Comparative study of Heuristics in Improved winner determination in Combinatorial auction

REPORT SUBMITTED

BY

Rajnish Kumar

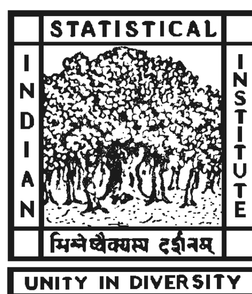
(ROLL NO: MTC0716)

AS A PARTIAL FULFILLMENT OF MASTER OF TECHNOLOGY(2-Yr)

IN COMPUTER SCIENCE

UNDER THE GUIDANCE OF

Prof. Bimal Kumar Roy(ASU, ISI kolkata)



INDIAN STATISTICAL INSTITUTE, KOLKATA

203 B. T. ROAD,

KOLKATA - 700108

JULY 2009

Indian Statistical Institute

Kolkata-700 108

CERTIFICATE

This is to certify that the thesis entitled "**Comparative study of heuristics in Improved winner determination in Combinatorial auction**" is submitted in the partial fulfilment of the degree of M. Tech. in Computer Science at Indian Statistical Institute, Kolkata. It is fully adequate, in scope and quality as a dissertation for the required degree.

The thesis is a faithfully record of bona fide research work carried out by Rajnish kumar under my supervision and guidance. It is further certified that no parts of this thesis has been submitted to any other university or institute for the award of any degree or diploma.

Prof. Bimal Kumar Roy
(Supervisor)

Countersigned
(External Examiner)
Date:

Acknowledgement

I take this opportunity to express my sincere gratitude to the supervisor of this study, Prof.Bimal kumar Roy(ASU, ISI kolkata) for his kind support and encouragement .I am also thankful to my co-adviser Prof.Asim Kumar Pal(IIM kolkata) to giving me opportunity to work under his guidance by collaborating himself with my supervisor to guide me for my dissertation .This work has been possible only because of his continuous suggestions, inspiration, motivation and full freedom given to me to incorporate my ideas. I also take this opportunity to thank all my teachers who have taught me in my M. Tech. course,and last but not the least I thank all my family and friends for their endless support.

Place : Kolkata
Date : Rajnish Kumar

Contents

1 Introduction **6**
- 1.1 Sequential auctions mechanisms 6
- 1.2 Parallel auction mechanisms 7
- 1.3 Method for fixing inefficient allocations 7
- 1.4 Combinatorial auction mechanisms 7

2 Winner determination in combinatorial auctions **8**
- 2.1 Enumeration of exhaustive partitions of items 8
- 2.2 Dynamic programming 9
- 2.3 More compact problem representation 10
- 2.4 NP-completeness of problem 10
- 2.5 Polynomial-time approximation algorithms 11
  - 2.5.1 Inapproximability 11

3 Problem definition **11**

4 Related work **12**

5 Description of algorithm **12**
- 5.1 Search space 12
- 5.2 Size of tree in MAIN SEARCH 14
- 5.3 Faster preferred child generation 16
- 5.4 Algorithm for construction of value added Bidtree 19
- 5.5 Preferred ordering of bids in MAIN SEARCH by value added Bidtree 20
- 5.6 Complexity of search 21

6 Heuristic computation **22**
- 6.1 Admissible heuristic 22
- 6.2 Faster heuristic computation via ITEM TREE 23
- 6.3 Incremental heuristic computation 25
- 6.4 Algorithm for Incremental heuristic computation 25

7 Preprocessing **25**

8 Algorithm for winner determination **25**
- 8.1 IDA^* Algorithm for winner determination 25
- 8.2 Time complexity of MAIN SEARCH 27

9 Experimental setup **27**

10 Experimental result **28**

11 Comparative study of heuristics in our algorithm for winner determination **31**
11.1 Definition 31
11.2 Introduction 31
11.3 Another heuristic function 31
11.4 Comparative heuristic estimate 33
11.5 Experimental setup 34
11.6 Experimantal result 34

12 Conclusion **43**

Abstract

Combinatorial auction is a type of auction where bidders can bid on combinations of items, tend to lead more efficient allocation of items between agents than traditional auction mechanisms where it might possible that agents' valuation be not additive. However, determining the winners so as to maximize revenue is NP-complete. In this report first, we will discuss about existing approaches for tackling this problem: exhaustive enumeration,dynamic programming and drawbacks. Second, we will discuss about the possibility of approximate winner determination in the general case with reasonable bound and Inapproximability result. After that we will present the possible existing approaches for optimal winner determination . We will study one of the search technique branch on items and try to improvise the search technique to find optimal winner determination in combinatorial auction with our search algorithm with improved data structure. Experiments are shown on weighted random bid distributions .This algorithm uses value added Bidtree for preferred child generation to make main search faster and item tree for faster heuristic computation. In later section, we will study the effect of different admissible heuristics on the performance of our algorithm by node generation,node expansion,item computation & bidtree node traversal. In last we will present experimental result for both of heuristics that is helpful to understand how heuristics effect performance in Winner determination in Combinatorial auction.

1 Introduction

Auctions are popular,distributed and autonomy-preserving ways of allocating items(good,resources,services,etc.)among agents. They are relatively efficient both in terms of process and income. They are exhaustively used among human bidders in a variety of task and resource allocation problems. More recently, Internet auction servers have been built that allow software agents to participate in the auctions.

In auction, the seller wants to sell the items and get highest possible payments for them while each bidders want to ac quire the items at lowest price. Auctions can be used among cooperative agents, but they also work in open systems consisting of self-interested agents. An auction can be analysed using noncooperative game theory: what strategies are self-interested agents best off using in the auction (and therefore will use), and will a desirable social outcome-for example, efficient allocation-still follow. Auction mechanisms can be designed so that desirable social outcomes follow even though each agent acts based on self-interest.

In this report,focuses on Auctions with multiple distinguishable items. These auctions are complex in the general case where the bidders have preferences over bundles, that is, a bidder's valuations for a bundle of items need not equal the sum of his valuations of the individual items in the bundle. This is often in the case for example,in electricity markets,equities trading,bandwith auctions, markets for trucking services, pollution right auctions, auctions for airport landing slote and auction for carrier-of-last-resort responsibilities for universal services.

1.1 Sequential auctions mechanisms

In a **Sequential auction** the items are auctioned one at a time. Determining the winners in such an auction is easy because that can be done by picking the highest bidder for each item separately. However, bidding in a sequential auction is difficult if the bidders have preferences over bundles. To determine her valuation for an item, the bidder needs to estimate what items she will receive in later auctions. This requires speculation on what the others will bid in the future because that affects what items she will receive. Furthermore, what the others bid in the future depends on what they believe others will bid, etc. This counter-speculation introduces computational cost and other wasteful overhead. Bidding rationally would involve optimally trading off the computational cost of lookahead against the gains it provides, but that would again depend on how others strike that tradeoff. Furthermore,even if lookahead were computationally manageable, usually uncertainty remains about the others bids because agents do not have exact information about each other. This often leads to inefficient allocations where bidders fail to get the combinations they want and get ones they do not.

1.2 Parallel auction mechanisms

In a **Parallel auction** the items are open for auction simultaneously, bidders may place their bids during a certain time period, and the bids are publicly observable. This has the advantage that the others bids partially signal to the bidder what the others bids will end up being so the uncertainty and the need for lookahead is not as drastic as in a sequential auction. However, the same problems prevail as in sequential auctions, albeit in a mitigated form.

In parallel auctions, an additional difficulty arises: each bidder would like to wait until the end to see what the going prices will be, and to optimize her bids so as to maximize payoff given the final prices. Because every bidder would want to wait, there is a chance that no bidding would commence. As a patch to this problem, activity rules have been used. Each bidder has to bid at least a certain volume (sum of her bid prices) by predefined time points in the auction, otherwise the bidders future rights are reduced. It looses the freedom for bidders. in some prespecified manner (for example, the bidder may be barred from the auction).

