284	1.284	1.284	1.286	1.286
116	1.000	1.412	1.412	1.410	1.410	1.410	1.410	1.410
117	1.000	0.999	0.999	1.105	1.105	1.105	1.105	1.105
118	1.000	1.114	1.114	1.225	1.225	1.225	1.225	1.225
119	1.000	1.000	1.169	1.168	1.168	1.168	1.170	1.170
120	1.000	1.128	1.129	1.126	1.061	1.061	1.128	1.128
121	1.000	1.171	1.171	1.169	1.172	1.172	1.172	1.172
122	1.000	1.228	1.228	1.226	1.228	1.228	1.229	1.229
123	1.000	1.053	1.107	1.106	1.105	1.105	1.105	1.105
124	1.000	1.345	1.276	1.343	1.342	1.346	1.346	1.346
125	1.000	1.000	0.999	0.998	0.997	0.997	1.000	1.000
126	1.000	1.345	1.207	1.342	1.342	1.344	1.345	1.345
127	1.000	1.048	1.048	1.046	1.048	1.048	1.048	1.048
128	1.000	1.238	1.000	1.314	1.317	1.317	1.317	1.317
129	1.000	1.181	1.181	1.238	1.238	1.238	1.240	1.240
130	1.000	0.999	1.000	1.054	1.056	1.056	1.056	1.056
131	1.000	1.182	1.001	1.300	1.300	1.300	1.300	1.300
132	1.000	1.302	1.182	1.360	1.360	1.360	1.360	1.360
133	1.000	1.000	1.054	1.105	1.105	1.105	1.105	1.105
134	1.000	1.113	1.000	1.111	1.111	1.111	1.111	1.113
135	1.000	1.000	1.049	1.193	1.195	1.195	1.195	1.195
136	1.000	0.938	1.000	1.057	1.056	1.056	1.056	1.056
137	1.000	1.559	1.559	1.556	1.556	1.556	1.556	1.556
138	1.000	1.171	1.114	1.226	1.228	1.228	1.228	1.228
139	1.000	1.128	1.127	1.190	1.189	1.189	1.189	1.189
140	1.000	1.228	1.227	1.226	1.226	1.226	1.226	1.226
141	1.000	1.259	1.260	1.321	1.323	1.323	1.323	1.323
142	1.000	1.218	1.218	1.215	1.217	1.217	1.217	1.217
143	1.000	0.999	1.042	1.040	1.040	1.040	1.040	1.040
144	1.000	1.663	1.663	1.661	1.659	1.659	1.659	1.663
145	1.000	1.182	1.181	1.179	1.181	1.181	1.181	1.181
146	1.000	1.172	1.171	1.169	1.171	1.171	1.171	1.171
147	1.000	1.051	1.051	1.048	1.051	1.051	1.051	1.051
148	1.000	1.146	1.147	1.144	1.146	1.146	1.146	1.146
149	1.000	1.000	1.147	1.145	1.147	1.147	1.147	1.148
150	1.000	1.088	1.044	1.131	1.131	1.131	1.131	1.131

Table 18 Performance ratio comparison of 4 Hill Climbers and CADIA (sample 101-150).

AUCTION NUMBER	PRICE	N2NORM	KO	DEMAND	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
151	1.000	1.128	1.128	1.126	1.125	1.128	1.128	1.128
152	1.000	1.103	1.205	1.204	1.204	1.204	1.204	1.204
153	1.000	1.103	1.154	1.152	1.152	1.152	1.152	1.152
154	1.000	1.000	1.000	0.998	1.000	1.000	1.000	1.000
155	1.000	1.049	1.049	1.145	1.145	1.145	1.145	1.145
156	1.000	1.194	1.193	1.191	1.191	1.191	1.191	1.193
157	1.000	1.056	1.056	1.111	1.110	1.110	1.110	1.110
158	1.000	1.000	1.000	1.270	1.272	1.272	1.273	1.273
159	1.000	1.257	1.257	1.255	1.255	1.258	1.258	1.258
160	1.000	1.000	1.000	0.997	1.000	1.000	1.000	1.000
161	1.000	1.000	1.060	1.057	1.057	1.059	1.060	1.060
162	1.000	1.054	1.054	1.105	1.105	1.105	1.107	1.107
163	1.000	1.106	1.000	1.158	1.158	1.158	1.158	1.158
164	1.000	1.241	1.000	1.239	1.239	1.242	1.242	1.242
165	1.000	0.902	0.999	1.047	1.046	1.046	1.046	1.046
166	1.000	1.109	0.999	1.160	1.160	1.160	1.160	1.160
167	1.000	1.051	1.051	1.152	1.152	1.152	1.152	1.152
168	1.000	1.108	1.108	1.159	1.159	1.159	1.159	1.159
169	1.000	1.000	1.220	1.219	1.218	1.218	1.218	1.222
170	1.000	1.121	1.120	1.180	1.179	1.181	1.181	1.182
171	1.000	1.759	1.758	1.756	1.759	1.759	1.759	1.759
172	1.000	1.304	1.184	1.302	1.304	1.304	1.304	1.304
173	1.000	1.000	1.000	0.997	1.000	1.000	1.000	1.000
174	1.000	1.258	1.259	1.256	1.256	1.258	1.258	1.258
175	1.000	1.069	1.069	1.134	1.137	1.137	1.137	1.137
176	1.000	1.208	1.277	1.274	1.274	1.274	1.277	1.277
177	1.000	1.000	1.000	1.106	1.107	1.107	1.107	1.108
178	1.000	1.000	1.128	1.190	1.192	1.192	1.192	1.192
179	1.000	1.054	1.108	1.269	1.268	1.270	1.270	1.271
180	1.000	1.258	1.257	1.256	1.255	1.255	1.258	1.258
181	1.000	1.121	1.061	1.118	1.118	1.118	1.118	1.118
182	1.000	1.048	1.049	1.046	1.048	1.048	1.048	1.048
183	1.000	1.400	1.401	1.397	1.401	1.401	1.401	1.401
184	1.000	1.053	1.054	1.159	1.159	1.159	1.159	1.159
185	1.000	1.196	1.196	1.242	1.242	1.242	1.242	1.242
186	1.000	1.121	1.120	1.118	1.118	1.118	1.120	1.120
187	1.000	1.182	1.181	1.180	1.179	1.179	1.179	1.179
188	1.000	1.482	1.483	1.480	1.480	1.480	1.482	1.482
189	1.000	1.137	1.136	1.135	1.134	1.134	1.137	1.137
190	1.000	1.051	1.051	1.202	1.202	1.202	1.202	1.202
191	1.000	1.206	1.275	1.272	1.274	1.276	1.276	1.276
192	1.000	1.046	1.000	1.091	1.092	1.092	1.092	1.092
193	1.000	1.074	1.293	1.293	1.293	1.293	1.293	1.293
194	1.000	1.054	1.053	1.051	1.051	1.053	1.053	1.053
195	1.000	1.000	1.000	1.095	1.095	1.095	1.095	1.095
196	1.000	1.194	1.194	1.192	1.191	1.194	1.194	1.194
197	1.000	1.228	1.229	1.226	1.226	1.226	1.226	1.226
198	1.000	1.061	1.061	1.178	1.180	1.180	1.180	1.180
199	1.000	1.277	1.137	1.273	1.273	1.273	1.276	1.276
200	1.000	1.055	1.113	1.111	1.111	1.111	1.111	1.111

Table 19 Performance ratio comparison of 4 Hill Climbers and CADIA (sample 151-200).

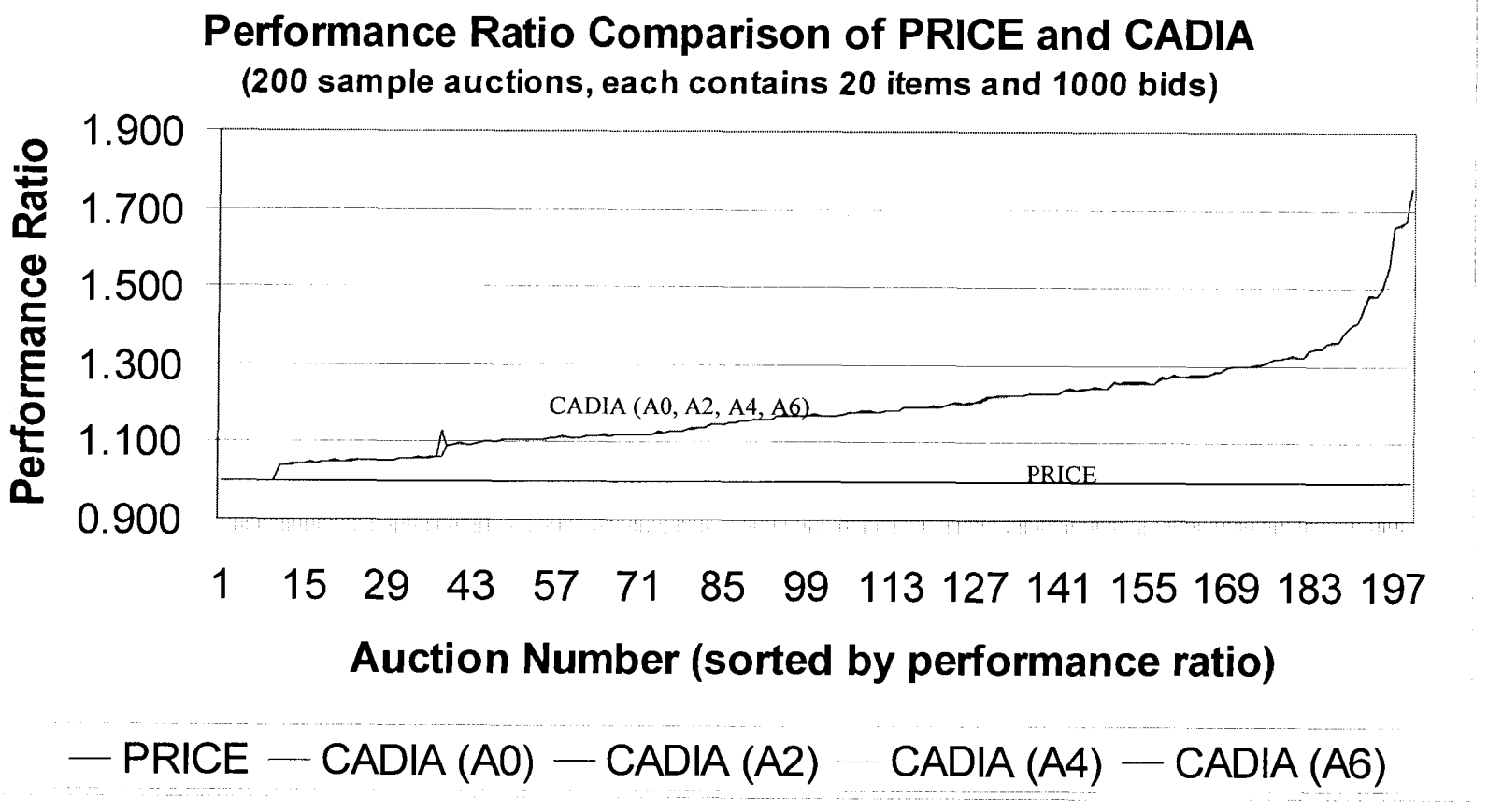

Figure 45 Performance ratio comparison of PRICE and CADIA.

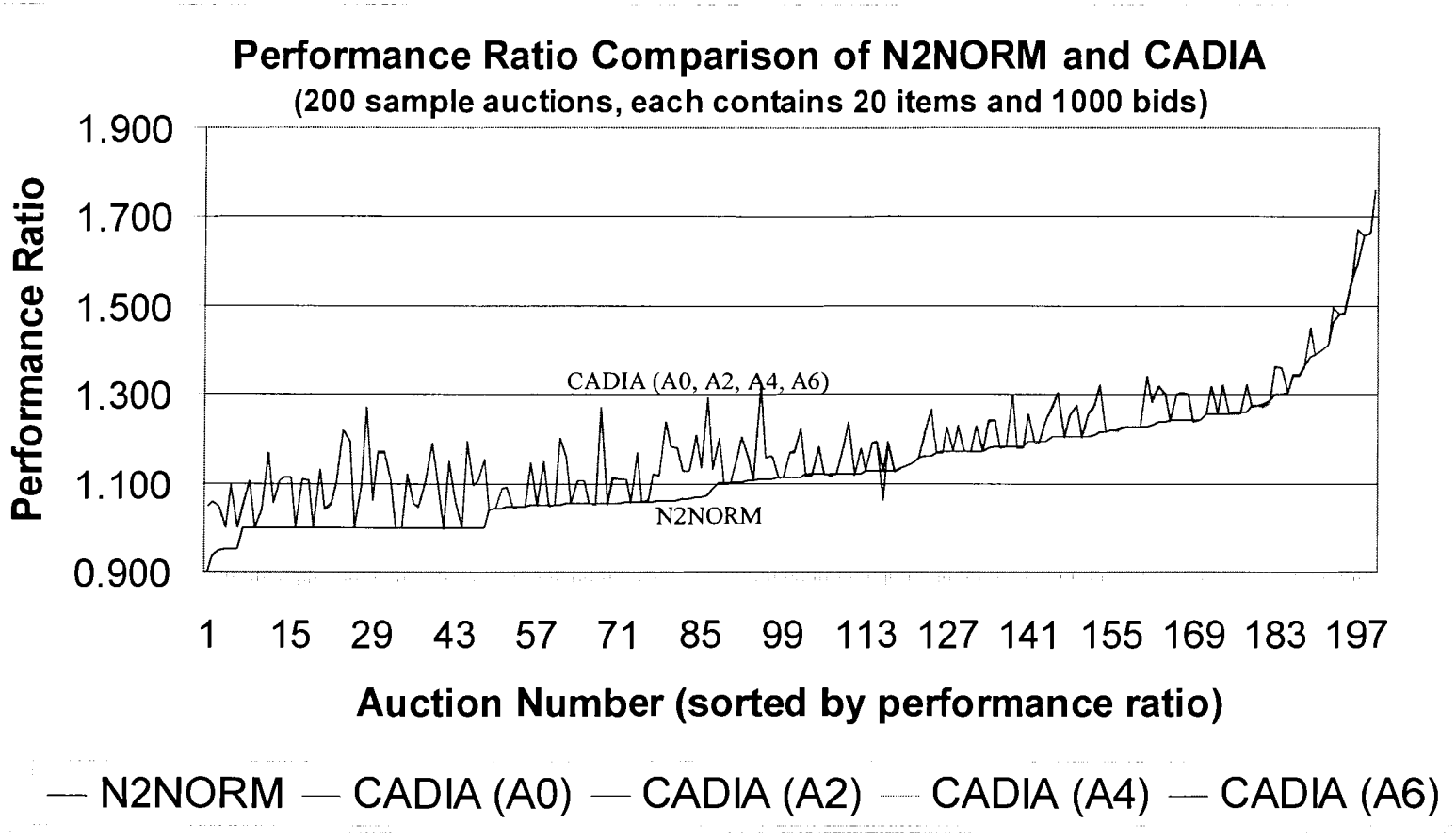

Figure 46 Performance ratio comparison of N2NORM and CADIA.

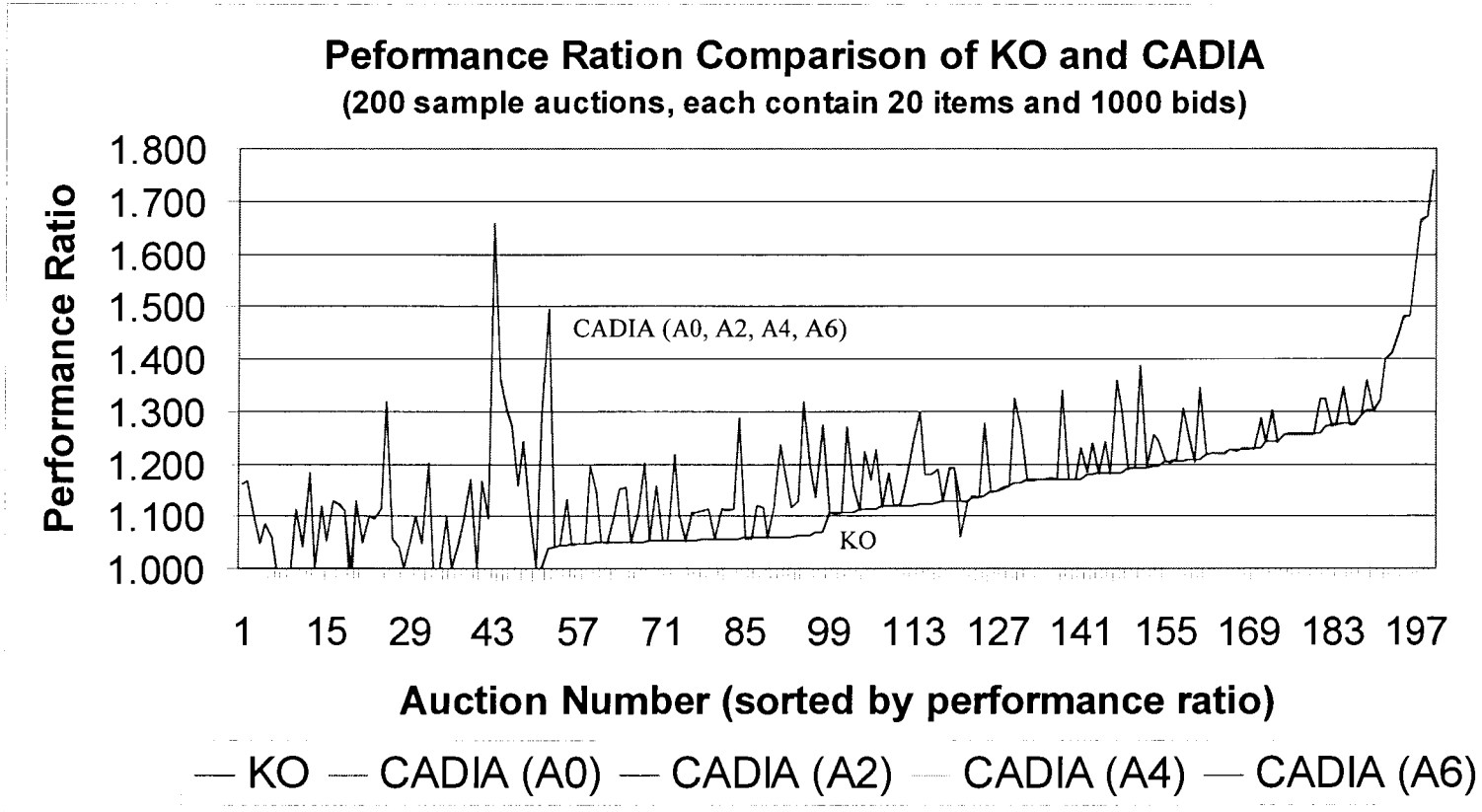

Figure 47 Performance ratio comparison of KO and CADIA.