1.3 Method for fixing inefficient allocations

In **Sequential and parallel auctions**, the computational cost of lookahead and counter-speculation cannot be recovered, but one can attempt to fix the inefficient allocations that stem from the uncertainties discussed above.

One such approach is to set up an aftermarket where the bidders can exchange items among themselves after the auction has closed. While this approach can undo some inefficiencies, it may not lead to an economically efficient allocation in general, and even if it does, that may take an impractically large number of exchanges among the agents.

1.4 Combinatorial auction mechanisms

Combinatorial auction can be used to overcome the need for lookahead and the inefficiencies that stem from the related uncertainties. In a combinatorial auction, there is one seller (or several sellers acting in concert) and multiple bidders. The bidders may place bids on combinations of items. This allows a bidder to express complementarities between items so she does not have to speculate into an items valuation the impact of possibly getting other,complementary items. For example, the Federal Communications Commission sees the desirability of combinatorial bidding in their bandwidth auctions, but so far combinatorial bidding has not been allowed largely due to perceived intractability of winner determination. This report focuses on winner determination in combinatorial auctions where each bidder can bid on combinations (that is, bundles) of indivisible items, and any number of her bids can be accepted.the rest of report is organised as follows.

2 Winner determination in combinatorial auctions

We assume that the auctioneer determines the winnersthat is, decides which bids are winning and which are losing so as to maximize the sellers revenue. Such winner determination is easy in non-combinatorial auctions. It can be done by picking the highest bidder for each item separately. This takes $O(am)$ time where a is the number of bidders, and m is the number of items.

Unfortunately, winner determination in combinatorial auctions is hard. Let M be the set of items to be auctioned, and let $m = |M|$. Then any agent, i, can place a bid, $b_i(S) > 0$,for any combination $S \subseteq M$. We define the length of a bid to be the number of items in the bid.

Clearly, if several bids have been submitted on the same combination of items, for winner determination purposes we can simply keep the bid with the highest price, and the others can be discarded as irrelevant since it can never be beneficial for the seller to accept one of these inferior bids.The highest bid price for a combination is

$$\bar{b}(S) = max(b_i(S)) \tag{1}$$

If agent i has not submitted a bid on combination S,we say $b_i(S) = 0$.So if no bidder has submitted a bid on combination S, then we say $\bar{b}(S) = 0$

Winner determination in a combinatorial auction is the following problem. The goal is to find a solution that maximizes the auctioneers revenue given that each winning bidder pays the prices of her winning bids:

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

where W is a partition.

Deffinition 2.1 A *partition* is a set of subsets of items so that each item is included in at most one of the subsets. Formally,let $\mathbf{S} = \{S \subseteq M\}$.Then the set of partitions is

$$A = \{W \subseteq \mathbf{S} \mid S, S' \in W \rightarrow S \cap S' = \emptyset\} \tag{3}$$

Note that in a partition W, some of the items might not be included in any one of the subsets $S \in W$.

2.1 Enumeration of exhaustive partitions of items

One way to optimally solve the winner determination problem is to enumerate all *exhaustive partitions* of items.

Definition 2.2. An *exhaustive partition* is a partition where each item is included in exactly one subset of the partition. An example of the space of exhaustive partitions is presented. The number of exhaustive partitions grows rapidly as the number of items in the auction increases. The exact number of exhaustive partitions is

$$\sum_{q=1}^{m} Z(m, q) \tag{4}$$

where $Z(m, q)$ is the number of exhaustive partitions with q subsets, that is, the number of exhaustive partitions on level q of the graph. The quantity $Z(m, q)$ also known as the Stirling number of the second kind is captured by the following recurrence:

$$Z(m, q) = qZ(m-1, q) + Z(m-1, q-1), \tag{5}$$

where $Z(m, m) = Z(m, 1) = 1$. This recurrence can be understood by considering the addition of a new item to a setting with $m-1$ items. The first term, $qZ(m-1, q)$, counts the number of exhaustive partitions formed by adding the new item to one of the existing exhaustive partitions. There are q choices because the existing exhaustive partitions have q subsets each. The second term, $Z(m-1, q-1)$, considers using the new item in a subset of its own, and therefore existing exhaustive partitions with only $m-1$ previous subsets are counted.

Proposition 2.1.The number of exhaustive partitions is $O(m^m)$ and $\omega(m^{m/2})$[3].

2.2 Dynamic programming

Rather than enumerating the exhaustive partitions as above, the space of exhaustive partitions can be explored more efficiently using dynamic programming. Based on the $\bar{b}(S)$ function, the dynamic program determines for each set S of items the highest possible revenue that can be obtained using only the items in S. The algorithm proceeds systematically from small sets to large ones. The needed optimal substructure property comes from the fact that for each set, S, the maximal revenue comes either from a single bid (with price $\bar{b}(S)$) or from the sum of the maximal revenues of two disjoint exhaustive subsets of S. $C(S)$ corresponds to the smaller of these two subsets(or to the entire set S if that has higher revenue than the two subsets together). For each S,all possible subsets (together with that subsets complement in S) are tried.

The time savings from dynamic programming compared to enumeration come from the fact that the revenue maximizing solutions for the subsets need

not be computed over and over again, but only once. The dynamic program runs in $O(3^m)$ time [3]. This is significantly faster than enumeration. Clearly, the dynamic program also takes $O(2^m)$ timebecause it looks at every $S \subseteq M$. Therefore, the dynamic program is still too complex to scale to large numbers of items above about 20 or 30 in practice.

The dynamic program executes the same steps independent of the number of bids. This is because the algorithm generates each combination S even if no bids have been placed on S. Interpreted positively this means that the auctioneer can determine how long winner determination will take regardless of the number of bids that will be received. So,independent of the number of bids, dynamic programming is the algorithm of choice if the number of items is small. Interpreted negatively this means that the algorithm will scale only to a small number of items even if the number of bids is small. we present a search algorithm that avoids the generation of partitions that include combinations of items for which bids have not been submitted. That allows our algorithm to scale up to significantly larger numbers of items.

2.3 More compact problem representation

If no bid is received on some combination S, then those partitions W that include S need not be considered. The enumeration and the dynamic programming discussed above do not capitalize on this observation. By capitalizing on this observation, one can restrict attention to relevant partitions.

Definition 2.3:The set of relevant partitions is

$$A' = \{W \in A \mid S \in W \Rightarrow \texttt{bid has received on } S\}. \tag{6}$$

That is, one can restrict attention to the following set of combinations of items.:

$$\mathbf{S'} = \{S \subseteq M \mid \texttt{bid has been received on } S\} \tag{7}$$

2.4 NP-completeness of problem

The important question is not how complex the dynamic program is, because it executes the same steps regardless of what bids have been received. Rather, the important question is whether there exists an algorithm that runs fast in the size of the actual input, which might not include bids on all combinations. In other words, what is the complexity of problem ? Unfortunately, no algorithm can, in general, solve it in polynomial time in the size of the input (unless $P = NP$): the problem is NP-complete. If we take each bid as a set S (of items) and the price,$\bar{b}(S)$, as the weight of set S then the problem is reduced to weighted set packing. NP-completeness of winner determination then follows from the fact that weighted set packing is NP-complete.

2.5 Polynomial-time approximation algorithms

One could try to devise an algorithm that will establish a worst case bound, that is, guarantee that the revenue from the optimal solution is no greater than some positive constant, k times the revenue of the best solution found by the algorithm. A considerable amount of research has focused on generating such approximation algorithms that run in polynomial time in the size of the input.

2.5.1 Inapproximability

Proposition 2.2: For the winner determination problem, no polynomial-time algorithm can guarantee a bound $k \leq n^{1-\epsilon}$ for any $\epsilon > 0$ (unless $NP = ZPP$).[4]

Proof: Assume for contradiction that there exists a polynomial-time approximation algorithm that establishes some bound $k \leq n^{1-\epsilon}$ for the winner determination problem.Then that algorithm could be used to solve the weighted independent set problem to thesame k in polynomial time. This is because a weighted independent set problem instance can be polynomially converted into a winner determination instance while preserving approximability. This can be done by generating one item for each edge in the graph. A bid is generated for each vertex in the graph. The bid includes exactly those items that correspond to the edges connected to the vertex.

Since the algorithm will k-approximate the weighted independent set problem in polynomial time, it will also k-approximate the independent set problem in polynomial time. A polynomial-time k-approximation algorithm for the independent set problem could directly be used to k-approximate the maximum clique problem in polynomial time. This is because the maximum clique problem is the independent set problem on the complement graph. But Hastad showed that no polynomial-time algorithm can establish a $k \leq n^{1-\epsilon}$ for any $\epsilon > 0$ for the maximum clique problem (unless $NP = ZPP$). Contradiction.□

Above negative result shows that no polynomial-time approximation algorithm can be constructed for achieving a reasonable worst case guarantee.

3 Problem definition

We are interested to find out the faster optimal winner determination in combinatorial auction. Problem is, for given a set of items, bids will come for any combination of items, optimal winner determination will be done by efficient searching with efficient data structure. So we have to design a efficient algorithm for searching using an efficient heuristic so that more prunning is done. It will also depends on heuristic computation and nature of heuristic. So there is a scope of study to the comparison of differnt heuristics in this context, to designing a better heuristic.

4 Related work

Combinatorial auction is very popular topic as it gives desirable social outcome with economical efficiency. So many of the approaches had been developed to find optimal winner determination.There had several approaches for this problem one is branch on items and other is branch on bid. In this report we will study on branch on items by improving performance by improving data structure. In previous branch on item technique it has done through IDA^*. For main search(depth first search) there is need of Bidtree(a tree in which all leaf has same depth & bid at leaf) to add node in Main search tree. For heuristic estimate, there is a need of bid list (a list of bids corresponding to each item in which item belongs),to find maximum average value for an item. The worst case time complexity for heuristic estimation for m' items is $O(mm'b)$, b is greatest number of bids in which a item belongs, m is item size(maximum possible items in a bid).

5 Description of algorithm

5.1 Search space

We use tree search to achieve these goals. The input (after only the highest bid is kept for every combination of items for which a bid was receivedall other bids are deleted) is a list of bids, one for each $S \in \mathbf{S'}$

$$\{B_1, ...B_n\} = \{(B_1.S, B_1.\bar{b}), ...(B_n.S, B_n.\bar{b})\} \tag{8}$$

where $B_j.S$ is the set of items in bid j,and $\bar{b}$ is the price in bid j. Each path in our search tree consists of a sequence of disjoint bids ,that is bids that do not share items with each other ($B_j.S \cap B_k.S = \emptyset$ for all bids j and k on the same path. So, at any point in the search ,a path corresponds to relevant partition. Let U be the set of items that are already used on the path:

$$U = U_{j|B_j\textbf{is on the path}} B_j.S \tag{9}$$

and let F be the set of free items:

$$F = M - U \tag{10}$$

A path ends when no bid be addded to the path.This occurs when for very bid ,some of its items have already been used on the path $(\forall, B_j.S \cap U \neq \emptyset)$. As the search proceeds down a path,a tally g is kept of the sum of the prices of the bids on the path.

$$g = \sum_{j|B_j\textbf{is on the path}} B_j.\bar{b} \tag{11}$$

At every search node, the revenue from the path, that is, the g-value, is compared to the best g-value found so far in the search tree to determine whether

the current solution (path) is the best one so far. If so, it is stored as the best solution found so far. Once the search completes, the stored solution is an optimal solution. here we have assumed that for all items bid has come. It is quite natural assumption. Otherwise we can manually add a dummy bid with 0 value.This assumption is lying in the fact that every relevant partition $W \in A'$ is captured by at least one path from the root to a node (interior or leaf) in the search tree. This guarantees that the algorithm finds the optimal solution. We had used dummy bid technique.

A naive method of constructing the search tree would include all bids (that do not include items that are already on the path) as the children of each node. Instead, the following proposition enables a significant reduction of the branching factor by capitalizing on the fact that the order of the bids on a path does not matter.

Figure 1.3

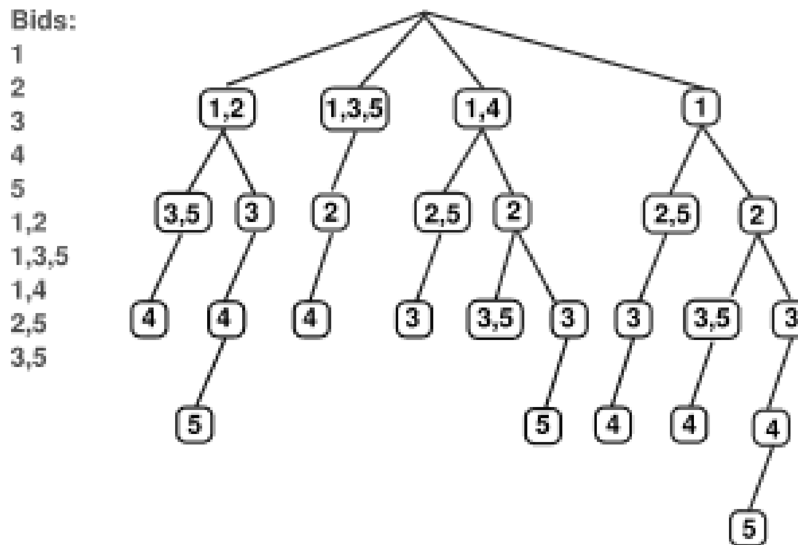


Figure 1.1 MAIN SEARCH tree

Proposition 3.1: Every relevant partition $W \in A'$ is represented in the tree by exactly one path from the root to a node (interior or leaf) if the children of a node are those bids that include the item with the smallest index among the items that have not been used on the path yet. [3]

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

- do not include items that have already been used on the path.

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

$$chilldren(\theta) = \{B \in \{B_1, ...B_n\} : i^* \in B.S, B.S \cap U = \phi\}. \tag{12}$$

Proof : We first prove that each relevant partition $W \in A'$ is represented by at most one path from the root to a node. The first condition of the proposition leads to the fact that a partition can only be generated in one order of bids on the path. So, for there to exist more than one path for a given partition,some bid would have to occur multiple times as a child of some node. However, the tree uses each bid as a child for a given node only once.

What remains to be proven is that each relevant partition is represented by some path from the root to a node in the tree. Assume for contradiction that some relevant partition $W \in A'$ is not. Then, at some point, there has to be a bid in that partition such that it is the bid with the item with the smallest index among those not on the path, but that bid is not inserted to the path. Contradiction. □

Our search algorithm restricts the children according to Proposition 3.1. this can be seen from above Figure 1.1 because all the bids considered at the first level include item 1. Figure.1.1 also illustrates the fact that the minimal index,i^*, does not coincide with the depth of the search tree in general.

To summarize, in the search tree, a path from the root to a node (interior or leaf) corresponds to a relevant partition. Each relevant partition $W \in A'$ is represented by exactly one such path. The other partitions $W \in A - A'$ are not generated. We call this search MAIN SEARCH.

5.2 Size of tree in MAIN SEARCH

Here we will analyse the worst case size of tree in MAIN SEARCH.

Propsition 3.2 The number of leaves in MAIN SEARCH is no greater than $(n/m)^m$. Also, the number of leaves in MAIN SEARCH is no greater than $\sum_{q=1}^{m} Z(m, q) \in O(m^m)$ (see Eqs. (4) and (5) for the definition of Z). Furthermore, the number of leaves in MAIN SEARCH is no greater than 2^n .