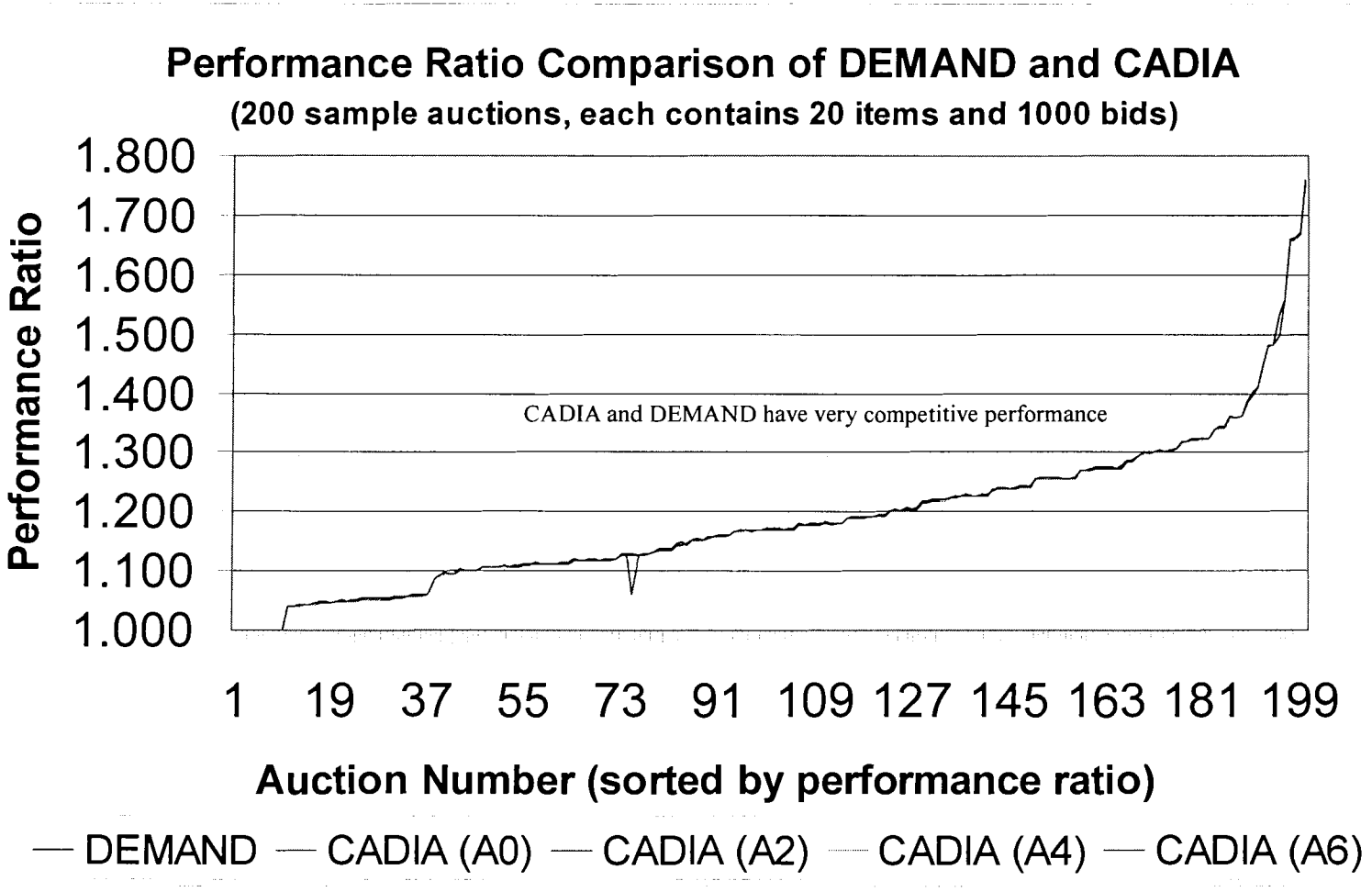

Figure 48 Performance ratio comparison of DEMAND and CADIA.

	PRICE	N2NORM	KO	DEMAND	CADIA (A0)	CADIA (A2)	CADIA (A4)	CADIA (A6)
Average	1.000	1.190	1.191	1.186	1.185	1.186	1.187	1.187

Table 20 Performance ratio comparison of hill climbers and CADIA.

Table 20 summarizes the average performance ratios. On average, DEMAND, which had a performance ratio of 1.186, performed best among all hill climbers. On the other hand, results show that CADIA had an average performance ratio of 1.185, 1.186, 1.187 and 1.187 when zero, two, four and six analysis bids were used respectively. We also recorded that in all 200 auctions, the number of auctions that CADIA achieved better revenue than PRICE, N2NORM, KO and DEMAND were 191, 148, 148 and 133 respectively (Table 21).

	CADIA vs PRICE	CADIA vs N2NORM	CADIA vs KO	CADIA vs DEMAND
Total=200	191	148	148	133

Table 21 Number of Auction that CADIA outperforms the hill climbers.

7.3.2.3 Comparison of CADIA and ESG

Tables 22 and 23 report the results of comparing CADIA with ESG, which have also been plotted as line chart (Figure. 49) for all two hundred sample auctions. The abbreviation A6 after the word CADIA means that six analysis bids are used. The first strategy of TBE as described in Section 6.2.2 is adopted here to organize the analysis bids. Since ESG is able to obtain better results at successive iterations of execution, for a fair comparison, we ran ESG in 460 iterations to match the execution time of CADIA.

The performance ratio of CADIA (A6) is used as a reference for the comparison and is thus set to one.

AUCTION NUMBER	CADIA(A6)	ESG	AUCTION NUMBER	CADIA(A6)	ESG
1	1.000	1.000	51	1.000	1.000
2	1.000	0.998	52	1.000	0.999
3	1.000	0.997	53	1.000	1.000
4	1.000	1.000	54	1.000	0.999
5	1.000	0.998	55	1.000	1.000
6	1.000	1.000	56	1.000	1.000
7	1.000	0.998	57	1.000	0.998
8	1.000	1.000	58	1.000	1.000
9	1.000	0.998	59	1.000	0.998
10	1.000	0.998	60	1.000	0.998
11	1.000	1.000	61	1.000	1.000
12	1.000	1.000	62	1.000	0.998
13	1.000	1.000	63	1.000	1.000
14	1.000	0.999	64	1.000	1.000
15	1.000	1.000	65	1.000	0.999
16	1.000	0.999	66	1.000	0.998
17	1.000	1.000	67	1.000	0.998
18	1.000	0.998	68	1.000	0.998
19	1.000	1.000	69	1.000	0.998
20	1.000	0.999	70	1.000	1.000
21	1.000	0.997	71	1.000	0.999
22	1.000	0.998	72	1.000	0.999
23	1.000	0.998	73	1.000	0.999
24	1.000	1.000	74	1.000	0.998
25	1.000	0.998	75	1.000	1.000
26	1.000	0.998	76	1.000	0.998
27	1.000	0.999	77	1.000	0.998
28	1.000	0.998	78	1.000	0.946
29	1.000	0.998	79	1.000	1.000
30	1.000	0.999	80	1.000	0.998
31	1.000	0.998	81	1.000	0.999
32	1.000	1.000	82	1.000	1.000
33	1.000	0.998	83	1.000	1.369
34	1.000	0.998	84	1.000	0.998
35	1.000	0.998	85	1.000	0.999
36	1.000	1.000	86	1.000	0.999
37	1.000	0.999	87	1.000	1.000
38	1.000	0.998	88	1.000	0.999
39	1.000	0.998	89	1.000	0.999
40	1.000	1.000	90	1.000	0.999
41	1.000	0.999	91	1.000	1.000
42	1.000	0.998	92	1.000	0.999
43	1.000	1.000	93	1.000	1.000
44	1.000	0.998	94	1.000	1.000
45	1.000	0.998	95	1.000	0.999
46	1.000	0.999	96	1.000	1.000
47	1.000	0.998	97	1.000	1.000
48	1.000	1.000	98	1.000	0.999
49	1.000	0.999	99	1.000	0.999
50	1.000	1.000	100	1.000	0.998

Table 22 Performance ratio comparison of ESG and CADIA (sample 1-100).

AUCTION NUMBER	CADIA(A6)	ESG	AUCTION NUMBER	CADIA(A6)	ESG
101	1.000	1.000	151	1.000	0.998
102	1.000	0.999	152	1.000	1.000
103	1.000	1.000	153	1.000	1.000
104	1.000	0.999	154	1.000	0.998
105	1.000	0.998	155	1.000	1.000
106	1.000	0.998	156	1.000	0.999
107	1.000	0.998	157	1.000	1.000
108	1.000	1.000	158	1.000	0.998
109	1.000	0.998	159	1.000	0.999
110	1.000	0.998	160	1.000	0.997
111	1.000	0.998	161	1.000	0.998
112	1.000	0.999	162	1.000	0.999
113	1.000	1.000	163	1.000	1.000
114	1.000	0.998	164	1.000	0.998
115	1.000	0.999	165	1.000	1.000
116	1.000	1.000	166	1.000	1.000
117	1.000	1.000	167	1.000	1.000
118	1.000	1.000	168	1.000	1.000
119	1.000	0.999	169	1.000	0.998
120	1.000	0.941	170	1.000	0.999
121	1.000	0.998	171	1.000	0.999
122	1.000	0.998	172	1.000	0.999
123	1.000	1.000	173	1.000	0.997
124	1.000	0.997	174	1.000	0.998
125	1.000	0.998	175	1.000	0.999
126	1.000	0.998	176	1.000	0.998
127	1.000	0.998	177	1.000	0.999
128	1.000	0.998	178	1.000	0.999
129	1.000	0.999	179	1.000	0.999
130	1.000	0.997	180	1.000	0.998
131	1.000	1.000	181	1.000	1.000
132	1.000	1.000	182	1.000	0.998
133	1.000	1.000	183	1.000	0.997
134	1.000	0.998	184	1.000	1.000
135	1.000	0.999	185	1.000	1.000
136	1.000	1.000	186	1.000	0.999
137	1.000	1.000	187	1.000	1.000
138	1.000	0.999	188	1.000	0.999
139	1.000	1.000	189	1.000	0.998
140	1.000	1.000	190	1.000	1.000
141	1.000	0.998	191	1.000	0.998
142	1.000	0.999	192	1.000	0.998
143	1.000	1.000	193	1.000	1.000
144	1.000	0.999	194	1.000	0.999
145	1.000	0.998	195	1.000	1.000
146	1.000	0.999	196	1.000	0.998
147	1.000	0.998	197	1.000	1.000
148	1.000	0.999	198	1.000	0.998
149	1.000	0.997	199	1.000	0.998
150	1.000	1.000	200	1.000	1.000

Table 23 Performance ratio comparison of ESG and CADIA (sample 101-200).

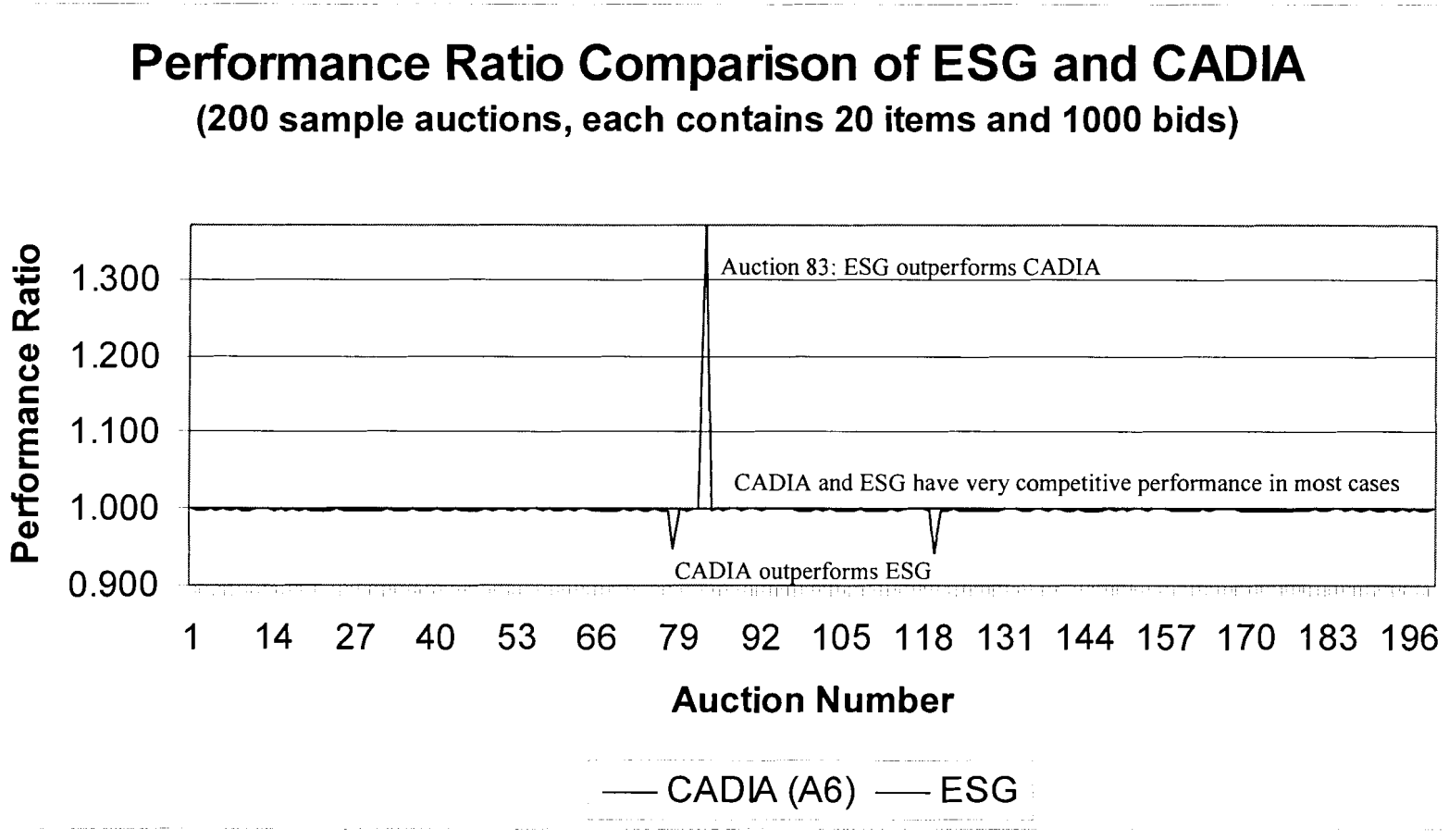

Figure 49 Performance ratio comparison of ESG and CADIA.

Results show that CADIA and ESG are very competitive because their average performance ratios are 1.0000 and 1.0002 respectively. We also recorded that in all 200 auctions, the number of auctions where CADIA outperformed ESG was 130 but the difference of the achieved revenue is within 0.3% in most cases. There was only 1 out of 200 auctions where ESG outperformed CADIA. Both achieved the same revenue in 69 out of 200 cases.

Since the 200 sample auctions do not cover all possible bid patterns, we may not conclude that CADIA is better than other approximation systems in all cases. However, from the empirical results, we found that CADIA can achieve better revenue than the hill climbers and the ESG in many cases. As a result, the evaluation shows that CADIA has its contribution to the CA determination problem.

7.3.3 Running Time Measurement of CADIA

In Section 7.2.3, a test plan for measuring the efficiency of CADIA by time clocking is presented. The sample sizes of the auctioned items and bids are selected in the ranges of 100 to 500 and 200 to 2000 respectively. There are a total of five hundred sample auctions grouped into fifty categories of size. Each category has the same number of items and bids. Table 24 summarizes the results of the average running time of CADIA for each category. Figure 50 depicts the running time for different numbers of items when the number of bids are fixed.

Time(s)	ITEMS				
BIDS	100	200	300	400	500
200	28.881	73.495	143.576	240.626	337.615
400	33.438	130.027	229.860	361.680	573.054
600	42.701	138.589	324.076	415.667	762.446
800	46.416	177.956	356.472	681.169	883.560
1000	56.711	201.199	406.675	710.311	1239.763
1200	67.887	244.702	527.198	905.021	1385.392
1400	70.826	353.298	680.048	1142.580	1470.334
1600	76.039	393.926	1089.306	1335.971	1790.274
1800	85.763	493.580	1156.873	1631.647	2363.959
2000	113.994	569.248	1559.152	2406.329	3142.839

Table 24 CADIA's Average running time in 50 different sizes of auction.

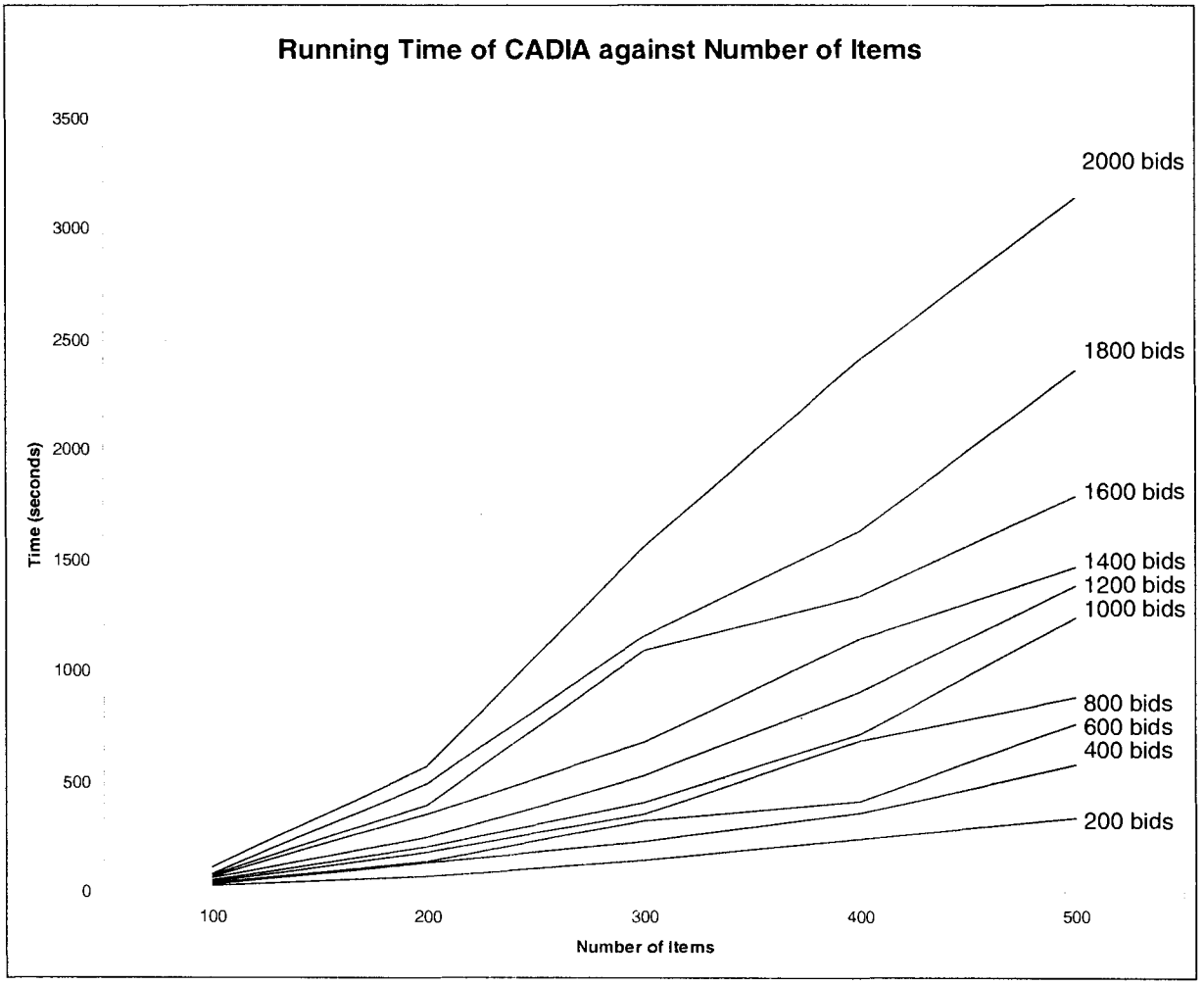

Figure 50 Running time of CADIA for different number of items.

As expected, the running time increases when the number of items or bids increases.