The number of nodes in MAIN SEARCH (excluding the root) is no greater than m times the number of leaves. The number of nodes in MAIN SEARCH is no greater than 2^n.[3]

Proof. We first prove that the number of leaves is no greater than $(n/m)^m$.The depth of the tree is at most m since every node on a path uses up at least one item. Let N_i be the set of bids that include item i but no items with a smaller index than i. Let $n_i = | N_i |$. Clearly, $\forall i, j, N_i \cap N_j = \phi$, so $n_1 + n_1 + \cdots + n_m = n$.

An upper bound on the number of leaves in the tree is given by $n_1 n_2 \ldots n_m$ because the branching factor at a node is at most n_i and i^* increases strictly along every path in the tree.The maximization problem.

max $n_1, n_2. \ldots . n_m$ s.t. $n_1 + n_2 + \cdots + n_m = n$

is solved by $n_1 = n_2 = \ldots . . = n_m$,even if n is not divisible by m,the value of the maximization is an upper bound.Therefore,the number of leaves in the tree is no greater than $(n/m)^m$.

Now prove that the number of leaves in MAIN SEARCH is no greater than$\sum_{q=1}^{m} Z(m,q) \in O(m^m)$. Since dummy bids are used, each path from the root to a leaf corresponds to a relevant exhaustive partition. Therefore the number of leaves is no greater than the number of exhaustive partitions (and is generally lower since not all exhaustive partitions are relevant). In Section 2.1 we showed that that the number of exhaustive partitions is $\sum_{q=1}^{m} Z(m,q)$. By Proposition 2.1, this is $O(m^m)$.

Next we prove that the number of nodes (and thus also the number of leaves) is no greater than 2^n. There are 2^n combinations of bids (including the one with no bids). In the search tree, each path from a root to a node corresponds to a unique combination of bids (the reverse is not true because in some combinations bids share items, so those combinations are not represented by any path in the tree). Therefore, the number of nodes is no greater than 2^n. Because there are at most m nodes on a path (excluding the root), the number of nodes in the tree (excluding the root) is no greater than m times the number of leaves.□

Proposition 3.3 The bound $(n/m)^m$ is always tighter (lower) than the bound 2^n .

Proof:

$$(n/m)^m < 2^n \leftrightarrow m \log(n/m) < n \leftrightarrow \log(n/m) < n/m. \tag{13}$$

which holds for all positive numbers n and m. □

The bound $(n/m)^m$ shows that the number of leaves (and nodes) in a MAIN SEARCH is polynomial in the number of bids even in the worst case if the number of items is fixed. On the other hand, as the number of items increases, the number of bids also increases ($n \geq m$ due to dummy bids), So in the worst case, the number of leaves (and nodes) in MAIN SEARCH remains exponential in the number of items m.

5.3 Faster preferred child generation

1 . {1, 2} - $4.5, 2. {1, 2, 3} - $6, 3. {1, 3} - $8, 4. {2, 4} - $3,

5. {2, 3} - $5, 6.{2} - $2, 7. {1} - $1, 8. {3} - $1, 9. {3, 4} - $6
10. {4} - $4

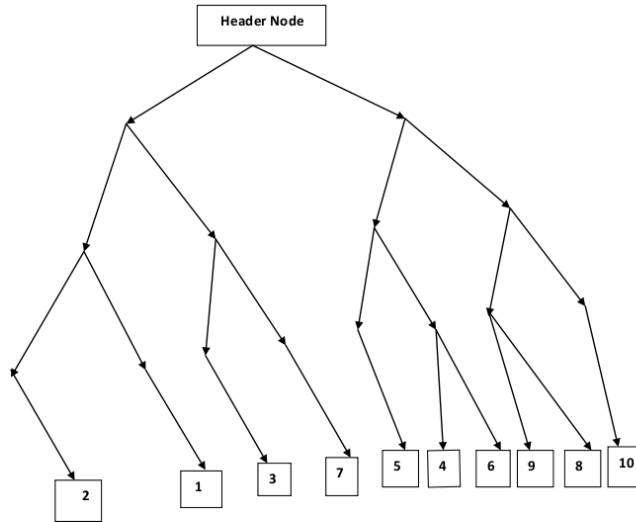


Figure 2.1 Bidtree

1 . {1, 2} - $4.5, 2. {1, 2, 3} - $6, 3. {1, 3} - $8, 4. {2, 4} - $3,
5. {2, 3} - $5, 6. {2} - $2, 7. {1} - $1, 8. {3} - $1,
9. {3, 4} - $6, 10.{4} - $4

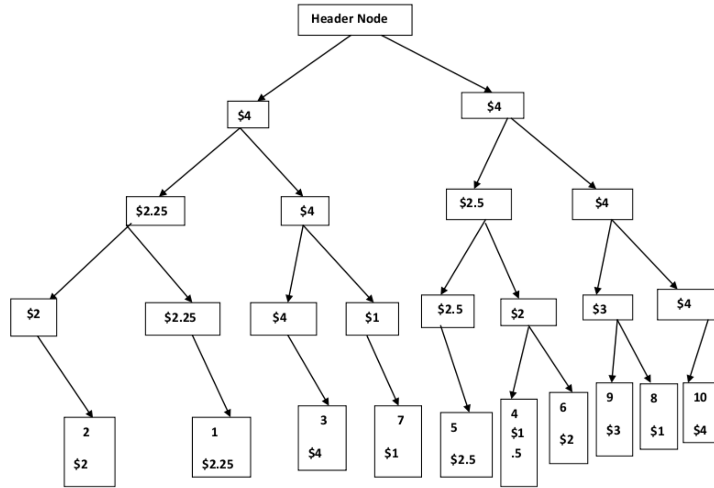


Figure 2.2 value adde Bidtree

At any given node,θ, of the tree, MAIN SEARCH has to determine children(θ). In other words, it needs to find those bids that satisfy the two conditions of Proposition 3.1. We use a more sophisticated scheme to make child generation faster. Our version of MAIN SEARCH uses a secondary depth-first search, SEARCH BID TREE(searching a bid for MAIN SEARCH in value added Bidtree) to quickly determine the children of a node. SEARCH BID TREE takes place in a different space: a data structure which we call the value added bidtree. It is a binary tree in which the bids are inserted up front as the leaves (only those parts of the tree are generated for which bids are received). It will take care of preprocessing that ensure the only maximum value bid (e.g several bids has come for same combination of items only maximum value bid is selected) is inserted. In value added Bidtree every node has information about which sub Bidtree has maximum average valued item & size of bid(number of items in a bid).

We can see in in Figure 2.1 for previous Bidtree (in which node has no information about in which sub Bidtree maximum average valued item is) but Figure 2.2 shows that how value added Bidtree node have information about sub Bidtree having maximum average valued item. As we see from the Figure 3.2 that value added Bidtree for same set of bids S, has 5 node generation(node actually generated) & 1 node expansion(node generated except leaves) in a MAIN

SEARCH where as in Bidtree (Figure 3.1) the node generation is 7 & node expansion is 3. Dotted bids is prunned in MAIN SEARCH by value added Bidtree, but in Bidtree it is generated due to expansion of interior node so less prunng is done via Bidtree.

The use of a Stopmask differentiates the value added Bidtree from a classic binary tree. The Stopmask is a vector with one variable for each item,$i \in M$. Stopmask[i] can take on any one of three values: BLOCKED, MUST, or ANY PREFERRED. If Stopmask[i] = BLOCKED, SEARCH BID TREE will never progress left at depth i. This has the effect that those bids that include item i are pruned instantly and in place. If, instead, Stopmask[i] = MUST, then SEARCH BID TREE cannot progress right at depth i. This has the effect that all other bids except those that include item i are pruned instantly and in place. Stopmask[i] = ANY PREFERRED corresponds to no pruning based on item i: SEARCH2 may go left or right at depth i. Fig. 2.3 illustrates how particular values in the Stopmask prune the Bidtree.

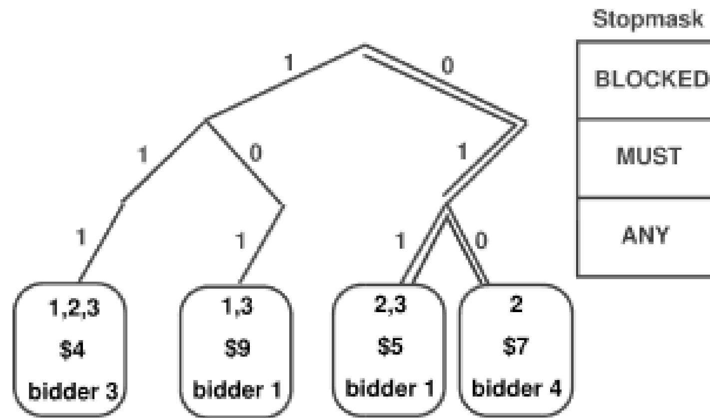


Figure 2.3 value added Bidtree traversal

SEARCH BID TREE is used to generate children in MAIN SEARCH. The basic principle is that at any given node of MAIN SEARCH, Stopmask[i] = BLOCKED for all $i \in U$, and Stopmask[i^*] =MUST, and Stopmask[i] = ANY PREFERRED for all other values of i. Given these variable settings, SEARCH BID TREE will return exactly those bids that satisfy Proposition 3.1.

We deploy a faster method for setting the Stopmask values. In what follows, we present the technique in detail. When MAIN SEARCH begins, Stopmask[1] = MUST, and Stopmask[i] = ANY PREFERRED for $i \in \{2, ..., m\}$. The first child of any given node,θ, of MAIN SEARCH is determined by a depth- first

search (SEARCH BID TREE) from the top of the value added Bidtree until a leaf (bid) is reached. This bid becomes the node that is added to the path of MAIN SEARCH. Every time a bid,B, is appended to the path of MAIN SEARCH, the algorithm sets Stopmask[i] = BLOCKED for all $i \in B.S$ and Stopmask[i] = MUST. These MUST and BLOCKED values are changed back to ANY PREFERRED when backtracking a bid from the path of MAIN SEARCH, and the MUST value is reallocated to the place in the Stopmask where it was before that bid was appended to the path. The next unexplored sibling of any child, q, of MAIN SEARCH is determined by continuing SEARCH BID TREE by backtracking in the value added Bidtree after MAIN SEARCH has explored the tree under q. Note that SEARCH BID TREE never needs to backtrack above depth i in the value added Bidtree because all items with smaller indices than i are already used on the path of MAIN SEARCH.

To ensure bactracking and to efficiently find appropriate bid. we had introduced three fields for every node in value added Bidtree. Fields are INFLAG, OUTFLAG & MAX AVG if node is visited then its INFLAG will be 1 otherwise 0. If all of the bids already explored below a node then OUTFLAG is 1 otherwise 0.INFLAG nonzero tells that this node has visited but one can do further exploration it depends on the value of OUTFLAG. If OUTFLAG is 0 then it can be explored otherwise exploration is not possible you should backtrack from that node to find appropriate bid. OUTFLAG nonzero means threre is no chance of further exploration. you have to backtrack. MAX AVG stores the maximum average value of an item in any of right or left sub Bidtree.

5.4 Algorithm for construction of value added Bidtree

for i=1 to m(item size)

If (item i exists & left child node has not generated earlier)
then node is generated ,INFLAG & OUTFLAG is 0,MAX AVG
is assigned by this average value of an item of an incoming bid

else if(item i exist & left child node has already generated)

if (average value of an item of an incoming bid is greater than MAX AVG)
then MAX AVG is updated by average value of an item of an
incoming bid

else no change;

else if (item not i exist & right child node has not generated)
node is generated ,INFLAG & OUTFLAG is 0,MAX AVG

is assigned by this average value of an item of an incoming bid

else
  if (average value of an item of an incoming bid is greater than MAX AVG)
    then MAX AVG is updated by average value of an item of an incoming bid

  else no change;

if(bid is not placed in leaf)
  bid is placed at leaf with INFLAG & OUTFLAG 0 ,MAX AVG is its average value of an item of an incoming bid
else
  if (MAX AVG less than average value of an item of an incoming bid)
    MAX AVG is updated by average value of an item of an incoming bid
  else no change;

5.5 Preferred ordering of bids in MAIN SEARCH by value added Bidtree

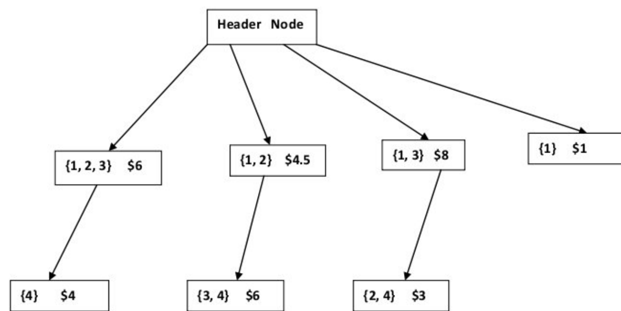


Figure 3.1 MAIN SEARCH by Bidtree traversal

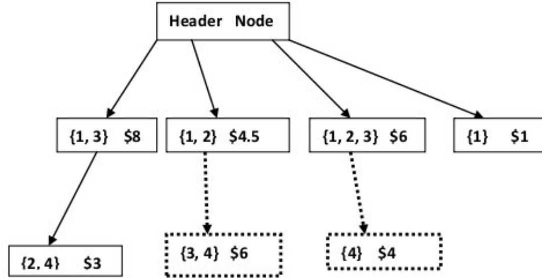


Figure 3.2 MAIN SEARCH by value added Bidtree traversal

By using value added bidtree we get an advantage over Bidtree is that is not follow a strict lexicographic ordering $< 0,1,1,0,0 >$,$< 0,1,0,1,1 >$,$< 0,1,0,1,0 >$.So previously Bidtree generates child in a order independent of value of bid,but in the case of value added Bidtree it generates child which is more promising to lead the solution early by pruning depth first search tree(MAIN SEARCH)earlier. we can see from Figure 3.1 and Figure 3.2 how value added Bidtree has done more pruning than Bidtree.

5.6 Complexity of search

The approach to generating children(θ) takes $O(nm)$ time in the worst case. As desired, the use of SEARCH BID TREE to generate children(θ) reduces this complexity,even in the worst case. For any given θ, finding the entire set children(θ), one at a time, corresponds to conducting a depth-first search (SEARCH BID TREE) in the value added Bidtree.The complexity of SEARCH BID TREE is no greater than the number of edges in the Bidtree times two (to account for backtracks). The following proposition gives a tight upper bound on the number of edges in the Bidtree.

proposition 3.4 In a tree that has uniform depth $m + 1$ (under the convention that the root is at depth 1), n leaves, and where any node has at most two children (as is the case in SEARCH BID TREE), the number of edges is at most[3]

$$nm - n \log n + 2.2^{\log n} - 2 \tag{14}$$

this bound is tight.Under the assumption that $m - \log n \geq c$ for some constant $c > 0$,this is

$$O(n(m - logn)). \tag{15}$$

on the other hand ,under the asumption that $m - \log n < c$,this is

$$O(n) \tag{16}$$

So worst case time complexity of SEARCH BID TREE in value added Bidtree has not reduced from Bidtree. Finding a preferred child through a value added Bidtree in SEARCH BID TREE has advantage over Bidtree. Worst case time complexity for SEARCH BID TREE by value added Bidtree is also $O(n)$.

6 Heuristic computation

6.1 Admissible heuristic

Definition: Winner determination problem is maximization problem so **Admissible heuristic** is defined as Heuristic function(e.g $h_1(F)$) should never underestimate the revenue from the unallocated items on the Path.

Heuristic 3.1:Heuristic $h_1(F)$ for unallocated items is computed using following function:

$$h_1(F) = \sum_{i \in F} c(i) \qquad \textbf{where} \qquad c(i) = max_{S|i \in S} \frac{\bar{b}(S)}{\mid S \mid} \tag{17}$$

where $\bar{b}(s)$ is value of a bid,$\mid S \mid$ is the number of items in a bid. For speeding the MAIN SEARCH we had used Incremental Heuristic computation.