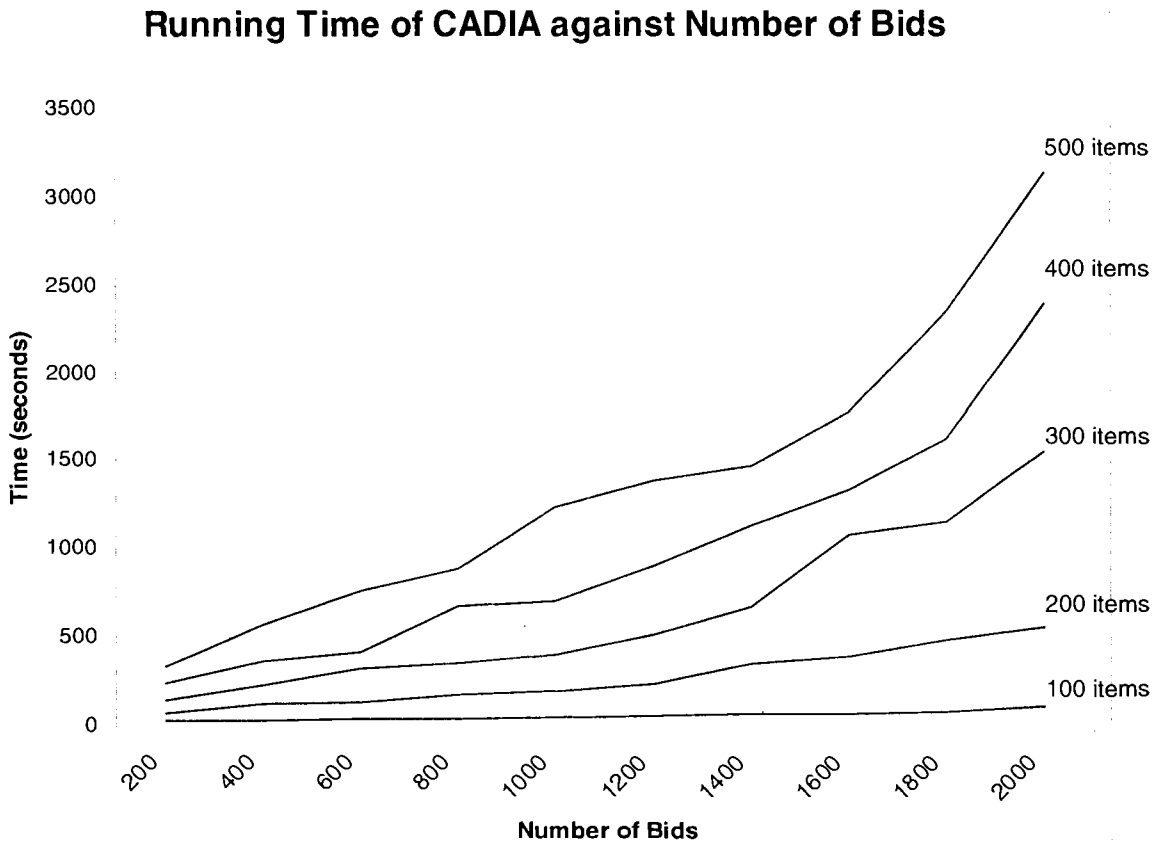

Figure 51 Running time of CADIA for different number of bids.

Similarly, Figure 51 depicts the running time for different numbers of bids when the number of items are fixed. The running time of CADIA grows more rapidly than a linear function.

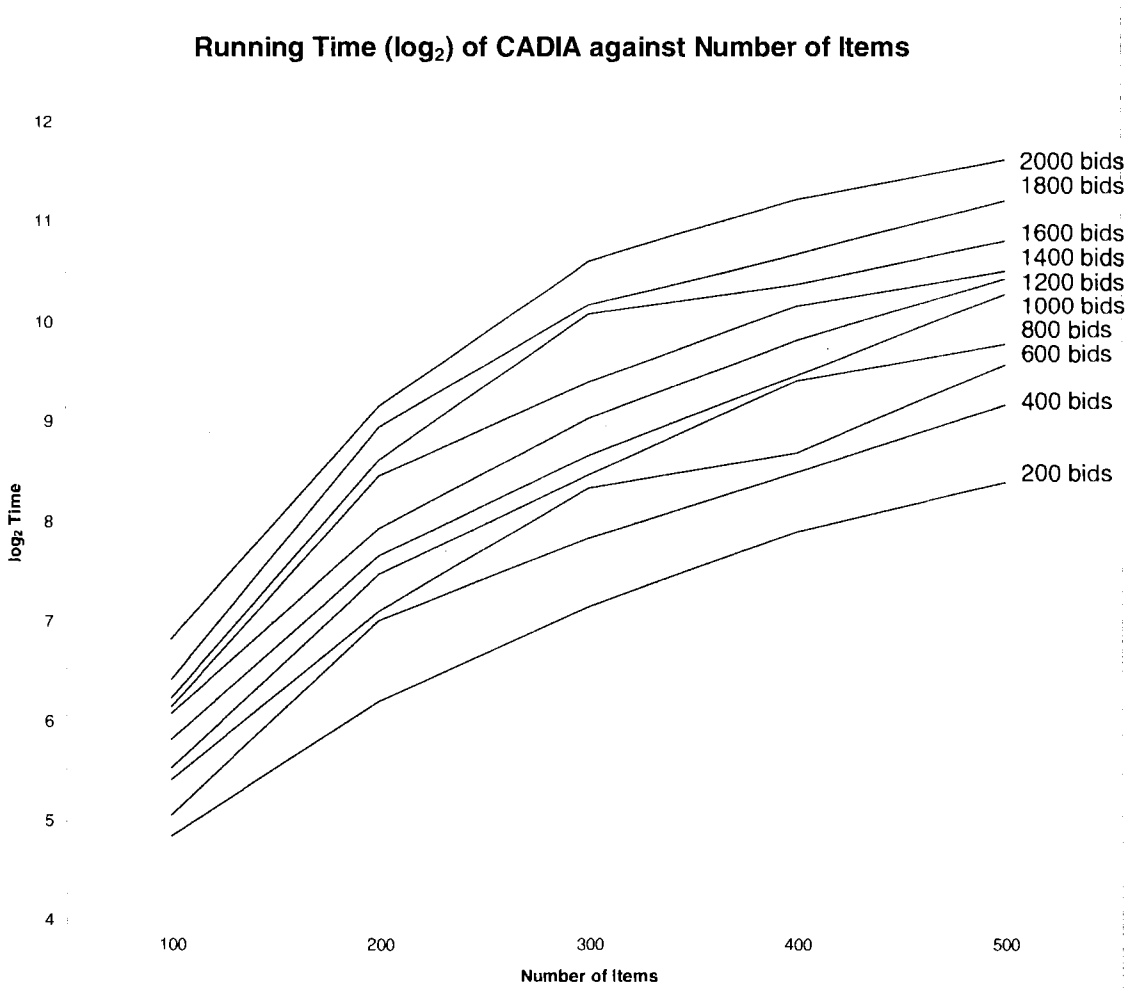

Figure 52 Logarithm of running time of CADIA for different number of items

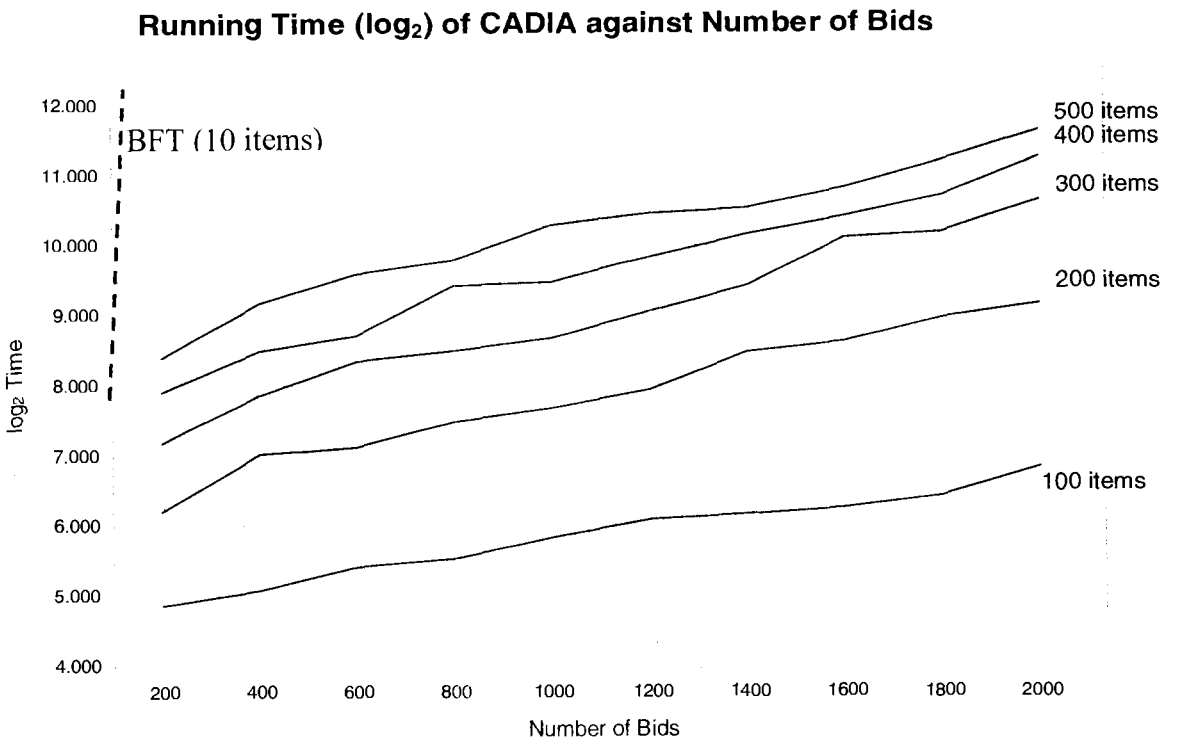

Figure 53 Logarithm of the running time of CADIA for different number of bids.

Figure 52 and 53 shows the results for Figure 50 and 51 in logarithmic scale respectively. The shapes of the graphs in Figure 52 and 53, which appear to be concave and straight lines respectively, suggest that CADIA's efficiency in auctions of no more than 500 items and 2000 bids are bounded by the complexity class of $O(2^n)$. Although both CADIA and BFT belong to the complexity class of $O(2^n)$, CADIA's running time grows much slower than that of the BFT (Figure 53). When BFT is used, in an auction of 10 items and 10 bids the winner determination time takes about 212 seconds but 1800 seconds with one additional item. If these values are converted into logarithm of base 2,

we will obtain 7.72 and 10.81 respectively. The graph of BFT has a much steeper slope and Y-intercept value as drafted in dashed line in Figure 53.

CHAPTER EIGHT:
DISSCUSSION

8.1 Discussion on CADIA's Performance

Although the 200 sample auctions do not cover all possible bid patterns, it allows us to identify some bid patterns that CADIA performs better or worse than others. We will analyse 2 cases that demonstrate the advantages and shortcomings of CADIA and other evaluated techniques. The results of such an analysis will be considered in CADIA's future enhancement.

In sample Auction-78, the revenue generated using CADIA, PRICE, N2NORM, KO, DEMAND, and ESG were $1935.830, $1637.610, $1835.670, $1636.870, $1931.300, and $1831.000 respectively. When the bid patterns are examined, it was found that both PRICE and KO included the highest price bid (bid-11) as one of its winners. It is due to the fact that the objective functions of both depend directly on bid prices. As a result, both PRICE and KO were trapped in local optima during the search. N2NORM uses both the bid price and bid size as the parameters in its objective function. That is, the higher the bid price per item the bid offers, the higher the chance it becomes a winner. Thus, bid-140 became one of its winners since it offered over $100 per item as compared to only $50 offered by other bids. However, such an objective function may also lead to a local optimum result. DEMAND weights a bid based on its bid price per item versus the prices offered by others for that item. Since DEMAND took its

neighbour's evaluation into consideration, it generated a better result in Auction-78 than other climbers. Rather than depending directly on the bid price in its objective function as found in some of the climbers, ESG starts with a candidate solution and generates a new solution using an update rule based on gradient. Since ESG relies on gradient information, the result could get stuck at a local optimum. Although CADIA, DEMAND and ESG were very competitive in Auction-78, CADIA, which does not rely on any local search techniques, was able to obtain the best result among all techniques. It was due to the fact that the search space in Auction-78 contained many local optima and all other methods are stuck at some local optima during the search.

A different result was observed in sample Auction-83. The revenue generated using CADIA, PRICE, N2NORM, KO, DEMAND, and ESG were $2077.000, $1389.160, $2033.530, $1441.160, $2129.900 and $2843.000 respectively. Both PRICE and KO were trapped in local optima during the search, which ended up much lower revenue, due to the same reason as described above (bid -222 offered the highest bid price among all). Since N2NORM considered also the bid size, it did not consider bid-222 a winner. Both DEMAND and ESG outperformed CADIA in this auction. When the winner pattern of the best performer ESG was examined, it was found that the winners were those who wanted one or two items. Since CADIA's objective function is based on the relaxation of the concept of itemsets, CADIA was misled by some bids that included the least frequent itemsets but offered lower bid prices.

In summary, the hill-climbers exploit the best available solution for possible improvement but neglect exploring a large portion of the search space. Gradient-based search methods are well-known for situations when the search space has a bowl shape.

When it is not the case, they could get stuck at local optima since the primary consideration relies on gradient information. In many cases, the success or failure of many hill-climbers and some gradient-based methods is determined by the initial start point. For problems with many local optima, particularly those where these optima have large basins of attraction, it's often very difficult to locate a globally optimal solution. CADIA attempts to explore the search space thoroughly but foregoes exploiting promising regions of the space. In addition, the cost required for CADIA to generate its knowledge makes CADIA runs much slower than all evaluated approximate techniques. For instance, the current implementation of CADIA takes two minutes to solve a problem of size of 100 items and 2000 bids, but almost an hour when the number of items is increased to 500. On the other hand, all evaluated approximate techniques takes less than a minutes to solve problems of size of hundred of items and thousands of bids.

8.2. Discussion on CADIA's Practicality

All sample auctions described in the thesis contain no more than five hundred items and two thousands bids. Such an upper limit of sample size for testing a proposed system has been adopted in most current research because it becomes uncommon to have a CA selling more than five hundred non-identical items. However, it may be common to have more than two thousand bids in a CA, especially if it is held on the Internet. The best CA winner determination system is one that always generates the optimal revenue (best accuracy) with the shortest running time (best performance) among all proposed

systems. Some techniques such as BFT focus primarily on the accuracy, while others such as GST focus solely on the performance.

CADIA has been evaluated in terms of its accuracy in Chapter 7. With the adjustments made at the TBE, CADIA is capable of finding the optimal revenue. The idea is to use analysis bids for revenue improvement at successive iterations during the winner determination process. In an auction of thousands of bids, it becomes impractical to analyze all bids if the winners have to be announced within minutes after the expiry of bids submission. Thus, CADIA has been designed to accept the number of analysis bids as an argument from the user during its execution. If more time is allowed, a more accurate result can be obtained by including more analysis bids.

If performance in terms of running time is the only concern, a greedy search based system is perhaps the most practical system because its objective function is based on the search for the highest bidding price. Such a system performs very well only in auctions of too many items but too few bids in which the chance of having bid conflicts is very low. In auctions of two hundred items and five thousand bids, for example, a greedy search based system may be trapped in a local maximum due to a high degree of conflicts among bids. Other domain-based heuristic systems may include a pre-processing step before they apply the core winner determination algorithms. For example, a system that has removed ninety percent of the bids during its pre-processing phase can definitely determine the winners in seconds even in auction of hundreds of items and thousands of bids. As a result, it becomes misleading to evaluate a CA technique or system based only on its performance but not accuracy.

CADIA has been evaluated in terms of its performance in Chapter 7. The graphs of CADIA's running time on sample auctions of up to 500 items and 2000 bids suggest that CADIA is bounded by the efficiency class $O(2^n)$. The inclusion of the scale factor variable C_{sf} (Section 6.2.1) is to address the practicality issue when the winner determination time is a major concern. By adjusting the value of C_{sf}, CADIA is able to determine winners in minutes or even in seconds. C_{sf} has been set to 0.5 by default and it is believed that bids with an unacceptably low bidding price will be rejected. That is, a bid whose bidding price is lower than half of the lower bound price is rejected. C_{sf}'s value domain is between 0 and 1. When it is set to 1, more bids are rejected because the full lower bound price is used instead. When it is set to 0, all bidding prices will not be checked. Figure 48 shows the relationships between running time and number of bids for different C_{sf} in auctions of 500 items.

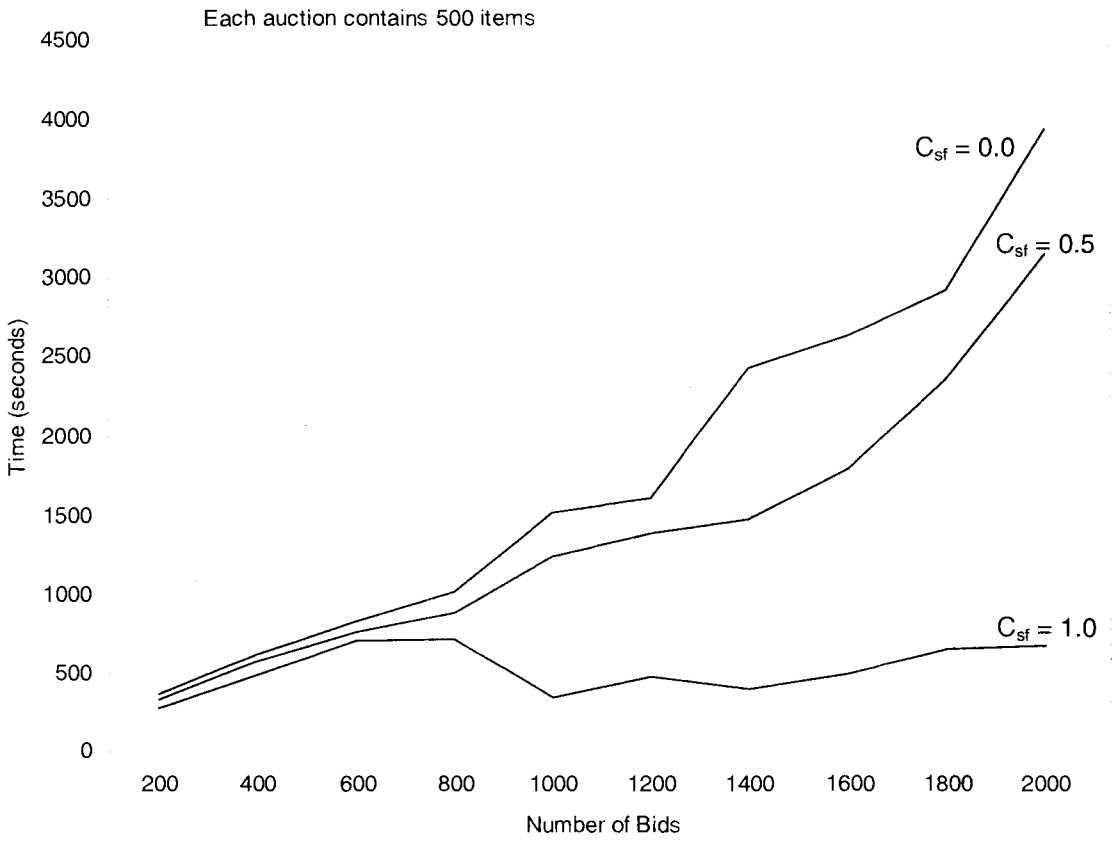

Figure 54 Running time of CADIA for different values of C_{sf}.