Proposition 3.5 From Heuristic 3.1 gives an upper bound on how much revenue the unallocated items F can contribute.

Proof: For any set $S \in F$, if a bid for S is determined to be winning, then each of the items in S contributes $\frac{\bar{b}(S)}{|S|}$ toward the revenue. Every item can be in only one winning bid.Therefore, the revenue contribution of any one item i can be at most $max_{S|i \in S} \frac{\bar{b}(S)}{|S|}$ To get an upper bound on how much all the unallocated items F together can contribute,so sum $max_{S|i \in S} \frac{\bar{b}(S)}{|S|}$ over all $i \in F$.

6.2 Faster heuristic computation via ITEM TREE

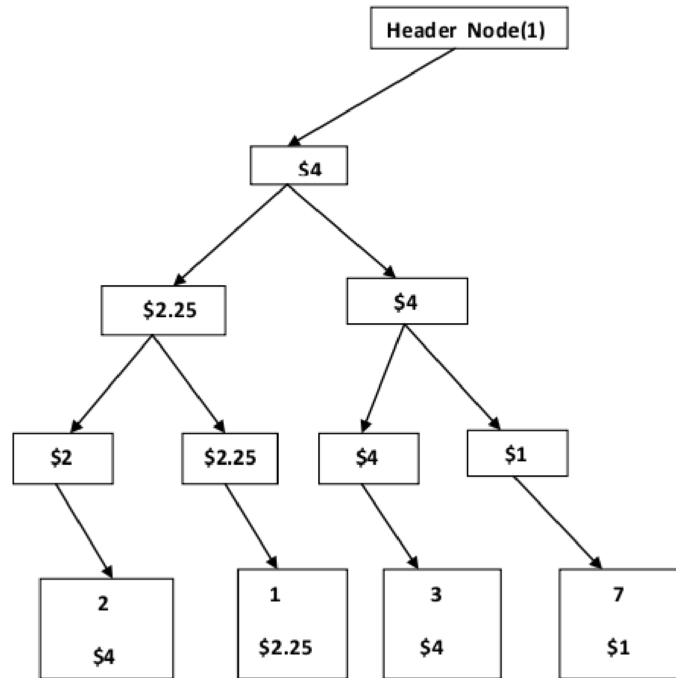


Figure 3.3 ITEM TREE for item 1

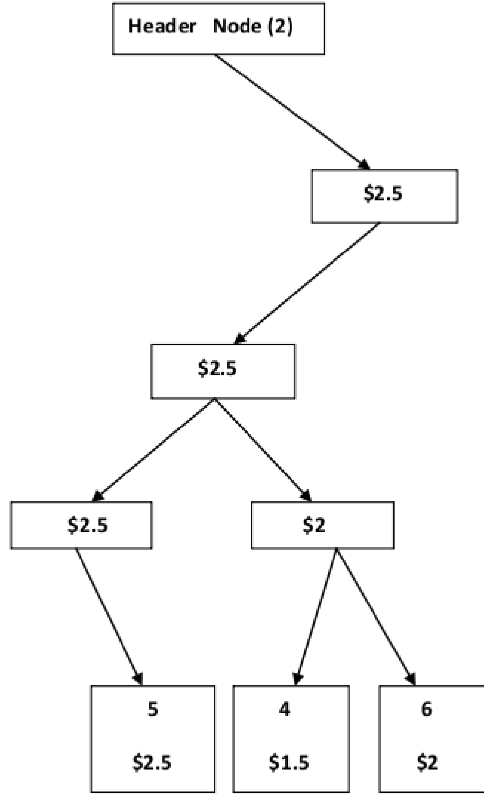


Figure 3.4 ITEM TREE for item 2

Here we will discuss about how we achieve faster heuristic computation. It is done via ITEM TREE(Figure 3.3 & 3.4). ITEM TREE is similar with value added Bidtree & construction is also same having the same fields like INFLAG,OUTFLAG & MAX AVG. Only difference is that ITEM TREE is constructed for every item separately and all the bids in the leaf for corresponding ITEM TREE having that item necessarily. Due to this ITEM TREE traversal (by using stopmask), worst case time complexity is $O(n')$ same as value added Bidtree traversal (to find a bid) for finding a bid for maximum average value of item. Hence by implementing ITEM TREE, Worst case time complexity for heuristic computation will reduce to $O(m'n')$,(m'- number of items for heuristic esimate,n'-greatest number of bids to which an item belongs). In previous case(traversing a list of bids for each item), worst case time complexity is $O(m'mn')$ for heuristic computation at a node(m-comes because of traversing through a bid to ensure there is no allocated item in a bid).

6.3 Incremental heuristic computation

In heuristic estimate for rest of items that has not yet allocated in the MAIN SEARCH is computed for each item separately. So by moving along path in MAIN SEARCH, it may possible that we have to recompute the heuristic estimate an item again and again. So inherently it will increase our overhead. To encounter this problem we initially maintain a MAX LIST which initialise by taking bid having maximum average value of an item for that corresponding item. It will be updated with moving along MAIN SEARCH path. If we have to compute heuristic for an item we first go through the MAX LIST and see any of allocated items in path have not exist in the corresponding bid for that item in MAX LIST then we will get the value for that item from this MAX LIST in $O(1)$ otherwise we have to search through ITEM TREE.

6.4 Algorithm for Incremental heuristic computation

INPUT: ITEM TREE ,MAX LIST & item i.

OUTPUT:Return maximum average value(MAX AVG) for item i(exclude bids having item allocated in current MAIN SEARCH path.

If bid $\in$ MAX LIST(i) not having any item allocated to current MAIN SEARCH path
    return(MAX AVG);
else
    search bid in ITEM TREE(i);
    return(MAX AVG);
Update MAX LIST

7 Preprocessing

When a bid arrives, it is inserted into the value added Bidtree. If a bid for the same set of items S already exists in the Bidtree (i.e. the leaf that already exists for new bid), only the bid at leaf will change its MAX AVG if new bid has more average value of an item than previous one,otherwise bid is discarded. Inserting a bid into the Bidtree will take m steps.There are n bids to insert. So, the overall time complexity of preprocessing with value added Bidtree construction is $O(mn)$.

8 Algorithm for winner determination

8.1 IDA^* Algorithm for winner determination

INPUT: A set of bids with its value
OUTPUT: Returns a set of winning bids that maximises the revenue

Global variable: f_limit

Algorithm 3.1 (IDA^* for winner determination).
//Returns a set of winning bids that maximizes the sum of the bid prices

1. $f_{limit} := \infty$
2.Construction of Value added Bidtree
3.Construction of ITEM TREE
4.Construction of MAX LIST;
5. Loop
    (a) winners, new-f := DFS-CONTOUR(M',ϕ, 0)
    (b) if winners $\neq$ null then return winners
    (c) f-limit := min(new-f, 0.90.f_{limit})

Algorithm 3.2 DFS-CONTOUR(F , winners, g).//depth first search
Returns a set of winning bids and a new f_{cost}

1. Compute $h_1(F)$ // Methods for doing this are presented, F is set of unallocated items
    for $i \in F$
        c(i)=Incremental heuristic computation(MAX LIST,ITEM TREE,i));
        Updation of MAX LIST
    $h_1(F) = \sum_{i \in F} c(i)$;// as definition from $h_1(F)$

3. If $g + h_1(F) < f_{limit}$ then return null, $g + h_1(F)$ // Pruning

4. Updation of **Stopmask** // global array variable

5. IF SEARCH BID TREE returns no children, then // End of a path reached

    (a)Update the fields of **value added bidtree**
        by making OUTFLAG zero for all node correspondinrg to bid, recently
            added
    (b)$f_{limit} = g$ // Revert to Branch & Bound once first leaf is reached
    (c) return winners, g