In Figure 48, the graphs show that the running time grows much slower when C_{sf} is set to 1 than it is set to 0.5 or 0.0. Thus, an auctioneer will have an option to trade off accuracy for efficiency. However, in some cases, a non-zero C_{sf} may accidentally reject a valuable bid and cause a non-optimal result. Thus, the value of C_{sf} must be selected with caution. The strategy is to compare the revenue generated for different values of C_{sf}. For example, C_{sf} can be started with 1, and is then decremented by 0.25 in each subsequent

test. The decrement continues only when the revenue is improved, and more importantly, when the time is allowed.

CHAPTER NINE:
CONCLUSION AND FUTURE WORK

The subjects of this thesis are the proposal and design of a novel and practical combinatorial auction winner determination approach using item association. The approach was developed and implemented into the system called CADIA. The thesis has reviewed the characteristics and benefits of CAs and surveyed the state of knowledge and techniques for solving the winner determination problem, followed by the hypothesis and core algorithms of the new approach, and its design and implementation. CADIA consists of four major components. In the first component PRE, redundant bids are removed. In the second component IAG, qualified bids output from PRE are used as seeds to generate candidate itemsets. IAG attempts to identify the smallest and least wanted itemsets from the candidate itemsets. In the third component WIN, the output from IAG is used in the identification of candidate and potential winners. In the fourth component TBE, both potential winners and possible losers are used for further analysis and improvement.

The empirical results show that CADIA is a practical technique that is able to handle CA auctions of hundreds of items and thousands of bids. The study of its accuracy and performance in terms of revenue generation and running time shows that it has met the criteria and goals of the design in achieving good approximate results. Although both CADIA and BFT belong to the class of $O(2^n)$, CADIA's running time grows much slower than that of the BFT. CADIA was found to be a good approximation system because its accuracy can be improved when analysis bids are used.

Since the 200 sample auctions do not cover all possible bid patterns, we may not conclude that CADIA is better than other approximation systems in all cases. However, from the empirical results, we found that CADIA can achieve better revenue than the hill climbers and the ESG in many cases. As a result, the evaluation shows that CADIA has its contribution to the CA determination problem. The current limitation of CADIA is that it runs slower than all the evaluated approximation techniques. Such a slower response time is due to the obvious fact that CADIA's core knowledge requires some time to generate.

There are a number of possible extensions and enhancements for the work presented here. CADIA's optimal strategy in the winner determination process can be seen as a number of iterations running the same set of algorithms on auction data from where tactical bids are removed. The number of iterations depends directly on the number of analysis bids selected. From the empirical results and observations, TBE with the second strategy as described in Section 6.2.2 has been very successful in searching for better revenue using the potential winners and possible losers as the knowledge. Since all potential winners and possible losers are the output from WIN during the first iteration of the process, it is possible to distribute all subsequent iterations of search on a number of processors. For example, if 4 analysis bids are used, there are $(2^4 - 1)$ or fifteen ways of removing the tactical bids from the original bid file which translates into additional fifteen iterations of search. Instead of running CADIA fifteen times sequentially on a single processor, it may be desirable to run CADIA on a multi-processor system, a multi-threading system, or even on multiple machines to further improve its performance.

CADIA's optimal strategy is based on the knowledge discovered from the auction data. Historical data from previous auctions will definitely help to enhance such knowledge. That is, when auctioned items are distributed in a similar pattern from auction to auction, it is possible to have the knowledge accumulated and used in future auction winner determination. Besides the item association technique, other data mining techniques such as clustering and decision trees can also be applied to structure the knowledge in a different way to aid in the revenue search.

Since each search space is different for each auction in CA problems, there seems no way to choose a single search method that can serve well in every case. Nevertheless each technique offers its own merit. Better solutions to CA problems can often be obtained by hybridizing different approaches. Effective search techniques should provide a mechanism for balancing the conflicting objectives: exploiting the optimal solutions and at the same time exploring the search space.

APPENDIX A

Sample Bid Input File

This appendix provides a sample bid input file containing one thousand bids and one hundred items, and the output files summarizing the results.

Input File: 1000 bids and 100 items

```
{0} {88 90} {102.30}
{1} {90 65} {102.30}
{2} {88 47} {102.30}
{3} {32} {52.00}
{4} {55} {52.00}
{5} {67 90} {203.30}
{6} {90 24} {103.30}
{7} {84 49} {103.30}
{8} {49 45} {103.30}
{9} {87} {52.00}
{10} {0} {51.00}
{11} {27} {151.00}
{12} {20 63 70 76} {9.28}
{13} {76 73 80 62} {8.28}
{14} {56 36 33 25} {306.28}
{15} {36 44 84 88} {406.28}
{16} {25 2 90 43} {406.28}
{17} {33 86 98 81} {207.28}
{18} {2} {51.00}
{19} {93 95 51 5} {306.28}
{20} {93 9 33 10} {307.28}
{21} {51 32 88 85} {206.28}
{22} {95 71 39 4} {107.28}
{23} {11 59} {103.30}
{24} {90 52} {103.30}
{25} {90 64} {103.30}
{26} {29} {52.00}
{27} {92 59 86} {55.74}
{28} {33 96} {302.30}
{29} {96 27} {202.30}
{30} {33 79} {103.30}
{31} {42 95} {103.30}
{32} {83 96 70} {55.74}
{33} {70 7 89} {55.74}
{34} {83 45 59} {55.74}
{35} {94 86 26 41 53} {258.90}
{36} {53 94 87 73 51} {358.90}
{37} {94 73 8 42 80} {158.90}
{38} {62 0} {203.30}
{39} {0 66} {203.30}
{40} {74} {151.00}
{41} {56 26} {102.30}
{42} {91} {51.00}
{43} {83 21 97 2 60} {358.90}
{44} {2 48 47 59 81} {259.90}
{45} {21 75 37 5 90} {258.90}
{46} {60 0 63 17 19} {158.90}
{47} {83 25 60 27 30} {158.90}
{48} {23 82 40 91 6 25} {510.59}
{49} {91 74 73 84 31 34} {608.59}
{50} {82 12 94 70 60 81} {510.59}
{51} {6 69 34 2 37 41} {210.59}
{52} {25} {151.00}
{53} {86 18} {3.30}
{54} {22 4} {202.30}
{55} {22 12} {202.30}
```

{56} {4 98} {103.30}
{57} {23} {52.00}
{58} {33 79} {203.30}
{59} {33 62} {203.30}
{60} {94} {51.00}
{61} {24} {52.00}
{62} {87} {52.00}
{63} {11} {151.00}
{64} {29} {52.00}
{65} {73 17} {202.30}
{66} {17 51} {202.30}
{67} {73 66} {103.30}
{68} {16 11} {103.30}
{69} {0} {51.00}
{70} {23 39} {4.30}
{71} {39 55} {4.30}
{72} {23 81} {4.30}
{73} {13 77} {4.30}
{74} {45 14} {103.30}
{75} {28} {151.00}
{76} {15} {52.00}
{77} {94 83} {103.30}
{78} {94 1} {102.30}
{79} {10} {52.00}
{80} {89 10 3 50 8 81} {411.59}
{81} {50 92 9 73 90 61} {411.59}
{82} {3 58 79 32 5 72} {312.59}
{83} {10 50 32 60 98 72} {312.59}
{84} {81 4 49 45 26 95} {310.59}
{85} {8 57 99 32 93 77} {111.59}
{86} {49} {52.00}
{87} {35} {52.00}
{88} {14} {52.00}
{89} {52} {52.00}
{90} {93 48} {3.30}
{91} {46 4} {103.30}
{92} {73 6} {103.30}
{93} {73 43} {103.30}
{94} {6 8} {3.30}
{95} {80 81} {4.30}
{96} {72} {52.00}
{97} {50} {151.00}
{98} {32 54 76 25} {8.28}
{99} {1 59} {203.30}
{100} {1 40} {302.30}
{101} {70} {52.00}
{102} {69 26} {202.30}
{103} {26 98} {203.30}
{104} {69 15} {103.30}
{105} {51} {51.00}
{106} {97 54 60 22 6} {359.90}
{107} {54 0 97 53 5} {360.90}
{108} {22 73 48 16 90} {358.90}
{109} {97 3 4 2 77} {258.90}
{110} {6 44 61 17 19} {159.90}
{111} {83 31} {103.30}
{112} {91 87} {103.30}
{113} {80 39} {4.30}
{114} {39 13} {4.30}
{115} {80 21} {3.30}
{116} {5 53} {104.30}
{117} {5 36} {104.30}
{118} {53 1} {103.30}
{119} {44 59} {103.30}
{120} {44 79} {3.30}
{121} {43} {52.00}
{122} {49 84 8 82} {306.28}
{123} {84 95 28 47} {305.28}
{124} {54 11} {3.30}
{125} {54 84} {3.30}
{126} {32} {52.00}
{127} {84 12} {202.30}
{128} {12 42} {103.30}
{129} {84 36} {103.30}
{130} {30 44 3} {254.74}
{131} {44 93 42} {254.74}
{132} {30 52 91} {155.74}
{133} {3 46 18} {55.74}
{134} {68 46} {4.30}
{135} {21} {151.00}
{136} {21 55 41 53 80} {260.90}
{137} {21 40 58 97 73} {357.90}
{138} {41 28 51 38 10} {259.90}
{139} {53 45 98 36 50} {159.90}
{140} {54} {52.00}
{141} {5} {52.00}
{142} {58 70} {3.30}
{143} {0 35 91 83} {207.28}
{144} {91 1 83 52} {307.28}
{145} {83 64 46 2} {208.28}
{146} {0 54 78 46} {8.28}
{147} {35 32 3 31} {7.28}
{148} {98} {52.00}
{149} {10 0} {103.30}
{150} {0 52} {103.30}
{151} {10 2} {3.30}
{152} {96 77 60 49 62} {59.90}
{153} {34 63} {103.30}
{154} {63 1} {103.30}
{155} {34 97} {3.30}
{156} {40 47} {102.30}
{157} {47 54} {103.30}
{158} {40 17} {102.30}
{159} {50} {151.00}
{160} {28 41 61} {55.74}
{161} {61 93 46} {55.74}
{162} {41 82 87} {55.74}
{163} {18 5} {104.30}

{164} {18 43} {104.30}
{165} {5 28} {103.30}
{166} {16 84} {3.30}
{167} {84 98} {3.30}
{168} {35 22 99} {154.74}
{169} {22 73 21} {153.74}
{170} {99 45 62} {54.74}
{171} {37} {151.00}
{172} {32 76 4 25} {207.28}
{173} {76 66 90 34} {207.28}
{174} {4 38 83 60} {107.28}
{175} {25 80 57 95} {107.28}
{176} {99} {151.00}
{177} {61} {52.00}
{178} {89 15} {4.30}
{179} {79} {52.00}
{180} {58} {51.00}
{181} {74 25 2} {453.74}
{182} {25 37 98} {254.74}
{183} {84} {51.00}
{184} {69 9 79 58} {107.28}
{185} {61} {52.00}
{186} {58 89 70 34 2} {558.90}
{187} {2 15 97 34 4} {458.90}
{188} {58 2 55 64 83} {359.90}
{189} {34 35 22 31 43} {158.90}
{190} {70 91 0 48 11} {158.90}
{191} {0 49} {103.30}
{192} {49 99} {3.30}
{193} {28 66 50 13 70} {59.90}
{194} {93 0} {202.30}
{195} {0 29} {203.30}
{196} {93 61} {3.30}
{197} {2 69} {302.30}
{198} {69 8} {302.30}
{199} {2 8} {302.30}
{200} {96} {151.00}
{201} {12 80} {3.30}
{202} {25 9} {103.30}
{203} {65} {151.00}
{204} {94} {51.00}
{205} {85 10 82 76 26} {158.90}
{206} {76 22 6 94 30} {159.90}
{207} {82 14 92 2 65} {158.90}
{208} {85 57 33 40 18} {58.90}
{209} {4} {51.00}
{210} {85 88 2 92 15} {458.90}
{211} {2 66 52 47 68} {359.90}
{212} {85 31 10 83 54} {359.90}
{213} {15 41 40 8 25} {258.90}
{214} {88 30 53 51 50} {258.90}
{215} {55 75} {104.30}
{216} {55 78} {104.30}
{217} {75 51} {103.30}
{218} {97} {52.00}
{219} {28 23} {103.30}
{220} {45 61 57} {155.74}
{221} {4} {151.00}
{222} {53 1 11} {354.74}
{223} {1 44 74} {453.74}
{224} {11 64 1} {354.74}
{225} {53 71 93} {55.74}
{226} {94} {51.00}
{227} {17 63 50 68 73 5 11} {463.33}
{228} {63 88 90 69 95 85 36} {662.33}
{229} {50 44 54 26 74 9 67} {363.33}
{230} {11 68 33 5 63 45 57} {264.33}
{231} {17 74 65 43 14 77 67} {164.33}
{232} {68 25 36 66 18 57 50} {65.33}
{233} {2 5} {3.30}
{234} {64 25} {203.30}
{235} {64 69} {203.30}
{236} {25 88} {202.30}
{237} {38} {52.00}
{238} {86} {151.00}
{239} {82 26 93} {253.74}
{240} {26 49 90} {54.74}
{241} {74 18} {103.30}
{242} {74 8} {102.30}
{243} {79 49 26 74} {307.28}
{244} {74 98 22 2} {406.28}
{245} {79 2 7 19} {307.28}
{246} {26 48 59 58} {107.28}
{247} {49 93 26 80} {107.28}
{248} {46 58 34 41 86} {58.90}
{249} {41 21 3 36 49} {59.90}
{250} {46 2 0 43 49} {59.90}
{251} {73 92 70 0 7 94 32} {263.33}
{252} {94 48 77 14 65 53 23} {165.33}
{253} {73 18 67 88 39 96 72} {164.33}
{254} {32 94 91 10 39 20 81} {65.33}
{255} {94 38 74 85} {306.28}
{256} {38 96 8 70} {307.28}
{257} {74 86 30 89} {207.28}
{258} {85 40 46 47} {206.28}
{259} {2} {51.00}
{260} {50 56} {302.30}
{261} {56 21} {302.30}
{262} {50 9} {203.30}
{263} {3} {51.00}
{264} {75 55} {4.30}
{265} {55 14} {4.30}
{266} {75 53} {4.30}
{267} {34} {151.00}
{268} {1 58 56 37} {505.28}
{269} {1 24 50 25} {406.28}
{270} {56 19 31 29} {307.28}
{271} {37 69 19 75} {307.28}

{272} {0} {151.00}
{273} {69} {151.00}
{274} {31} {51.00}
{275} {22 1 67 45 91 75} {310.59}
{276} {75 94 68 28 49 34} {411.59}
{277} {67 33 76 96 95 57} {411.59}
{278} {45 55 13 53 51 56} {411.59}
{279} {91 25 41 50 55 95} {410.59}
{280} {22 41 43 46 59 91} {212.59}
{281} {28 61 80 75} {108.28}
{282} {80 24 45 52} {8.28}
{283} {25 56 50 35 74 41} {210.59}
{284} {35 63 26 90 15 67} {312.59}
{285} {56 73 38 39 96 60} {310.59}
{286} {82} {51.00}
{287} {31 72} {203.30}
{288} {72 0} {203.30}
{289} {31 36} {203.30}
{290} {81 90 22 67 56} {358.90}
{291} {22 58 89 65 88} {457.90}
{292} {67 46 7 90 35} {359.90}
{293} {81 44 13 26 60} {358.90}
{294} {56 16 78 8 64} {259.90}
{295} {90 42 64 86 35} {159.90}
{296} {15} {52.00}
{297} {60 56 94} {353.74}
{298} {60 88 67} {254.74}
{299} {94 85 68} {154.74}
{300} {11 99} {202.30}
{301} {99 64} {203.30}
{302} {11 68} {103.30}
{303} {53 15} {4.30}
{304} {53 19} {4.30}
{305} {15 86} {3.30}
{306} {51} {151.00}
{307} {28} {151.00}
{308} {49 53 17 19 88} {359.90}
{309} {88 37 0 70 94} {457.90}
{310} {17 42 88 40 13} {358.90}
{311} {49 81 31 60 16} {259.90}
{312} {53 22 38 63 59} {160.90}
{313} {89 63 25 80} {8.28}
{314} {82 7} {202.30}
{315} {82 6} {103.30}
{316} {7 4} {102.30}
{317} {4} {151.00}
{318} {73 44 97 51} {206.28}
{319} {44 83 24 82} {307.28}
{320} {51 34 67 13} {307.28}
{321} {52} {52.00}
{322} {87 2} {103.30}
{323} {2 5} {103.30}
{324} {3} {51.00}
{325} {75 24} {104.30}
{326} {75 70} {104.30}
{327} {10 5} {4.30}
{328} {5 49} {4.30}
{329} {46 23 52 51 67} {160.90}
{330} {23 90 98 22 57} {159.90}
{331} {51 29 48 79 8} {159.90}
{332} {46 29 19 7 95} {159.90}
{333} {52 81 59 37 8} {59.90}
{334} {64} {52.00}
{335} {56 86 88 61} {306.28}
{336} {56 45 38 81} {407.28}
{337} {61 73 93 39} {207.28}
{338} {88 26 92 63} {207.28}
{339} {7} {51.00}
{340} {23 89 2} {55.74}
{341} {23 30 34} {55.74}
{342} {98 62} {4.30}
{343} {74} {51.00}
{344} {76 69 51} {154.74}
{345} {60 4 96 26 70 31} {409.59}
{346} {60 28 71 11 2 85} {609.59}
{347} {96 19 1 31 40 39} {410.59}
{348} {70 65 69 15 86 26} {410.59}
{349} {31 29 17 62 88 89} {311.59}
{350} {0} {51.00}
{351} {20 18} {4.30}
{352} {18 19} {4.30}
{353} {44 66 70} {55.74}
{354} {70 14 44} {55.74}
{355} {66 99 46} {55.74}
{356} {83} {52.00}
{357} {27 72 9} {255.74}
{358} {27 29 10} {255.74}
{359} {9 65 5} {55.74}
{360} {72 18 73} {55.74}
{361} {78} {52.00}
{362} {31 41 49 59 97 11} {212.59}
{363} {59 89 94 45 26 80} {311.59}
{364} {31 8 74 66 14 61} {311.59}
{365} {11 63 76 40 22 75} {111.59}
{366} {41 6 54 35 90 72} {13.59}
{367} {83} {52.00}
{368} {60} {151.00}
{369} {28 38 96 2 24 8} {510.59}
{370} {28 56 45 85 22 60} {708.59}
{371} {2 49 74 64 54 82} {411.59}
{372} {96 15 49 62 97 12} {312.59}
{373} {24 76 25 3 73 83} {311.59}
{374} {8 3 23 72 89 18} {212.59}
{375} {45 95 77} {354.74}
{376} {95 66 94} {254.74}
{377} {77 37 3} {254.74}
{378} {45 69 71} {254.74}
{379} {61} {52.00}

{380} {92 20 14} {156.74}
{381} {92 79 11} {55.74}
{382} {12} {151.00}
{383} {83 95} {203.30}
{384} {95 45} {302.30}
{385} {97 9} {4.30}
{386} {9 67} {4.30}
{387} {69 39 57 72} {208.28}
{388} {72 32 25 84} {307.28}
{389} {69 92 55 84} {307.28}
{390} {57 35 48 0} {108.28}
{391} {39 62 60 27} {107.28}
{392} {15 84 90} {254.74}
{393} {90 11 55} {154.74}
{394} {84 55 47} {54.74}
{395} {81 16} {104.30}
{396} {81 79} {104.30}
{397} {1} {151.00}
{398} {52 83 80} {56.74}
{399} {83 92 39} {56.74}
{400} {52 19 45} {55.74}
{401} {4} {51.00}
{402} {73 79 71} {155.74}
{403} {71 38 85} {155.74}
{404} {4} {151.00}
{405} {96} {151.00}
{406} {28 50} {202.30}
{407} {28 9} {203.30}
{408} {50 78} {103.30}
{409} {57} {52.00}
{410} {45} {51.00}
{411} {17} {151.00}
{412} {74} {51.00}
{413} {70 60 39 20 25} {259.90}
{414} {39 49 40 87 12} {359.90}
{415} {20 51 72 17 75} {259.90}
{416} {60 81 58 40 32} {258.90}
{417} {25 16 68 72 31} {59.90}
{418} {8 37 78} {354.74}
{419} {37 58 89} {254.74}
{420} {78 7 71} {155.74}
{421} {8 10 49} {55.74}
{422} {80 83} {4.30}
{423} {83 95} {3.30}
{424} {1} {151.00}
{425} {47} {151.00}
{426} {82 65 11 2 90 69 5 79} {414.13}
{427} {69 31 77 52 36 19 21 91} {616.13}
{428} {11 9 33 82 29 88 40 28} {614.13}
{429} {90 79 0 18 93 29 43 89} {417.13}
{430} {65 47 25 98 19 71 62 84} {416.13}
{431} {79 11 19 28 98 8 44 96} {415.13}
{432} {22 4 8} {353.74}
{433} {8 4 95} {353.74}
{434} {54 85 53} {55.74}
{435} {54 33 38} {55.74}
{436} {50 97} {103.30}
{437} {30 9 28 78 55} {60.90}
{438} {28 53 46 54 49} {60.90}
{439} {74 89} {103.30}
{440} {61 31 82} {354.74}
{441} {82 3 41} {254.74}
{442} {61 60 88} {254.74}
{443} {31 75 14} {155.74}
{444} {40 82 74} {353.74}
{445} {82 60 66} {254.74}
{446} {40 16 47} {254.74}
{447} {24 26} {103.30}
{448} {21} {151.00}
{449} {41 92 4 15} {108.28}
{450} {41} {52.00}
{451} {11 74 99 12 17} {256.90}
{452} {74 93 75 13 85} {358.90}
{453} {99 41 84 75 2} {158.90}
{454} {17 1 39 71 11} {158.90}
{455} {11 4 99 71 57} {158.90}
{456} {52 51} {103.30}
{457} {51 32} {103.30}
{458} {8 27 94} {253.74}
{459} {27 61 85} {254.74}
{460} {8 25 69} {253.74}
{461} {94} {151.00}
{462} {2} {151.00}
{463} {96} {51.00}
{464} {28 46 96} {154.74}
{465} {96 77 7} {154.74}
{466} {85 56 80} {154.74}
{467} {85 76 67} {55.74}
{468} {30 10 50 79 44 60 56 90} {315.13}
{469} {50 61 74 48 11 6 88 44} {415.13}
{470} {44 54 77 9 29 20 3 1} {317.13}
{471} {56 21 85 63 36 28 15 68} {316.13}
{472} {60 21 39 66 11 64 87 76} {217.13}
{473} {79 69 71 36 62 65 85 53} {217.13}
{474} {51} {51.00}
{475} {4 16 78} {155.74}
{476} {16 77 93} {155.74}
{477} {78 49 84} {55.74}
{478} {4 62 42} {55.74}
{479} {71} {52.00}
{480} {74 59} {3.30}
{481} {74 98} {3.30}
{482} {99} {51.00}
{483} {40 57 75 80} {8.28}
{484} {57 18 78 51} {8.28}
{485} {40 68 74 16} {7.28}
{486} {73} {51.00}
{487} {2} {151.00}

{488} {38 94 12} {254.74}
{489} {94 49 14} {155.74}
{490} {38 44 57} {155.74}
{491} {12 21 30} {54.74}
{492} {95 82} {202.30}
{493} {82 83} {103.30}
{494} {95 57} {3.30}
{495} {29 95} {203.30}
{496} {95 79} {103.30}
{497} {8} {151.00}
{498} {97} {52.00}
{499} {21 42} {3.30}
{500} {42 74} {3.30}
{501} {90} {51.00}
{502} {74 40 99} {153.74}
{503} {99 1 80} {154.74}
{504} {40 56 65} {153.74}
{505} {78 69} {103.30}
{506} {69 9} {3.30}
{507} {8} {51.00}
{508} {96} {51.00}
{509} {37} {51.00}
{510} {19 11} {103.30}
{511} {19 96} {103.30}
{512} {11 58} {102.30}
{513} {27} {51.00}
{514} {83} {52.00}
{515} {73 64} {3.30}
{516} {73 81} {3.30}
{517} {71 2 56 87} {107.28}
{518} {56 30 75 23} {108.28}
{519} {26} {151.00}
{520} {37} {151.00}
{521} {3} {51.00}
{522} {96 36 75 94 15} {259.90}
{523} {15 97 71 35 73} {160.90}
{524} {36 2 27 7 6} {158.90}
{525} {96 49 8 22 37} {157.90}
{526} {90 29} {103.30}
{527} {90 39} {103.30}
{528} {24 26 93} {254.74}
{529} {93 19 54} {255.74}
{530} {26 90 63} {154.74}
{531} {24 8 70} {55.74}
{532} {17 73 97} {54.74}
{533} {17 46 15} {55.74}
{534} {73 98 27} {54.74}
{535} {77} {52.00}
{536} {70 47} {3.30}
{537} {70 44} {3.30}
{538} {58} {151.00}
{539} {12} {51.00}
{540} {66 70 85} {155.74}
{541} {85 73 83} {154.74}
{542} {93 42} {103.30}
{543} {27 9} {103.30}
{544} {83 98} {104.30}
{545} {98 39} {104.30}
{546} {83 15} {4.30}
{547} {77} {52.00}
{548} {91 83} {3.30}
{549} {10} {52.00}
{550} {74 89 14} {155.74}
{551} {14 85 27} {154.74}
{552} {51 19 64 10 1 37} {111.59}
{553} {1 31 42 24 64 59} {112.59}
{554} {14 85} {203.30}
{555} {85 49} {103.30}
{556} {2 69} {202.30}
{557} {2 12} {202.30}
{558} {69 77} {103.30}
{559} {16 88 42 63 75 96} {212.59}
{560} {42 81 82 72 77 50} {212.59}
{561} {75 6 98 66 27 54} {113.59}
{562} {63 37 72 79 90 81} {112.59}
{563} {96 75 14 29 77 4} {112.59}
{564} {16 32 12 87 9 75} {13.59}
{565} {1} {51.00}
{566} {66} {52.00}
{567} {20 36 90 35} {108.28}
{568} {13 25} {203.30}
{569} {25 79} {203.30}
{570} {91 22} {302.30}
{571} {22 93} {202.30}
{572} {91 49} {103.30}
{573} {45 53 68 56 50} {158.90}
{574} {31 56 41} {154.74}
{575} {41 50 75} {155.74}
{576} {31 90 10} {154.74}
{577} {59 23} {4.30}
{578} {23 83} {4.30}
{579} {59 19} {4.30}
{580} {12 8 30} {254.74}
{581} {12 34 83} {354.74}
{582} {8 47 79} {254.74}
{583} {30 61 34} {55.74}
{584} {48 95} {103.30}
{585} {95 1} {102.30}
{586} {93} {51.00}
{587} {5 0} {103.30}
{588} {0 10} {103.30}
{589} {88} {51.00}
{590} {91 85} {302.30}
{591} {85 87} {203.30}
{592} {91 92} {103.30}
{593} {41 52} {4.30}
{594} {52 38} {4.30}
{595} {9 28 62} {255.74}

{596} {62 36 40} {155.74}
{597} {9 74 97} {55.74}
{598} {71 87 70 55} {109.28}
{599} {55 83 36 2} {8.28}
{600} {71 11 48 64} {8.28}
{601} {12 2 18} {54.74}
{602} {12 20 34} {54.74}
{603} {21 7} {302.30}
{604} {21 58} {202.30}
{605} {3 25 88} {453.74}
{606} {25 76 51} {354.74}
{607} {88 44 66} {154.74}
{608} {3 72 69} {154.74}
{609} {65} {51.00}
{610} {65 44} {102.30}
{611} {65 24} {103.30}
{612} {44 63} {3.30}
{613} {39 25 31 45} {306.28}
{614} {45 27 17 29} {306.28}
{615} {31 49 95 16} {207.28}
{616} {25 0 70 98} {107.28}
{617} {39 31 4} {154.74}
{618} {30 64 95} {55.74}
{619} {30 3 16} {55.74}
{620} {26} {151.00}
{621} {99} {51.00}
{622} {39 74 59 50} {307.28}
{623} {59 44 30 10} {308.28}
{624} {74 68 32 65} {307.28}
{625} {50 45 98 40} {206.28}
{626} {39 19 85 64} {108.28}
{627} {3 88 84 9 48 47} {310.59}
{628} {88 79 85 55 28 32} {311.59}
{629} {3 37 71 35 87 25} {311.59}
{630} {9 0 5 16 60 62} {112.59}
{631} {16 88 4 71} {207.28}
{632} {16 12 22 32} {107.28}
{633} {88 59 83 66} {8.28}
{634} {36} {52.00}
{635} {31 99} {302.30}
{636} {31 60} {302.30}
{637} {75 62 65 73} {207.28}
{638} {73 96 92 31} {306.28}
{639} {75 84 52 87} {208.28}
{640} {65 9 54 87} {108.28}
{641} {62 95 48 46} {8.28}
{642} {28} {151.00}
{643} {56 43 85 90} {106.28}
{644} {43 40 32 88} {107.28}
{645} {56 51 5 29} {107.28}
{646} {51 99} {202.30}
{647} {99 22} {202.30}
{648} {51 27} {202.30}
{649} {19} {52.00}
{650} {33 85} {102.30}
{651} {85 11} {102.30}
{652} {14 36 51 56 32 87 99 91} {416.13}
{653} {99 54 7 82 92 84 1 41} {615.13}
{654} {14 43 2 32 72 95 99 21} {416.13}
{655} {56 78 73 8 6 38 50 24} {316.13}
{656} {51 69 89 50 16 83 74 29} {316.13}
{657} {91 56 79 75 28 30 6 18} {217.13}
{658} {59 99 16 50} {207.28}
{659} {50 69 40 42} {306.28}
{660} {16 10 19 21} {208.28}
{661} {99 78 96 35} {207.28}
{662} {59 17 72 73} {207.28}
{663} {54} {52.00}
{664} {40 0} {302.30}
{665} {40 44} {202.30}
{666} {0 89} {103.30}
{667} {26} {151.00}
{668} {6} {52.00}
{669} {5 51 76} {55.74}
{670} {51 76 13} {55.74}
{671} {5 44 58} {54.74}
{672} {14 0 95} {154.74}
{673} {95 56 80} {154.74}
{674} {0 87 28} {54.74}
{675} {73} {151.00}
{676} {12} {151.00}
{677} {20 69} {103.30}
{678} {96} {151.00}
{679} {31} {51.00}
{680} {94 30 48 49} {8.28}
{681} {30 83 65 72} {8.28}
{682} {77 27} {103.30}
{683} {27 63} {3.30}
{684} {45 0} {202.30}
{685} {0 58} {302.30}
{686} {45 66} {3.30}
{687} {27} {151.00}
{688} {88 27} {102.30}
{689} {88 79} {3.30}
{690} {2} {51.00}
{691} {8} {151.00}
{692} {61 40} {103.30}
{693} {61 58} {103.30}
{694} {40 32} {3.30}
{695} {84} {51.00}
{696} {50 26} {202.30}
{697} {26 74} {302.30}
{698} {50 14} {103.30}
{699} {99 38 54} {255.74}
{700} {99 78 88} {254.74}
{701} {54 1 15} {155.74}
{702} {38 92 17} {55.74}
{703} {22} {51.00}

{704} {33 47 75} {254.74}
{705} {33 40 91} {353.74}
{706} {49} {52.00}
{707} {21} {51.00}
{708} {13 14 32} {56.74}
{709} {54} {52.00}
{710} {88 92} {203.30}
{711} {88 29} {203.30}
{712} {92 28} {103.30}
{713} {44 15} {3.30}
{714} {6 98 26} {155.74}
{715} {6 8 76} {155.74}
{716} {98 2 14} {55.74}
{717} {94} {151.00}
{718} {74} {51.00}
{719} {65} {51.00}
{720} {8 43 2 72 78 81} {12.59}
{721} {72 89 17 56 64 63} {12.59}
{722} {67 12} {203.30}
{723} {12 17} {202.30}
{724} {0 66 8 59} {7.28}
{725} {59 19 72 17} {8.28}
{726} {46} {52.00}
{727} {47} {51.00}
{728} {97 69} {203.30}
{729} {69 56} {302.30}
{730} {78 66 96} {55.74}
{731} {96 14 87} {55.74}
{732} {35} {52.00}
{733} {84} {51.00}
{734} {23 42} {104.30}
{735} {42 36} {104.30}
{736} {23 71} {4.30}
{737} {3 77} {103.30}
{738} {3 5} {103.30}
{739} {62} {52.00}
{740} {12 88 33 97 98} {558.90}
{741} {98 28 73 83 15} {359.90}
{742} {33 34 18 16 15} {359.90}
{743} {12 19 97 5 34} {359.90}
{744} {15 61} {104.30}
{745} {15 75} {4.30}
{746} {60} {151.00}
{747} {1 72 76} {155.74}
{748} {72 64 94} {155.74}
{749} {76 24 31} {155.74}
{750} {1 43 13} {55.74}
{751} {76 39} {4.30}
{752} {79 34} {203.30}
{753} {34 81} {103.30}
{754} {20 78 74} {55.74}
{755} {78 71 26} {55.74}
{756} {74 47 32} {54.74}
{757} {85 1 23 88} {306.28}
{758} {85 55 66 3} {207.28}
{759} {88 13 2 29} {107.28}
{760} {94 61} {3.30}
{761} {94 66} {3.30}
{762} {86 43 82} {254.74}
{763} {86 20 80} {255.74}
{764} {82 17 9} {154.74}
{765} {43 38 99} {55.74}
{766} {79} {52.00}
{767} {83} {52.00}
{768} {25 20} {103.30}
{769} {25 76} {3.30}
{770} {87} {52.00}
{771} {83} {52.00}
{772} {41} {52.00}
{773} {45} {151.00}
{774} {39 9 4 80} {208.28}
{775} {39 47 9 11} {207.28}
{776} {80 11 55 14} {108.28}
{777} {9 65 90 39} {107.28}
{778} {1} {51.00}
{779} {31} {151.00}
{780} {24 46 19} {56.74}
{781} {19 66 60} {55.74}
{782} {39} {52.00}
{783} {44 66 74} {154.74}
{784} {66 11 78} {155.74}
{785} {8} {51.00}
{786} {95 97 87 80} {208.28}
{787} {87 58 33 63} {207.28}
{788} {95 52 78 74} {207.28}
{789} {97 7 2 84} {206.28}
{790} {51} {51.00}
{791} {71 73} {203.30}
{792} {73 76} {203.30}
{793} {17} {151.00}
{794} {80} {52.00}
{795} {89 29} {4.30}
{796} {29 82} {3.30}
{797} {4 45} {202.30}
{798} {4 26} {302.30}
{799} {45 87} {103.30}
{800} {80 74 89} {55.74}
{801} {93 33 15} {354.74}
{802} {33 16 0} {354.74}
{803} {15 12 81} {155.74}
{804} {93 5 18} {55.74}
{805} {8 7 68 65 2 74 4 71} {514.13}
{806} {74 4 86 2 72 1 33 10} {714.13}
{807} {68 70 56 64 53 74 27 54} {517.13}
{808} {65 99 20 58 8 73 84 31} {513.13}
{809} {8 33 13 25 9 89 63 53} {217.13}
{810} {71 79 72 91 62 96 40 90} {216.13}
{811} {93 60 35} {254.74}

{812} {35 42 50} {155.74}
{813} {60 83 32} {55.74}
{814} {0} {151.00}
{815} {98 38 28} {155.74}
{816} {38 84 18} {155.74}
{817} {98 68 27} {155.74}
{818} {28 51 72} {154.74}
{819} {28} {51.00}
{820} {98 26} {103.30}
{821} {98 1} {103.30}
{822} {26 64} {3.30}
{823} {95} {51.00}
{824} {21} {51.00}
{825} {59 43 25} {55.74}
{826} {25 57 85} {54.74}
{827} {28} {51.00}
{828} {63 74} {103.30}
{829} {74 20} {103.30}
{830} {36 73} {103.30}
{831} {16} {52.00}
{832} {56 39 95} {354.74}
{833} {39 51 88} {254.74}
{834} {56 28 66} {154.74}
{835} {95 98 30} {55.74}
{836} {13 66} {104.30}
{837} {13 50} {103.30}
{838} {13 95} {3.30}
{839} {45 74} {202.30}
{840} {74 5} {103.30}
{841} {45 77} {3.30}
{842} {84} {51.00}
{843} {9 34 76} {255.74}
{844} {34 93 39} {354.74}
{845} {2} {51.00}
{846} {82} {151.00}
{847} {48} {52.00}
{848} {58} {51.00}
{849} {0 21} {202.30}
{850} {21 19} {103.30}
{851} {0 77} {103.30}
{852} {7} {151.00}
{853} {8} {151.00}
{854} {29} {52.00}
{855} {71 80} {4.30}
{856} {80 2} {3.30}
{857} {4} {151.00}
{858} {56} {51.00}
{859} {43} {52.00}
{860} {89 53 96 62 44 57 56} {364.33}
{861} {56 63 28 50 82 59 6} {463.33}
{862} {44 27 95 72 69 61 39} {463.33}
{863} {57 88 22 19 48 79 62} {365.33}
{864} {62 74 9 88 54 55 49} {265.33}
{865} {53 11 8 95 66 15 83} {264.33}
{866} {38 13 69 33} {207.28}
{867} {38 86 11 29} {207.28}
{868} {69 55 43 19} {108.28}
{869} {13 23 85 46} {108.28}
{870} {33 0 66 97} {107.28}
{871} {31 1} {202.30}
{872} {31 35} {203.30}
{873} {1 39} {103.30}
{874} {34 39 45} {254.74}
{875} {39 34 11} {254.74}
{876} {34 85 73} {253.74}
{877} {45 89 54} {55.74}
{878} {74} {151.00}
{879} {25} {151.00}
{880} {97} {52.00}
{881} {12} {51.00}
{882} {27} {51.00}
{883} {99} {151.00}
{884} {68} {52.00}
{885} {25 95 79 44 40} {457.90}
{886} {40 11 68 64 4} {458.90}
{887} {95 20 48 1 49} {359.90}
{888} {25 27 14 55 63} {259.90}
{889} {1} {51.00}
{890} {92} {52.00}
{891} {17} {151.00}
{892} {20} {52.00}
{893} {93 40} {302.30}
{894} {40 32} {103.30}
{895} {93 97} {103.30}
{896} {50 31 96 43} {306.28}
{897} {50 96 86 99} {405.28}
{898} {96 73 86 99} {305.28}
{899} {31 22 62 13} {107.28}
{900} {73 35 47} {154.74}
{901} {47 28 39} {154.74}
{902} {35 47 96} {154.74}
{903} {50 45} {102.30}
{904} {50 63} {103.30}
{905} {45 33} {102.30}
{906} {96 1} {202.30}
{907} {1 38} {203.30}
{908} {96 74} {102.30}
{909} {99 52} {103.30}
{910} {47} {51.00}
{911} {80} {52.00}
{912} {71} {52.00}
{913} {72 29 67 42 68} {61.90}
{914} {42 33 49 45 23} {59.90}
{915} {98 3} {103.30}
{916} {4} {151.00}
{917} {2 1} {202.30}
{918} {1 62} {203.30}
{919} {2 36} {3.30}