6. max-Revenue := 0, best-Winners := null, next-f := 0

7.generate children by SEARCH BID TREE of a node until no more child,and making a **sibling list**

8. For each bid $\in$ **sibling list**
    (a) solution, new-f := DFS-CONTOUR(F-bid.S, winners $\cup$ bid, g+ $bid.b$)

(b) If solution $\neq$ null and new-f > maxRevenue, then
   i. max-Revenue := new-f
   ii. best-Winners := solution
(c) next-f := max(next-f, new-f)

9. If bestWinners $\neq$ null then return bestWinners, maxRevenue
   else return null, next-f

8.2 Time complexity of MAIN SEARCH

The time complexity for per MAIN SEARCH node is $O(n'm')$ because each of the node activities have child generation(it requires $O(n')$),h_1-function computation(it reqires $O(m'n')$) if m' items are unallocated. So Overall time complexity for a node genration in MAIN SEARCH is $O(n'm')$. However IDA^* can generates some of nodes multiple times because all of the nodes generated in one iteration are regenerated in next iteration. Each iteration generates at least one more search node than previous iteration. Therefore, if the MAIN SERACH has p nodes (it returned the optimal solution) then it means it has generated at most $\frac{p(p+1)}{2}$ search nodes, but in practice it will generate fewer node because f_{limit} is decreased more across iterations.

9 Experimental setup

To determine the efficiency of the algorithm in practice, we ran experiments on a core 2-duo microprocessor (2 GHz with 1 Gigabyte of RAM) in C with following bid distributions:

- **Weighted random**: pick the number of items randomly from 1,2.....,m. Randomly choose that many items without replacemant. Pick the value of bid randomly between 1 and the number of items in the bid.

It may possible that same bid can be generated more than once. In real scenario many bids come for same combination of items, but our preprocessing will take care of that ,actually during value added Bidtree construction low value bid for same combination of items will be discarded. We had taken average value for 50 runs for each item size & bid size combination.

10 Experimental result

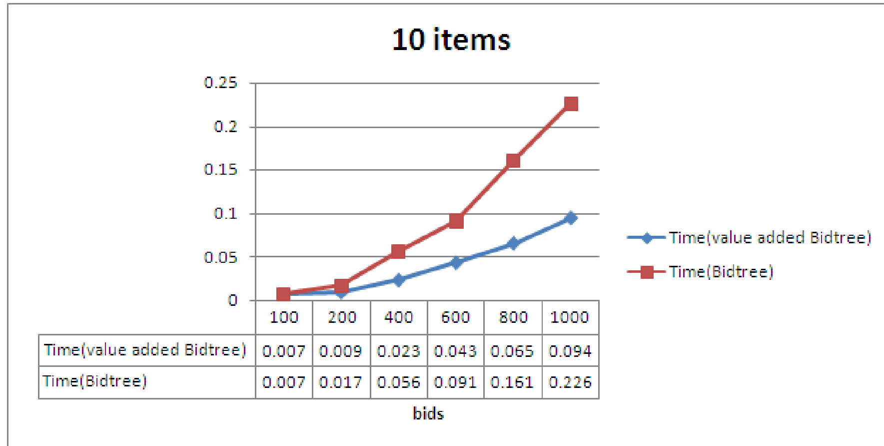


Figure 4.1

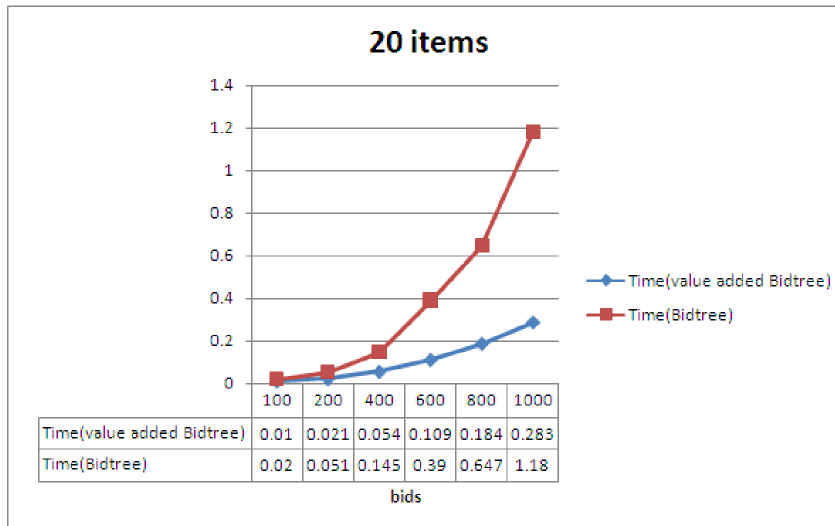


Figure 4.2

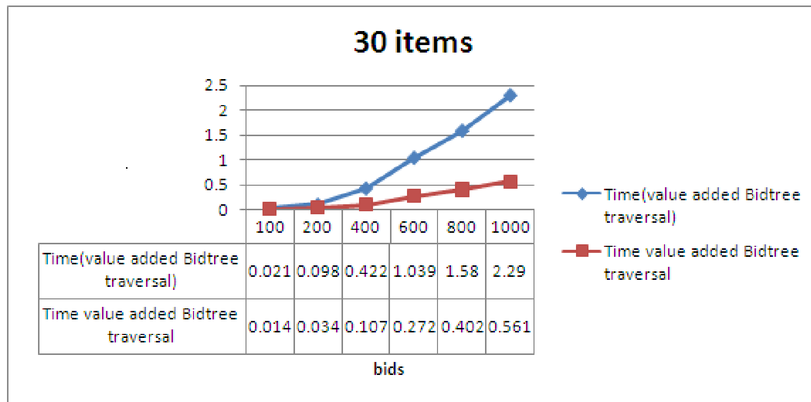


Figure 4.3

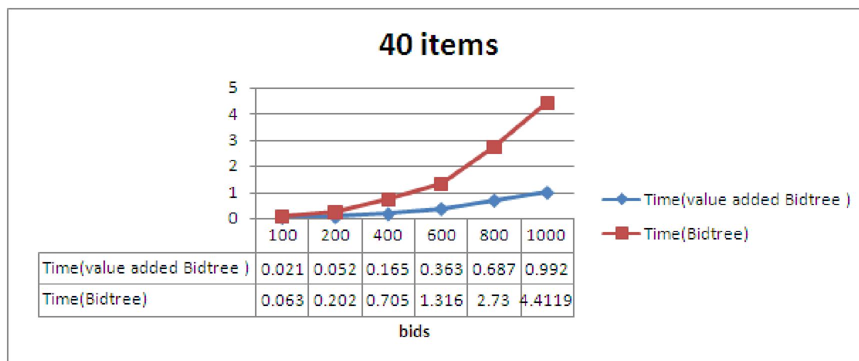
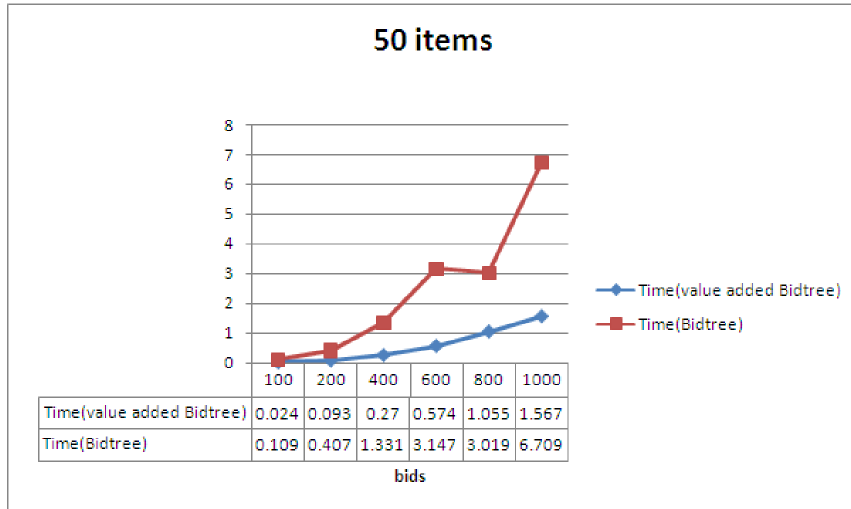