{920} {36 96} {103.30}
{921} {96 41} {103.30}
{922} {16} {52.00}
{923} {11} {151.00}
{924} {74 21} {102.30}
{925} {74 61} {103.30}
{926} {2 89 96 31} {306.28}
{927} {31 57 67 1} {107.28}
{928} {96 7 29 36} {107.28}
{929} {2 29 49 22} {107.28}
{930} {73 99} {202.30}
{931} {99 23} {103.30}
{932} {73 92} {3.30}
{933} {84 41} {3.30}
{934} {84 29} {3.30}
{935} {77} {52.00}
{936} {35} {52.00}
{937} {13} {52.00}
{938} {45 10 76} {55.74}
{939} {73} {151.00}
{940} {24} {52.00}
{941} {36 60 78} {55.74}
{942} {60 26 75} {54.74}
{943} {9} {52.00}
{944} {17} {151.00}
{945} {56} {151.00}
{946} {1} {151.00}
{947} {5 99 83} {155.74}
{948} {63 14} {4.30}
{949} {14 61} {4.30}
{950} {63 35} {4.30}
{951} {74 97} {103.30}
{952} {74 18} {3.30}
{953} {29 15} {104.30}
{954} {21} {51.00}
{955} {40} {151.00}
{956} {51 36} {103.30}
{957} {98 44 37 85 48 25 46} {363.33}
{958} {44 37 73 93 41 59 83} {463.33}
{959} {25 86 3 52 15 30 4} {463.33}
{960} {48 92 56 33 7 51 18} {463.33}
{961} {98 96 6 75 82 47 31} {463.33}
{962} {85 3 7 18 51 15 46} {363.33}
{963} {32} {52.00}
{964} {66} {52.00}
{965} {68} {52.00}
{966} {46 31} {103.30}
{967} {64 27} {103.30}
{968} {35 94} {103.30}
{969} {94 78} {3.30}
{970} {45} {151.00}
{971} {34 90} {202.30}
{972} {90 99} {202.30}
{973} {34 83} {103.30}
{974} {74} {151.00}
{975} {40 51 43} {154.74}
{976} {43 84 33} {154.74}
{977} {77} {52.00}
{978} {73} {151.00}
{979} {27} {151.00}
{980} {59 7} {203.30}
{981} {7 44} {302.30}
{982} {2 68 33 16 71 40 64} {164.33}
{983} {40 37 76 18 10 71 5} {165.33}
{984} {80 89 76 69 31} {59.90}
{985} {80 32 58 8 34} {58.90}
{986} {90} {51.00}
{987} {8} {51.00}
{988} {2 40} {302.30}
{989} {40 41} {203.30}
{990} {2 35} {203.30}
{991} {61 67} {4.30}
{992} {82 51 16} {154.74}
{993} {16 23 96} {155.74}
{994} {82 90 38} {54.74}
{995} {58} {51.00}
{996} {65} {51.00}
{997} {3} {151.00}
{998} {53 30 67} {56.74}
{999} {30 66 12} {55.74}{941} {8} {151.00}

Output File(1): winners, total revenue, total running time when analysis bids=0

```
C:\CADIA>cadia items.txt bids.txt 100 1000 0

=====Lower Bound Result ==========================================
BidderID    Bid Price       ItemList
    3         52.00         32
    9         52.00         87
   26         52.00         29
   29        202.30         27  96
  122        306.28         8  49  82  84
  164        104.30         18  43
  203        151.00         65
  215        104.30         55  75
  218         52.00         97
  227        463.33         5  11  17  50  63  68  73
  292        359.90         7  35  46  67  90
  306        151.00         51
  309        457.90         0  37  70  88  94
  334         52.00         64
  361         52.00         78
  370        708.59         22  28  45  56  60  85
  380        156.74         14  20  92
  395        104.30         16  81
  409         52.00         57
  425        151.00         47
  427        616.13         19  21  31  36  52  69  77  91
  450         52.00         41
  479         52.00         71
  528        254.74         24  26  93
  538        151.00         58
  545        104.30         39  98
  581        354.74         12  34  83
  668         52.00         6
  699        255.74         38  54  99
  734        104.30         23  42
  739         52.00         62
  744        104.30         15  61
  794         52.00         80
  806        714.13         1  2  4  10  33  72  74  86
  836        104.30         13  66
  847         52.00         48
  885        457.90         25  40  44  79  95
  943         52.00         9
  997        151.00         3

Total Revenue = 7521.52
0.010000 seconds

=====CADIA Result ================================================
BidderID    Bid Price        ItemList
    3         52.00         32
   11        151.00         27
   40        151.00         74
```

```
 75        151.00          28
 79        52.00           10
106        359.90          6  22  54  60  97
132        155.74          30  52  91
135        151.00          21
163        104.30          5  18
171        151.00          37
198        302.30          8  69
216        104.30          55  78
221        151.00          4
222        354.74          1  11  53
237        52.00           38
288        203.30          0  72
292        359.90          7  35  46  67  90
301        203.30          64  99
306        151.00          51
319        307.28          24  44  82  83
326        104.30          70  75
335        306.28          56  61  86  88
363        311.59          26  45  59  80  89  94
380        156.74          14  20  92
382        151.00          12
396        104.30          79  81
409        52.00           57
411        151.00          17
430        416.13          19  25  47  62  65  71  84  98
462        151.00          2
476        155.74          16  77  93
555        103.30          49  85
584        103.30          48  95
634        52.00           36
675        151.00          73
734        104.30          23  42
782        52.00           39
787        207.28          33  58  63  87
836        104.30          13  66
843        255.74          9  34  76
884        52.00           68
896        306.28          31  43  50  96
953        104.30          15  29
989        203.30          40  41
997        151.00          3

Total Revenue = 7678.24
42.572000 seconds

maxRound is 0 and maxRevenue is 7678.24
Total time required: 42.572000 seconds
==========================================================================
```

Output File(2): winners, total revenue, total running time when analysis bids=2

```
C:\CADIA>cadia items.txt bids.txt 100 1000 2

=====Lower Bound Result ==========================================
BidderID    Bid Price       ItemList
    3        52.00          32
    9        52.00          87
   26        52.00          29
   29        202.30         27  96
  122        306.28         8  49  82  84
  164        104.30         18  43
  203        151.00         65
  215        104.30         55  75
  218        52.00          97
  227        463.33         5  11  17  50  63  68  73
  292        359.90         7  35  46  67  90
  306        151.00         51
  309        457.90         0  37  70  88  94
  334        52.00          64
  361        52.00          78
  370        708.59         22  28  45  56  60  85
  380        156.74         14  20  92
  395        104.30         16  81
  409        52.00          57
  425        151.00         47
  427        616.13         19  21  31  36  52  69  77  91
  450        52.00          41
  479        52.00          71
  528        254.74         24  26  93
  538        151.00         58
  545        104.30         39  98
  581        354.74         12  34  83
  668        52.00          6
  699        255.74         38  54  99
  734        104.30         23  42
  739        52.00          62
  744        104.30         15  61
  794        52.00          80
  806        714.13         1  2  4  10  33  72  74  86
  836        104.30         13  66
  847        52.00          48
  885        457.90         25  40  44  79  95
  943        52.00          9
  997        151.00         3

Total Revenue = 7521.52
0.010000 seconds

=====CADIA Result ================================================
BidderID    Bid Price       ItemList
    3        52.00          32
   11        151.00         27
```

```
  40        151.00         74
  75        151.00         28
  79        52.00          10
 106        359.90         6  22  54  60  97
 132        155.74         30  52  91
 135        151.00         21
 163        104.30         5  18
 171        151.00         37
 198        302.30         8  69
 216        104.30         55  78
 221        151.00         4
 222        354.74         1  11  53
 237        52.00          38
 288        203.30         0  72
 292        359.90         7  35  46  67  90
 301        203.30         64  99
 306        151.00         51
 319        307.28         24  44  82  83
 326        104.30         70  75
 335        306.28         56  61  86  88
 363        311.59         26  45  59  80  89  94
 380        156.74         14  20  92
 382        151.00         12
 396        104.30         79  81
 409        52.00          57
 411        151.00         17
 430        416.13         19  25  47  62  65  71  84  98
 462        151.00         2
 476        155.74         16  77  93
 555        103.30         49  85
 584        103.30         48  95
 634        52.00          36
 675        151.00         73
 734        104.30         23  42
 782        52.00          39
 787        207.28         33  58  63  87
 836        104.30         13  66
 843        255.74         9  34  76
 884        52.00          68
 896        306.28         31  43  50  96
 953        104.30         15  29
 989        203.30         40  41
 997        151.00         3

Total Revenue = 7678.24
44.044000 seconds

================================================================
Use Winners & Losers Information to improve result
=====Round #1 ==================================================
Delete: 734 Delete: 930
BidderID    Bid Price        ItemList
   63        151.00         11
   75        151.00         28
```

```
  87        52.00          35
  88        52.00          14
 106        359.90         6  22  54  60  97
 132        155.74         30  52  91
 163        104.30         5  18
 176        151.00         99
 234        203.30         25  64
 237        52.00          38
 261        302.30         21  56
 262        203.30         9  50
 306        151.00         51
 308        359.90         17  19  49  53  88
 325        104.30         24  75
 358        255.74         10  27  29
 387        208.28         39  57  69  72
 396        104.30         79  81
 418        354.74         8  37  78
 425        151.00         47
 461        151.00         94
 476        155.74         16  77  93
 544        104.30         83  98
 566        52.00          66
 584        103.30         48  95
 598        109.28         55  70  71  87
 612        3.30           44  63
 624        307.28         32  65  68  74
 685        302.30         0  58
 722        203.30         12  67
 735        104.30         36  42
 744        104.30         15  61
 763        255.74         20  80  86
 773        151.00         45
 792        203.30         73  76
 798        302.30         4  26
 846        151.00         82
 869        108.28         13  23  46  85
 890        52.00          92
 918        203.30         1  62
 926        306.28         2  31  89  96
 971        202.30         34  90
 976        154.74         33  43  84
 980        203.30         7  59
 989        203.30         40  41
 997        151.00         3

Total Revenue = 7975.34
48.930000 seconds

=====Round #2 ==================================================
Delete: 734
BidderID    Bid Price        ItemList
   63        151.00         11
   75        151.00         28
   87        52.00          35
```

```
  88        52.00          14
 106        359.90         6  22  54  60  97
 132        155.74         30  52  91
 163        104.30         5  18
 176        151.00         99
 234        203.30         25  64
 237        52.00          38
 261        302.30         21  56
 262        203.30         9  50
 306        151.00         51
 308        359.90         17  19  49  53  88
 325        104.30         24  75
 358        255.74         10  27  29
 387        208.28         39  57  69  72
 396        104.30         79  81
 418        354.74         8  37  78
 425        151.00         47
 461        151.00         94
 476        155.74         16  77  93
 544        104.30         83  98
 566        52.00          66
 584        103.30         48  95
 598        109.28         55  70  71  87
 612        3.30           44  63
 624        307.28         32  65  68  74
 685        302.30         0  58
 722        203.30         12  67
 735        104.30         36  42
 744        104.30         15  61
 763        255.74         20  80  86
 773        151.00         45
 792        203.30         73  76
 798        302.30         4  26
 846        151.00         82
 869        108.28         13  23  46  85
 890        52.00          92
 918        203.30         1  62
 926        306.28         2  31  89  96
 971        202.30         34  90
 976        154.74         33  43  84
 980        203.30         7  59
 989        203.30         40  41
 997        151.00         3

Total Revenue = 7975.34
49.180000 seconds


=====Round #3 ==================================================
Delete: 930
BidderID    Bid Price        ItemList
   59        203.30         33  62
   63        151.00         11
   75        151.00         28
```

```
  86        52.00          49
  88        52.00          14
  96        52.00          72
  99        203.30         1  59
 101        52.00          70
 129        103.30         36  84
 132        155.74         30  52  91
 164        104.30         18  43
 171        151.00         37
 216        104.30         55  78
 235        203.30         64  69
 261        302.30         21  56
 262        203.30         9  50
 292        359.90         7  35  46  67  90
 325        104.30         24  75
 358        255.74         10  27  29
 368        151.00         60
 396        104.30         79  81
 409        52.00          57
 411        151.00         17
 425        151.00         47
 432        353.74         4  8  22
 461        151.00         94
 476        155.74         16  77  93
 519        151.00         26
 544        104.30         83  98
 584        103.30         48  95
 591        203.30         85  87
 606        354.74         25  51  76
 612        3.30           44  63
 624        307.28         32  65  68  74
 668        52.00          6
 685        302.30         0  58
 699        255.74         38  54  99
 710        203.30         88  92
 734        104.30         23  42
 743        359.90         5  12  19  34  97
 744        104.30         15  61
 763        255.74         20  80  86
 773        151.00         45
 782        52.00          39
 791        203.30         71  73
 836        104.30         13  66
 846        151.00         82
 926        306.28         2  31  89  96
 989        203.30         40  41
 997        151.00         3

Total Revenue = 8420.84
48.089000 seconds

maxRound is 3 and maxRevenue is 8420.84
Total time required: 190.253000 seconds
================================================================
```

APPENDIX B

Source Code for Brute Force Technique

This appendix provides the source code of the brute force technique (BFT) program used in the evaluation. The program can handle up to 10 bids and 10 items.

```
/*-------------------------------------------------------------------
Author:         Andy Law
Date:           2003
Description:
The program uses brute force technique (BFT) to identify winners in a
combinatorial auction.
Program Execution Format:
bft.exe items_file.txt bids_file.txt
-------------------------------------------------------------------
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
//-------------------------------------------------------------------
#define MAXLEN                  80
#define MAXITEM                 10
#define MAXBIDDER               10
#define TRUE                    1
#define FALSE                   0
#define DELETED                 -1
#define NO_ITEM                 -2
//-------------------------------------------------------------------
typedef struct itemMatrix       //item information ADT
{
        double itemPrice;
} ITEMMATRIX;

typedef struct bidMatrix        //bid information ADT
{
        int bidderID;
        int itemList[MAXITEM];
```

```
        double bidPrice;
        int sold;               //1 if sold, -1 if deleted
} BIDMATRIX;

//-----------------------------------------------------------------
BIDMATRIX bMatrix[MAXBIDDER]; //bid information
ITEMMATRIX iMatrix[MAXITEM];  //item information
int bidderListIdx=0;          //total number of bids
int processedBidder=0;        //number of bids
int winner[MAXBIDDER];
double tempRev=0.0;
//-----------------------------------------------------------------
void readInput(char *argv[]);
void processItemInput(char *buf, int *itemNo, double *itemValue);
void processBidInput(char *buf);
void initialize();
void processAuction();
int auctionCompete(int max);
void displayResult();
int conflict2(int a1,int a2);
int conflict3(int a1,int a2, int a3);
int conflict4(int a1,int a2, int a3, int a4);
int conflict5(int a1,int a2, int a3, int a4, int a5);
int conflict6(int a1,int a2, int a3, int a4, int a5,int a6);
int conflict7(int a1,int a2, int a3, int a4, int a5,int a6,int a7);
int conflict8(int a1,int a2, int a3, int a4, int a5,int a6,int a7,int a8);
int conflict9(int a1,int a2, int a3, int a4, int a5,int a6,int a7,int a8,int a9);
int conflict10(int a1,int a2, int a3, int a4, int a5,int a6,int a7,int a8,int a9,int
a10);
//-----------------------------------------------------------------
//main() module.
//-----------------------------------------------------------------
int main (int argc, char* argv[]) {
        clock_t start, finish;
        double duration;

        if (argc !=3)
        {
                printf("Usage: BFT <items_file> <bids_file>\n");
                exit (1);
        }
        start = clock();
        printf( "==========================\n");
        printf( "   Best Search - Beginning\n");
        printf( "==========================\n");
        initialize();
        readInput(argv);
        processAuction();
        displayResult();
        finish = clock();
```

```
        printf( "=========================\n");
        printf( "    Best Search - Ending\n");
        printf( "=========================\n");
        duration = (double)(finish - start) / CLOCKS_PER_SEC;
        printf( "%.61f seconds\n", duration );
        return 0;
}
//---------------------------------------------------------------
//Initializes the matrix for storing the bids information.
//---------------------------------------------------------------
void initialize(){
        int i,j;

        for (i=0;i<MAXBIDDER;i++)
                winner[i]=-1;
        for (i=0;i<MAXBIDDER;i++){
                bMatrix[i].bidderID=-1;
                bMatrix[i].bidPrice=-1.0;
                bMatrix[i].sold=FALSE;
                for (j=0;j<MAXITEM;j++)
                        bMatrix[i].itemList[j]=-2;
        }
}
//---------------------------------------------------------------
//Displays the auction result to the command screen.
//---------------------------------------------------------------
void displayResult(){
        int i,j;
        double rev=0.0;

        printf("BidderID    Bid Price        ItemList\n");
        for (i=0;i<bidderListIdx;i++){
                if (winner[i]== TRUE) {
                        printf("%5i        ",bMatrix[i].bidderID);
                        printf("%-15.2f",bMatrix[i].bidPrice);
                        rev=rev+bMatrix[i].bidPrice;
                        for (j=0;j<MAXITEM;j++){
                                if (bMatrix[i].itemList[j]==1)
                                        printf("%i  ",j);
                        }
                        printf("\n");
                }
        }
        printf("\nTotal Revenue = %.2f\n",rev);
}
//---------------------------------------------------------------
//Reset the winner list.
//---------------------------------------------------------------
void reset()
{
```

```
        int i;

        for (i=0;i<bidderListIdx;i++)
                winner[i]=FALSE;
}
//-----------------------------------------------------------------
//Auction is processed here using brute force technique.
//-----------------------------------------------------------------
void processAuction(){
        int a1,a2,a3,a4,a5,a6,a7,a8,a9;
        int a10;
        int n=0;
        int ret=FALSE;
        double maxRev=0.0;

        for (a1=0;a1<bidderListIdx;a1++){
                if (maxRev<bMatrix[a1].bidPrice){
                        maxRev=bMatrix[a1].bidPrice;
                        reset();
                        winner[a1]=TRUE;
                }
                for (a2=0;a2<bidderListIdx;a2++){
                        if (a1 != a2){
                                ret=conflict2(a1,a2);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice;
                                                reset();
                                                winner[a1]=TRUE;
                                                winner[a2]=TRUE;
                                        }
                                }
                        }
                for (a3=0;a3<bidderListIdx;a3++){
                        if (a1!=a2 && a1!= a3 && a2!=a3){
                                ret=conflict3(a1,a2,a3);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice;
                                                reset();
                                                winner[a1]=TRUE;
                                                winner[a2]=TRUE;
                                                winner[a3]=TRUE;
                                        }
                                }
                        }
                for (a4=0;a4<bidderListIdx;a4++){
                        if (a1!=a2 && a1!= a3 && a1!=a4 &&
```

```
                                a2!=a3 && a2!=a4 &&
                                a3!=a4){
                                ret=conflict4(a1,a2,a3,a4);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice;
                                                reset();
                                                winner[a1]=TRUE;
                                                winner[a2]=TRUE;
                                                winner[a3]=TRUE;
                                                winner[a4]=TRUE;
                                        }
                                }
                        }
                for (a5=0;a5<bidderListIdx;a5++){
                        if (a1!=a2 && a1!= a3 && a1!=a4 && a1!=a5 &&
                                a2!=a3 && a2!=a4 && a2!=a5 &&
                                a3!=a4 && a3!=a5 &&
                                a4!=a5){
                                ret=conflict5(a1,a2,a3,a4,a5);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce+bMatrix[a5].bidPrice){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice+bMatrix[a5].bidPrice;
                                                reset();
                                                winner[a1]=TRUE;
                                                winner[a2]=TRUE;
                                                winner[a3]=TRUE;
                                                winner[a4]=TRUE;
                                                winner[a5]=TRUE;
                                        }
                                }
                        }
                for (a6=0;a6<bidderListIdx;a6++){
                        if (a1!=a2 && a1!= a3 && a1!=a4 && a1!=a5 && a1!=a6 &&
                                a2!=a3 && a2!=a4 && a2!=a5 && a2!=a6 &&
                                a3!=a4 && a3!=a5 && a3!=a6 &&
                                a4!=a5 && a4!=a6 &&
                                a5!=a6){
                                ret=conflict6(a1,a2,a3,a4,a5,a6);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce+bMatrix[a5].bidPrice+
                                                                bMatrix[a6].bidPrice){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice+bMatrix[a5].bidPrice+
```

```
                                        bMatrix[a6].bidPrice;
                                    reset();
                                    winner[a1]=TRUE;
                                    winner[a2]=TRUE;
                                    winner[a3]=TRUE;
                                    winner[a4]=TRUE;
                                    winner[a5]=TRUE;
                                    winner[a6]=TRUE;
                                }
                            }
                        }
                for (a7=0;a7<bidderListIdx;a7++){
                        if (a1!=a2 && a1!= a3 && a1!=a4 && a1!=a5 && a1!=a6 && a1!=a7 &&
                                a2!=a3 && a2!=a4 && a2!=a5 && a2!=a6 && a2!=a7 &&
                                a3!=a4 && a3!=a5 && a3!=a6 && a3!=a7 &&
                                a4!=a5 && a4!=a6 && a4!=a7 &&
                                a5!=a6 && a5!=a7 &&
                                a6!=a7) {
                                ret=conflict7(a1,a2,a3,a4,a5,a6,a7);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce+bMatrix[a5].bidPrice

        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice+bMatrix[a5].bidPrice

        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice;
                                    reset();
                                    winner[a1]=TRUE;
                                    winner[a2]=TRUE;
                                    winner[a3]=TRUE;
                                    winner[a4]=TRUE;
                                    winner[a5]=TRUE;
                                    winner[a6]=TRUE;
                                    winner[a7]=TRUE;
                                }
                            }
                        }
                for (a8=0;a8<bidderListIdx;a8++){
                        if (a1!=a2 && a1!= a3 && a1!=a4 && a1!=a5 && a1!=a6 && a1!=a7 &&
a1!=a8 &&
                                a2!=a3 && a2!=a4 && a2!=a5 && a2!=a6 && a2!=a7 && a2!=a8 &&
                                a3!=a4 && a3!=a5 && a3!=a6 && a2!=a7 && a2!=a8 &&
                                a4!=a5 && a4!=a6 && a4!=a7 && a4!=a8 &&
                                a5!=a6 && a5!=a7 && a5!=a8 &&
                                a6!=a6 && a6!=a8 &&
                                a7!=a8) {
                                ret=conflict8(a1,a2,a3,a4,a5,a6,a7,a8);
                                if (ret==FALSE){
```

```
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce+bMatrix[a5].bidPrice

        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice+bMatrix[a8].bidPrice){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice+bMatrix[a5].bidPrice

        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice+bMatrix[a8].bidPrice;
                                                reset();
                                                winner[a1]=TRUE;
                                                winner[a2]=TRUE;
                                                winner[a3]=TRUE;
                                                winner[a4]=TRUE;
                                                winner[a5]=TRUE;
                                                winner[a6]=TRUE;
                                                winner[a7]=TRUE;
                                                winner[a8]=TRUE;
                                        }
                                }
                        }
                for (a9=0;a9<bidderListIdx;a9++){
                        if (a1!=a2 && a1!= a3 && a1!=a4 && a1!=a5 && a1!=a6 && a1!=a7 &&
a1!=a8 && a1!=a9 &&
                                a2!=a3 && a2!=a4 && a2!=a5 && a2!=a6 && a2!=a7 && a2!=a8 &&
a2!=a9 &&
                                a3!=a4 && a3!=a5 && a3!=a6 && a2!=a7 && a2!=a8 && a2!=a9 &&
                                a4!=a5 && a4!=a6 && a4!=a7 && a4!=a8 && a4!=a9 &&
                                a5!=a6 && a5!=a7 && a5!=a8 && a5!=a9 &&
                                a6!=a6 && a6!=a8 && a6!=a9 &&
                                a7!=a8 && a7!=a9 &&
                                a8!=a9) {
                                ret=conflict9(a1,a2,a3,a4,a5,a6,a7,a8,a9);
                                if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce+bMatrix[a5].bidPrice

        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice+bMatrix[a8].bidPrice+bMatrix[a9].bidPri
ce){

        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice+bMatrix[a5].bidPrice

        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice+bMatrix[a8].bidPrice+bMatrix[a9].bidPri
ce;
                                                reset();
                                                winner[a1]=TRUE;
                                                winner[a2]=TRUE;
                                                winner[a3]=TRUE;
                                                winner[a4]=TRUE;
                                                winner[a5]=TRUE;
                                                winner[a6]=TRUE;
                                                winner[a7]=TRUE;
                                                winner[a8]=TRUE;
                                                winner[a9]=TRUE;
```

```
                                        }
                                    }
                                }
                        for (a10=0;a10<bidderListIdx;a10++){
                                if (a1!=a2 && a1!= a3 && a1!=a4 && a1!=a5 && a1!=a6 && a1!=a7 &&
a1!=a8 && a1!=a9 && a1!=a10 &&
                                    a2!=a3 && a2!=a4 && a2!=a5 && a2!=a6 && a2!=a7 && a2!=a8 &&
a2!=a9 && a2!=a10 &&
                                    a3!=a4 && a3!=a5 && a3!=a6 && a2!=a7 && a2!=a8 && a2!=a9 &&
a2!=a10 &&
                                    a4!=a5 && a4!=a6 && a4!=a7 && a4!=a8 && a4!=a9 && a4!=a10
&&
                                    a5!=a6 && a5!=a7 && a5!=a8 && a5!=a9 && a5!=a10 &&
                                    a6!=a6 && a6!=a8 && a6!=a9 && a6!=a10 &&
                                    a7!=a8 && a7!=a9 && a7!=a10 &&
                                    a8!=a9 && a8!=a10 &&
                                    a9!=a10) {
                                    ret=conflict10(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10);
                                    if (ret==FALSE){
                                        if
(maxRev<bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].bidPri
ce+bMatrix[a5].bidPrice
        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice+bMatrix[a8].bidPrice+bMatrix[a9].bidPri
ce+bMatrix[a10].bidPrice){
        maxRev=bMatrix[a1].bidPrice+bMatrix[a2].bidPrice+bMatrix[a3].bidPrice+bMatrix[a4].
bidPrice+bMatrix[a5].bidPrice
        +bMatrix[a6].bidPrice+bMatrix[a7].bidPrice+bMatrix[a8].bidPrice+bMatrix[a9].bidPri
ce+bMatrix[a10].bidPrice;
                                            winner[a1]=TRUE;
                                            winner[a2]=TRUE;
                                            winner[a3]=TRUE;
                                            winner[a4]=TRUE;
                                            winner[a5]=TRUE;
                                            winner[a6]=TRUE;
                                            winner[a7]=TRUE;
                                            winner[a8]=TRUE;
                                            winner[a9]=TRUE;
                                            winner[a10]=TRUE;
                                        }
                                    }
                                }
                        }//a10
                        }//a9
                        }//a8
                        }//a7
                        }//a6
                        }//a5
                        }//a4
                        }//a3
                        }//a2
                }//a1
                printf("}\n");
```

```
}
//-------------------------------------------------------------------
//conflict2()
//-------------------------------------------------------------------
int conflict2(int a1, int a2){
	int i;

	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	return FALSE;
}
//-------------------------------------------------------------------
//conflict3()
//-------------------------------------------------------------------
int conflict3(int a1, int a2, int a3){
	int i;

	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	return FALSE;
}
//-------------------------------------------------------------------
//conflict4()
//-------------------------------------------------------------------
int conflict4(int a1, int a2, int a3, int a4){
	int i;

	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
			return TRUE; //there is a conflict
```

```
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        return FALSE;
}
//-----------------------------------------------------------------
//conflict5()
//-----------------------------------------------------------------
int conflict5(int a1, int a2, int a3, int a4, int a5){
        int i;

        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a5].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a2].itemList[i]+bMatrix[a5].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
```

```
            if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a3].itemList[i]+bMatrix[a5].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a4].itemList[i]+bMatrix[a5].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        return FALSE;
}
//-----------------------------------------------------------------
//conflict6()
//-----------------------------------------------------------------
int conflict6(int a1, int a2, int a3, int a4, int a5, int a6){
        int i;

        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a1].itemList[i]+bMatrix[a5].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a1].itemList[i]+bMatrix[a6].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a2].itemList[i]+bMatrix[a5].itemList[i])==2)
                    return TRUE; //there is a conflict
```

```
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a2].itemList[i]+bMatrix[a6].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a3].itemList[i]+bMatrix[a5].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a3].itemList[i]+bMatrix[a6].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a4].itemList[i]+bMatrix[a5].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a4].itemList[i]+bMatrix[a6].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a5].itemList[i]+bMatrix[a6].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	return FALSE;
}
//-----------------------------------------------------------------
//conflict7()
//-----------------------------------------------------------------
int conflict7(int a1,int a2,int a3,int a4,int a5,int a6,int a7){
	int i;

	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
		if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
			return TRUE; //there is a conflict
	}
	for (i=0;i<MAXITEM;i++){
```

```
        if ((bMatrix[a1].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
```

```
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a4].itemList[i]+bMatrix[a6].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a4].itemList[i]+bMatrix[a7].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a5].itemList[i]+bMatrix[a6].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a5].itemList[i]+bMatrix[a7].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a6].itemList[i]+bMatrix[a7].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        return FALSE;
}
//-----------------------------------------------------------------
//conflict8()
//-----------------------------------------------------------------
int conflict8(int a1,int a2,int a3,int a4,int a5,int a6,int a7,int a8){
        int i;

        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a5].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a1].itemList[i]+bMatrix[a6].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
```

```
        if ((bMatrix[a1].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
```

```
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a4].itemList[i]+bMatrix[a5].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a4].itemList[i]+bMatrix[a6].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a4].itemList[i]+bMatrix[a7].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a4].itemList[i]+bMatrix[a8].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a5].itemList[i]+bMatrix[a6].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a5].itemList[i]+bMatrix[a7].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a5].itemList[i]+bMatrix[a8].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a6].itemList[i]+bMatrix[a7].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a6].itemList[i]+bMatrix[a8].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a7].itemList[i]+bMatrix[a8].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        return FALSE;
}
//-----------------------------------------------------------------
//conflict9()
//-----------------------------------------------------------------
int conflict9(int a1,int a2,int a3,int a4,int a5,int a6,int a7,int a8,int a9){
        int i;

        for (i=0;i<MAXITEM;i++){
```

```
        if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
```

```
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
```

```
            if ((bMatrix[a4].itemList[i]+bMatrix[a9].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a5].itemList[i]+bMatrix[a6].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a5].itemList[i]+bMatrix[a7].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a5].itemList[i]+bMatrix[a8].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a5].itemList[i]+bMatrix[a9].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a6].itemList[i]+bMatrix[a7].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a6].itemList[i]+bMatrix[a8].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a6].itemList[i]+bMatrix[a9].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a7].itemList[i]+bMatrix[a8].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a7].itemList[i]+bMatrix[a9].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
            if ((bMatrix[a8].itemList[i]+bMatrix[a9].itemList[i])==2)
                    return TRUE; //there is a conflict
        }
        return FALSE;
}
//-------------------------------------------------------------------
//conflict10()
//-------------------------------------------------------------------
int conflict10(int a1,int a2,int a3,int a4,int a5,int a6,int a7,int a8,int a9,int a10){
        int i;
```

```
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a2].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a3].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a1].itemList[i]+bMatrix[a10].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a3].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
```

```
        if ((bMatrix[a2].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a2].itemList[i]+bMatrix[a10].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a4].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a3].itemList[i]+bMatrix[a10].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a5].itemList[i])==2)
                return TRUE; //there is a conflict
```

```
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a4].itemList[i]+bMatrix[a10].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a5].itemList[i]+bMatrix[a6].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a5].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a5].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a5].itemList[i]+bMatrix[a9].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a5].itemList[i]+bMatrix[a10].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a6].itemList[i]+bMatrix[a7].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
        if ((bMatrix[a6].itemList[i]+bMatrix[a8].itemList[i])==2)
                return TRUE; //there is a conflict
}
for (i=0;i<MAXITEM;i++){
```

```
                if ((bMatrix[a6].itemList[i]+bMatrix[a9].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a6].itemList[i]+bMatrix[a10].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a7].itemList[i]+bMatrix[a8].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a7].itemList[i]+bMatrix[a9].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a8].itemList[i]+bMatrix[a9].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a8].itemList[i]+bMatrix[a10].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        for (i=0;i<MAXITEM;i++){
                if ((bMatrix[a9].itemList[i]+bMatrix[a10].itemList[i])==2)
                        return TRUE; //there is a conflict
        }
        return FALSE;
}
//-----------------------------------------------------------------
//Once the bid with highest bidding price or highest profit is found,
//all conflict bids will be deleted.
//-----------------------------------------------------------------
int auctionCompete(int max){
        int m,n;

        if (bMatrix[max].sold==DELETED)
                return 0;
        if (bMatrix[max].sold==FALSE){
                bMatrix[max].sold=TRUE;
                processedBidder++;
        }
        for (m=0;m<bidderListIdx;m++){
                if (bMatrix[m].sold==FALSE){ //
                        for (n=0;n<MAXITEM;n++){
                                if ((bMatrix[max].itemList[n]+bMatrix[m].itemList[n])==2){
                                        bMatrix[m].sold=DELETED;
                                        processedBidder++;
                                        break;
                                }
```

```
                        }
                }
        }
        return 0;
}
//-------------------------------------------------------------------
//Reads input data into the internal memory structures.
//-------------------------------------------------------------------
void readInput(char *argv[]){
        char buf[81];
        int temp1=0;
        double temp2=0.0;

        FILE *fp1, *fp2;
        if ((fp1 = fopen(argv[1], "r"))==NULL){
                printf("Cannot open <item file>\n");
                exit (1);
        }
        if ((fp2 = fopen(argv[2], "r"))==NULL){
                printf("Cannot open <bid file>\n");
                exit (1);
        }
        while (TRUE){
                if (fgets(buf, MAXLEN, fp1)==NULL)
                        break;
                processItemInput(buf, &temp1, &temp2);
                iMatrix[temp1].itemPrice=temp2;
        }
        while (TRUE){
                if (fgets(buf, MAXLEN, fp2)==NULL)
                        break;
                processBidInput(buf);
        }
}
//-------------------------------------------------------------------
//Reads items_file into iMatrix.
//-------------------------------------------------------------------
void processItemInput(char *buf, int *itemNo, double *itemValue){
        int i=0,j=0, process=FALSE;
        char temp[MAXLEN];

        while (TRUE){
                if (process==TRUE || i>MAXLEN)
                        break;
                else if (buf[i++]=='('){
                        while (TRUE){
                                if (buf[i]==')'){
                                        process=TRUE;
                                        break;
                                }
```

```
                        temp[j++]=buf[i];
                        i++;
                }
                temp[j]='\0';
                *itemNo=atoi(temp);
        }
        else
                i++;
    }

    j=0;
    process=FALSE;
    while (TRUE){
        if (process==TRUE || i>MAXLEN)
                break;
        else if (buf[i++]=='{'){
                while (TRUE){
                        if (buf[i]=='}'){
                                process=TRUE;
                                break;
                        }
                        temp[j++]=buf[i];
                        i++;
                }
                temp[j]='\0';
                *itemValue=atof(temp);
        }
        else
                i++;
    }
}
//-------------------------------------------------------------------
//Reads bids_file into bMatrix.
//-------------------------------------------------------------------
void processBidInput(char *buf){
    int i=0,j=0, process=FALSE;
    char temp[MAXLEN];

    while (TRUE){
        if (process==TRUE || i>MAXLEN)
                break;
        else if (buf[i++]=='{'){
                while (TRUE){
                        if (buf[i]=='}'){
                                process=TRUE;
                                break;
                        }
                        temp[j++]=buf[i];
                        i++;
                }
```

```
                temp[j]='\0';
                bMatrix[bidderListIdx].bidderID=atoi(temp);
        }
        else
                i++;
    }

    j=0;
    process=FALSE;
    while (TRUE){
        if (process==TRUE || i>MAXLEN)
                break;
        else if (buf[i++]=='{'){
                while (TRUE){
                        if (buf[i]=='}'){
                                process=TRUE;
                                break;
                        }
                        temp[j++]=buf[i];
                        i++;
                }
                temp[j]='\0';
        }
        else
                i++;
    }

    j=0;
    process=FALSE;
    while (TRUE){
        if (process==TRUE || i>MAXLEN)
                break;
        else if (buf[i++]=='{'){
                while (TRUE){
                        if (buf[i]=='}'){
                                process=TRUE;
                                break;
                        }
                        temp[j++]=buf[i];
                        i++;
                }
                temp[j]='\0';
                bMatrix[bidderListIdx].bidPrice=atof(temp);
        }
        else
                i++;
    }
    bidderListIdx++;
}
//-------------------------------------------------------------
```

APPENDIX C

Source Code for Greedy Search Technique

This appendix provides the source code of the greedy search technique (GST) program used in the evaluation.

```
/*-------------------------------------------------------------------
Author:         Andy Law
Date:           2003

Description:
The program uses greedy search technique (GST) to identify winners in a
combinatorial auction.

Program Execution Format:
bft.exe items_file.txt bids_file.txt
-------------------------------------------------------------------
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
//-------------------------------------------------------------------
#define MAXLEN          80
#define MAXITEM                 20
#define MAXBIDDER       1000
#define TRUE            1
#define FALSE           0
#define DELETED                 -1
#define NO_ITEM                 -2
//-------------------------------------------------------------------
typedef struct itemMatrix       //item information ADT
{
        double itemPrice;
} ITEMMATRIX;

typedef struct bidMatrix        //bid information ADT
{
        int bidderID;
        int itemList[MAXITEM];
        double subtotalValue;
        double bidPrice;
```

```
        int sold;                               //1 if sold, -1 if deleted
} BIDMATRIX;

//-----------------------------------------------------------------
BIDMATRIX bMatrix[MAXBIDDER]; //bids information
ITEMMATRIX iMatrix[MAXITEM];  //items information
int bidderListIdx=0;                    //total number of bids
int processedBidder=0;
int winner[MAXBIDDER];
double tempRev=0.0;

//-----------------------------------------------------------------
void readInput(char *argv[]);
void processItemInput(char *buf, int *itemNo, double *itemValue);
void processBidInput(char *buf);
void initialize();
void processAuction();
int auctionCompete(int max);
void displayResult();
//-----------------------------------------------------------------
//main() module
//-----------------------------------------------------------------
int main (int argc, char* argv[]) {
        clock_t start, finish;
        double duration;

        if (argc !=3)
        {
                printf("Usage: auction <item_file> <bid_file>\n");
                exit (1);
        }
        start = clock();

    printf( "%s\n",argv[1]);
    printf( "============================\n");
    printf( "   Greedy Search - Beginning\n");
    printf( "============================\n");
        initialize();
        readInput(argv);
        processAuction();
        displayResult();

        finish = clock();
    printf( "============================\n");
    printf( "    Gready Search - Ending\n");
    printf( "============================\n");
        duration = (double)(finish - start) / CLOCKS_PER_SEC;
        printf( "%.6lf seconds\n", duration );
        return 0;
}
//-----------------------------------------------------------------
//Initializes the matrix for storing the auction information
//-----------------------------------------------------------------
void initialize(){
        int i,j;
        for (i=0;i<MAXBIDDER;i++)
                winner[i]=-1;
        for (i=0;i<MAXBIDDER;i++){
                bMatrix[i].bidderID=-1;
```

```
                bMatrix[i].bidPrice=-1.0;
                bMatrix[i].sold=FALSE;
                bMatrix[i].subtotalValue=0;
                for (j=0;j<MAXITEM;j++)
                        bMatrix[i].itemList[j]=-2; //must be even number
        }
}
//-----------------------------------------------------------------
//Displays the auction results to the command screen.
//-----------------------------------------------------------------
void displayResult(){
        int i,j;
        double rev=0.0;

        printf("BidderID    Bid Price        ItemList\n");
        for (i=0;i<bidderListIdx;i++){
                if (bMatrix[i].sold== TRUE) {
                        printf("%5i        ",bMatrix[i].bidderID);
                        printf("%-15.2f",bMatrix[i].bidPrice);
                        rev=rev+bMatrix[i].bidPrice;
                        for (j=0;j<MAXITEM;j++){
                                if (bMatrix[i].itemList[j]==1)
                                        printf("%i  ",j);
                        }
                        printf("\n");
                }
        }
        printf("\n");
        printf("Total Revenue = %.2f",rev);
        printf("\n");
}
//-----------------------------------------------------------------
//Auction is processed.
//-----------------------------------------------------------------
void processAuction(){
        int i=0, highestBidIdx=-1;
        double highestPrice=0.0;

        while (1)       //always true
        {
                highestBidIdx=-1;
                highestPrice=0.0;
                if (processedBidder == bidderListIdx)
                        break;
                for (i=0;i<bidderListIdx;i++)
                {
                        if (bMatrix[i].sold==FALSE)
                                if (bMatrix[i].bidPrice>highestPrice)
                                {
                                        highestPrice=bMatrix[i].bidPrice;
                                        highestBidIdx=i;
                                }
                }
                auctionCompete(highestBidIdx);
        }
}
//-----------------------------------------------------------------
//Once the bid with highest bidding price or highest profit is found,
//all conflict bids will be deleted.
```

```
//-----------------------------------------------------------------
int auctionCompete(int max){
        int m,n;

        if (bMatrix[max].sold==DELETED)
                return 0;
        if (bMatrix[max].sold==FALSE){
                bMatrix[max].sold=TRUE;
                processedBidder++;
        }
        for (m=0;m<bidderListIdx;m++){
                if (bMatrix[m].sold==FALSE){ //
                        for (n=0;n<MAXITEM;n++){
                                if
((bMatrix[max].itemList[n]+bMatrix[m].itemList[n])==2){
                                        bMatrix[m].sold=DELETED;
                                        processedBidder++;
                                        break;
                                }
                        }
                }
        }
        return 0;
}
//-----------------------------------------------------------------
//Reads inputs into the internal memory structures.
//-----------------------------------------------------------------
void readInput(char *argv[]){
        char buf[81];
        int temp1=0;
        double temp2=0.0;

        FILE *fp1, *fp2;
        if ((fp1 = fopen(argv[1], "r"))==NULL){
                printf("Cannot open <item file>\n");
                exit (1);
        }
        if ((fp2 = fopen(argv[2], "r"))==NULL){
                printf("Cannot open <bid file>\n");
                exit (1);
        }
        while (TRUE){
                if (fgets(buf, MAXLEN, fp1)==NULL)
                        break;
                processItemInput(buf, &temp1, &temp2);
                iMatrix[temp1].itemPrice=temp2;
                //printf("%i %.2f\n",temp1, temp2);
        }
        while (TRUE){
                if (fgets(buf, MAXLEN, fp2)==NULL)
                        break;
                processBidInput(buf);
        }
}
//-----------------------------------------------------------------
//Reads items_file.txt into iMatrix.
//-----------------------------------------------------------------
void processItemInput(char *buf, int *itemNo, double *itemValue){
        int i=0,j=0, process=FALSE;
        char temp[MAXLEN];
```

```
        while (TRUE){
                if (process==TRUE || i>MAXLEN)
                        break;
                else if (buf[i++]=='{'){
                        while (TRUE){
                                if (buf[i]=='}'){
                                        process=TRUE;
                                        break;
                                }
                                temp[j++]=buf[i];
                                i++;
                        }
                        temp[j]='\0';
                        *itemNo=atoi(temp);
                }
                else
                        i++;
        }

        j=0;
        process=FALSE;
        while (TRUE){
                if (process==TRUE || i>MAXLEN)
                        break;
                else if (buf[i++]=='{'){
                        while (TRUE){
                                if (buf[i]=='}'){
                                        process=TRUE;
                                        break;
                                }
                                temp[j++]=buf[i];
                                i++;
                        }
                        temp[j]='\0';
                        *itemValue=atof(temp);
                }
                else
                        i++;
        }
}
//-----------------------------------------------------------------
//Reads bids_file.txt into bMatrix.
//-----------------------------------------------------------------
void processBidInput(char *buf){
        int i=0,j=0, process=FALSE;
        char temp[MAXLEN];

        while (TRUE){
                if (process==TRUE || i>MAXLEN)
                        break;
                else if (buf[i++]=='{'){
                        while (TRUE){
                                if (buf[i]=='}'){
                                        process=TRUE;
                                        break;
                                }
                                temp[j++]=buf[i];
                                i++;
```

```
                }
                temp[j]='\0';
                bMatrix[bidderListIdx].bidderID=atoi(temp);
            }
            else
                i++;
        }

        j=0;
        process=FALSE;
        while (TRUE){
            if (process==TRUE || i>MAXLEN)
                break;
            else if (buf[i++]=='{'){
                while (TRUE){
                    if (buf[i]=='}'){
                        process=TRUE;
                        break;
                    }
                    temp[j++]=buf[i];
                    i++;
                }
                temp[j]='\0';
                processBidItem(temp);
            }
            else
                i++;
        }

        j=0;
        process=FALSE;
        while (TRUE){
            if (process==TRUE || i>MAXLEN)
                break;
            else if (buf[i++]=='{'){
                while (TRUE){
                    if (buf[i]=='}'){
                        process=TRUE;
                        break;
                    }
                    temp[j++]=buf[i];
                    i++;
                }
                temp[j]='\0';
                bMatrix[bidderListIdx].bidPrice=atof(temp);
            }
            else
                i++;
        }
        bidderListIdx++;
}

//-----------------------------------------------------------------
//Stores those items that are wanted by particular bids in bidMatrix
//-----------------------------------------------------------------
void processBidItem(char itemList[]){
        char buf[10];
        int len = strlen(itemList);
        int i=0, j=0, itemListIdx=-1;
```

```
        double subtotal=0.0;

        while (TRUE){
                if (i>len)
                        break;
                else if (itemList[i]==' ' || itemList[i]=='\0') {
                        buf[j]='\0';
                        i++;
                        j=0;
                        itemListIdx=atoi(buf);
                        bMatrix[bidderListIdx].itemList[itemListIdx]=TRUE;
                        subtotal=subtotal+iMatrix[itemListIdx].itemPrice;
                }
                else
                        buf[j++]=itemList[i++];
        }
        bMatrix[bidderListIdx].subtotalValue=subtotal;
}
//---------------------------------------------------------------------
```

BIBLIOGRAPHY

[Agrawal et al., 1993] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In Proceedings of 1993 ACM-SIGMOD International Conference on Management of Data (SIGMOD'93), pages 207-216.

[Agrawal and Srikant, 1994] R. Agrawal and S. Sarawagi. Fast algorithms for mining association rules in large databases. In Research Report RJ9839, IBM Almaden Research Center.

[Agrawal and Srikant, 1995] R. Agrawal and S. Sarawagi. Mining sequential patterns. In Proceedings of International Conference on Data Engineering (ICDE), pages 3-14.

[Agrawal et al., 1996] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. Advances in Knowledge Discovery and Data Mining, pages 307-328. AAAI/MIT Press.

[Agrawal et al.,1997] R. Agrawal, A Gupta, and S. Sarawagi. Modeling multidimensional databases. In Proceeding 1997 International Conference Data Engineering(ICDE), pages 232-243.

[Agrawal et al., 2000] R. Agrawal, C. Aggarwal, and V.V.V. Prasad. A tree projection algorithm for generation of frequent itemsets. In Journal of Parallel and Distributed Computing.

[Aggarwal and Yu, 1999] C.C. Aggarwal and P.S. Yu. A new framework for itemset generation. In Proceedings 1998 of ACM Symposium on Principles of Database Systems (PODS), pages 18-24.

[Andersson et al.,2000] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In ICMAS, pages 39-46.

[Banks et al., 1989] J.S. Banks, J.O. Ledyard, and D.P. Porter. Allocating uncertain and unresponsive resources: an experimental approach, Rand Journal of Economics, vol. 20, 1, pages 1–25.

[Bichler, 1999] Martin Bichler. A roadmap to auction-based negotiation protocols for electronic commerce. In Proceedings of the 33rd Hawaii International Conference on System Sciences.

[Bjorndal and Jornsten, 2000] Mette Bjørndal and Kurt Jørnsten. An Analysis of a combinatorial auction. Department of Finance and Management Science, Norwegian School of Economics and Business Administration, Norway.

[Boutilier et al., 1999] C. Boutilier, M. Goldszmidt, and B. Sabata. Sequential auctions for the allocation of resources with complementarities. In Proceedings of IJCAI. Pages 527–534.

[Brin et al., 1997a] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: generalizing association rules to correlations. In Proceedings of 1997 ACM-SIGMOD International Conference on Management of Data (SIGMOD), pages 265-276.

[Brin et al., 1997b] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket analysis. In Proceedings of 1997 ACM-SIGMOD International Conference on Management of Data (SIGMOD), pages 255-264.

[Case, 2001] James Case. Mathematical challenges of combinatorial auction design. SIAM News, Volume 34, Number 5.

[Cooper and Steinberg, 1974] L. Copper and D. Steinberg. Methods and applications of linear programming. W.B. Saunders, Philadelphia.

[Cramton, 1997] P. Cramton. The FCC spectrum auctions: an early assessment. Journal of Economics and Management Strategy 6:3, pages 431-495.

[Cramton and Schwartx, 2000] P. Cramton, J.A. Schwartx. Collusive bidding: lessons from the FCC spectrum auctions. Journal of Regulatory Economics 17(3), pages 229-252.

[de Vries and Vohra, 2000] Sven de Vries and Rakesh Vohra. Combinatorial auctions: A survey. INFORMS Journal on Computing, Volume 15, No. 3.

[Dunham, 2003] Margaret H. Dunham. Data mining. Pearson Education, Inc. New Jersey, USA.

[Fujishima et al., 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In IJCAI, pages 548-553.

[Gonen and Lehmann, 2000] R. Gonen, D. Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In Proceeding of ACM Conference on Electronic Commerce (ACM-EC), pages 13–20.

[Goodrich and Tamassia, 2002] M.T. Goodrich and R. Tamassia. Algorithm Design. John Wiley & Sons.

[Graves et al., 1993] R. Graves, J. Sankaran, and L. Schrage. An auction method for course registration, Interfaces, 23, 5.

[Han et al., 2000] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In Proceedings of 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD), pages 1-12.

[Han and Fu, 1995] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In Proceedings of 1995 International Conference on Very Large Data Bases (VLDB), pages 420-431.

[Han and Kamber, 2001] Jiawei Han and Micheline Kamber. Data mining: concepts and techniques. Morgan Kaufmann Publishers, San Diego, CA, USA.

[Haeussler et al., 2002] Ernest F. Haeussler, Richard P. Paul, and Tech Laurel. Introductory mathematical analysis for business, economics and life and social sciences, 10th edition. Prentice-Hall, Inc.

[Hoffman and Padberg, 1993] K. Hoffman, and M.W. Padberg. Solving airline crew scheduling problems by branch and cut, Management Science, 39, 657–682.

[Holte, 2001] Robert C. Holte. Combinatorial auctions, knapsack problems and hill climbing search. In the Proceedings of AI'2001 (the Canadian conference on Artificial Intelligence), a volume in Springer's LNAI series.

[Hoos and Boutilier, 2000] Holger Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In Proceedings of the National Conference on Artificial Intelligence (AAAI), pages 22-29.

[Huberman et al., 1997] Bernardo A Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. Science, 275:51-54.

[Huberman et al., 2000] Bernado A Huberman, Tad Hogg and Arun Swami. Using unsuccessful auction bids to identify latent demand, Xeror Palo Alto Research Center.

[ILOG, 2005] ILOG. ILOG AMPL CPLEX SystemVersion 9.0 User's Guide, pages 64-66.

[Jackson, 1976] C. Jackson. Technology for spectrum markets, Ph. D. Thesis submitted to the Department of Electrical Engineering, School of Engineering, MIT.

[Johnsonbaugh and Schaefer, 2004] Richard Johnsonbaugh and Marcus Schaefer. Algorithms. Pearson Education, Inc. New Jersey, USA, pages 429-482.

[Karp, 1972] R. M. Karp. Reducibility among combinatorial problems. Complexity of Computer Computations, Plenum Press, NY, pages 85-103.

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

[Klemperer, 2000] Paul Klemperer. The economic theory of auctions. Edward Elgar Publishing.

[Krishna, 2002] Vijay Krishna. Auction theory. Academic Press, pages 1 – 10, 223 – 232.

[Lavi and Nisan, 2000] Ron Lavi and Noam Nisan. Competitive analysis of online auctions. In Proceedings of the 2nd ACM Conference on Electronic Commerce.

[Lawler, 1985] E. L. Lawler. The travelling salesman problem: A guided tour of combinatorial optimization. Wiley. New York.

[Lawler et al., 1992] E. L. Lawler, J. K. Lenstra, A. Kan, and D. B. Shmoys. The travelling salesman problem. Wiley Interscience.

[Lehmann et al., 1999] D. Lehmann, L. O'Callaghan, and Y. Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions, manuscript.

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

[Leyton-Brown et al., 2000b] K. Leyton-Brown, M. Tennenholtz, and Y. Shoham. An algorithm for multi-unit combinatorial auctions. In Proceedings of AAAI.

[Levitin, 2003] Anany Levitin. The design & analysis of Algorithms. Addison Wesley.

[Lustig and Puget, 2001] Irvin J. Lustig and Jean-Francois Puget. Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. In Interfaces 31: 6 pages 29–53.

[Mannila et al., 1994] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94), pages 181-192.

[Miller, 2000] Ronald E. Miller. Optimization. New York, John Wiley & Sons, Inc.

[Nemhauser and Wolsey, 1999] G.L. Nemhauser and L.A. Wolsey. Integer and Combinatorial Optimization. John Wiley & Sons.

[Nisan, 2000] Noam Nisan. Bidding and allocation in combinatorial auctions. In ACM Conference on Electronic Commerce, pages 1-12.

[Olson et al., 2000] M. Olson, P.J. Ledyard, J. Swanson, and D. Torma. The first use of a combined value auction for transportation services, Social Science Working Paper No. 1093, California Institute of Technology.

[Papadimitriou and Steiglitz, 1998] C.H. Papadimitriou and K. Steiglitz. Combinatorial optimization: Algorithms and complexity. Dover Publications.

[Park et al., 1995] J.S. Park, M.S. Chen, and P.S. Yu. Efficient parallel mining for association rules. In Proceedings of 4th International Conference on Information and Knowledge Management, pages 31-36.

[Park et al., 2000] S. Park, W.W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In Proceedings of 2000 International Conference on Data Engineering (ICDE), pages 23-32.

[Parkes,1999] David C Parkes. Optimal auction design for agents with hard valuation problems. In Agent-Mediated Electronic Commerce Workshop at the International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 1999.

[Pasquier et al, 1999] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In Proceedings of 7th International Conference on Database Theory (ICDT), pages 398-416.

[Pei et al., 2000] J. Pei, J. Han, and R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets. In Proceedings of 2000 ACM-SIGMOD International Workshop Data Mining and Knowledge Discovery (DMKD), pages 11-20.

[Pekeč and Rothkopf, 2000] Aleksandar Pekeč and Michael H. Rothkopf. Combination auction design. School of Business, Duke University

[Rassenti et al., 1982] S J Rassenti, V L Smith, and R L Bulfin. A combinatorial auction mechanism for airport time slot allocation. Bell J. of Economics, 13:402-417.

[Ronen, 2001] Amir Ronen. On approximating optimal auctions (extended abstract). In the 3rd ACM Conference on Electronic Commerce.

[Rothkopf et al., 1998] Michael H Rothkopf, Aleksandar Pekeč, and Ronald M Harstad. Computationally manageable combinatorial auctions. Management Science, 44(8):1131-1147.

[Rothkopf et al., 2000] Michael H Rothkopf and Aleksandar Pekeč. Making the FCC's first combinatorial auction work well. An official filing with the Federal Communications Commission.

[Russell and Norvig, 2003] Stuart Russell and Peter Norvig. Artificial intelligence. Pearson Education, Inc. New Jersey, USA, pages 712 – 762.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In IJCAI, pages 542-547.

[Sandholm, 2000] T. Sandholm. Issues in computational Vickrey auctions. In International Journal of Electronic Commerce 4 (3) (2000) 107–129.

[Sandholm et al., 2001a] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In AGENTS Workshop on Agent-Based Approaches to B2B.

[Sandholm et al., 2001b] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In IJCAI.

[Sandholm, 2002] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence, 135:1–54.

[Savasere et al., 1995] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In Proceedings of 1995 International Conference on Very Large Data Bases (VLDB), pages 432-443.

[Silverstein et al., 1998] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In Proceedings of 1998 International Conference on Very Large Data Bases (VLDB), pages 594-605.

[Sipser, 1992] M. Sipser. The history and status of the P versus NP question. Proceedings of 24th ACM Symposium on Theory of Computing, pages 603-618.

[Sipser, 1997] M. Sipser. Introduction to the Theory of Computation. PWS Publishing Company, Boston. Pages 223-271.

[Smith et al., 1997] B.M. Smith, S.C. Brailsford, P.M. Hubbard, and H.P. Williams. The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared.

[Srinivasan et al., 1998] S. Srinivasan, J. Stallert, and A.B. Whinston. Portfolio trading and electronic Networks, manuscript.

[Strevell and Chong, 1985] M. Strevell and P. Chong. Gambling on vacation, Interfaces, vol. 15, 63-67.

[Schuurmans et al., 2001] D. Schuurmans, F. Southey, and R.C. Holte. The Exponentiated Subgradient Algorithm for Heuristic Boolean Programming. In Proceedings of the 17th IJCAI.

[Tennenholtz, 2000] M. Tennenholtz. Some tractable combinatorial auctions. In Proceedings of AAAI.

[Toivonen, 1996] H. Toivonen. Sampling large databases for association rules. In Proceedings of 1996 International Conference on Very Large Data Bases (VLDB), pages 134-145.

[Wellman et al., 2001] M.P. Wellman, W.E. Walsh, P.R. Wurman, and J.K. MacKie-Mason. Auction protocols for decentralized scheduling, Games and Economic Behavior, 35(1-2), 271-303.