Figure 4.4



	100	200	400	600	800	1000
Time(value added Bidtree)	0.024	0.093	0.27	0.574	1.055	1.567
Time(Bidtree)	0.109	0.407	1.331	3.147	3.019	6.709

Figure 4.5

In this section we will discuss about the result we have got by using value added Bidtree & ITEM TREE data structure. Initially in literature[3] is suggesting about Bidtree same as value added Bidtree except that node has not an information about path to leaf node having maximum average valued item. Another data structure is ITEM TREE is also a value addded Bidtree having all the leaf node the proper item, it is used in retrieving bid(MAX AVG) having maximum average valued item faster. Its worst case time complexity is $O(n')$ (Proposition 3.4), where n' is number of bids in which item belongs, earlier it was $0(mn')$(traversing through bid as well as bid list). we have also used the MAX LIST which is used for faster heuristic computation by incremental heuristic computation.

The 90% f-limit criterian is used to decrease the f-limit faster between iteration. we cannot take rate of decreasing for f-limit too fast because it will increase depth of (MAIN SEARCH)depth first tree tremendously in last iteration and more node generation would be possible so less prunnng will be there, apart from that if rate of decreasing is too slow then new iteration repeats a large portion of search from previous iteration. A good rate of decreasing is achieved by running algorithm to various rate of decreasing for f-limit. So,90% criteria is good rate of decreasing the f-limit, is achieved through experimentation. In previous case bids are generated in lexicographic order like $< 0, 1, 1, 0, 0 >, < 0, 1, 0, 1.1 >, < 0, 1, 0, 1, 0 >$ irrespective of the bid value, but in our case we first chose those children having maximum average valued item so it is helpful to more pruning of depth first tree because we get probable solution earlier in depth first search tree(MAIN SEARCH).

11 Comparative study of heuristics in our algorithm for winner determination

11.1 Definition

Node generation: It is the number of nodes actually generated in MAIN SEARCH.

Node expansion: It is the number of nodes except leaf node in MAIN SEARCH.

Item computation: During heuristic computation, compute the value of item for heuristic estimate.

Bidtree node traversal: During searching for child in MAIN SEARCH, value added Bidtree is traversed from root to leaf through its node called Bidtree node traversal.

11.2 Introduction

In initial section we had seen that we can improve performance by using efficient data structure.In this section and further we will study about different heuristic and its impact on performance. Actually we have seen that the performance of algorithm depends on node generation,node expansion,value added Bidtree traverasal,heuristic computation. Heuristic computation & value added Bidtree traversal depends on the node generation,& node expansion. So if we give tighter bound heuristic function it seems that that it will do more pruning. So we will study about two admissible heuristic function and how it affect performance of IDA^* algorithm in our case.

11.3 Another heuristic function

Heuristic 3.2: We have another heuristic.This heuristic function is as follows:

$$h_2(F) = \sum_{j \in F} C'_n(j) \tag{18}$$

where F is set of unallocated items

$$C'_{n-1}(i_k) = max_{i_1,i_2,....i_m} C_{n-1}(i_j), j = 1, 2, ...m \tag{19}$$

$$C'_n(j) = max_{q \in F_{n-1}}(max_{(q \in F_{n-1} | i_k \in S_{n-1})} \frac{(\bar{b}_{n-1}(S_{n-1}) - C'_{n-1}(i_k)}{| S_{n-1} | -1}, \tag{20}$$

$$max_{(q \in F_{n-1} | (i_k \notin S_{n-1})}(C_{n-1}(q))) \tag{21}$$

$$where \qquad C_{n-1}(q) = max_{q \in S_{n-1}} \frac{\bar{b}_{n-1}(S_{n-1})}{| S_{n-1} |} \tag{22}$$

where F_{n-1} is set of maximum posssible items in $(n-1)$th iteration,

S_{n-1} is of set of items in $(n-1)$th iteration.

Proposition 3.6: If any item k having maximum average value bid is not in winning solution and maximum average value of k is reduced from bids having item k then $F' = \{i | i \neq k, i \in F\}$.

$$v(k) + \sum_{i \in F'} c'(i) \leq \sum_{i \in F} c(i) \tag{23}$$

where,$v(k)$ is maximum average value for an item k,$c'(i)$ is the maximum average value of an item by reducing value of bid having item k by $v(k)$. $c(i)$ is the maximum average value of an item from initial bid combination.

Proof: Proof by contradiction,we assume that LHS is greater than RHS.So If item k maximum average value bid is not in winning combination then it will reduce the average of those items that has in winning bid with item k.So,for items i in a bid having item k,
$F' = \{j | j \neq k, j \in F\}$,
$c^{'}(i) < c(i)$,rest other items unaffected so
$\forall j \in F', c^{'}(j) \leq c(j)$

$$v(k) + \sum_{j \in F'} c'(j) \leq \sum_{j \in F} c(j) \tag{24}$$

It contradicts our assumption that $v(k) + \sum_{j \in F'} c'(j) > \sum_{j \in F} c(j)$.□

Proposition 3.7 Heuristic 3.2 is admissible heuristic(it never underestimate the revenue from unallocated items) in winner determination problem

Proof:For any set $S \in F$. It has two cases
1. If all the bids in winning solution having maximum average value for items and
2. If any of the the maximum average valued bid for an item is not in winning solution.

case 1: If all bids in winning solution,by reducing the maximum average value for an item from a bid reduce the value of a bid by its average from that bid so the average value of item in that is not reduced so next maximum item will come from that bid so bid value is always reduced by its average value so that sum of all items maximum average is not going to reduce so sum is simply $\sum_{S|i \in S}(max_{S|i \in S})\frac{(\bar{b}(S)}{|S|}, \forall\ i \in F$

case 2: If any of the maximum average valued bid for an item is not

in winning solution. we take an element k , its bid corresponding to maximum average value is not in winning solution. So by reducing all the bids having that item k by its maixmum average value. So from proposition 3.6. iteratively it can be shown that $h_2(F)$ is not greater than $h_1(F)$.□

11.4 Comparative heuristic estimate

In this section we will study about how different heuristic affect performance. Performance of heuristics on algorithm depends on node generation,nodes expansion and data structure.To study the impact of two different heuristics we have taken the algorithm is same in both cases in every aspect of implementation even similar kind of data structure is used in both cases. The motivation behind study is that we want to study the impact on performance in case of admissible heuristic where heuristic $h1$ is less computational intensive than heuristic $h2$.Heuristic $h2$ is giving more tight bound. Heuristic $h2$ is giving tighter upper bound so we are expecting that it should do more pruning by less node expansion such that less node generation. We have done so many experiment to investigate actual impact of both heuristic in winner determination in combinatorial Auction. We illustrate from next example that heuristic $h2$ is giving tighter upper bound than $h1$

Let us take a set of bids

bids	bid value
{1,2}	2
{1,2,3}	7
{2,3}	4
{3,4}	8

In heuristic $h1$, the estimate of an item is maximum average of each item in set of bids in which item is coming So

$h_1(F) = (7/3(1) + 7/3(2) + 4(3) + 4(4)) = 12.66$ estimate of heuristic $h1$

In heuristic $h2$,the estimate of item is maximum of maximum average of items in set of bids,so in 1st iteration maximum of maximum average of item is 4 for item 3 & 4 for item 4(ties can be broken arbitrarily). So we have taken item 3(for low index) before next iteration bid has changed as following

bids	bid value
{1,2}	2
{1,2}	$7 - 4 = 3$
{2}	$4 - 4 = 0$
{4}	$8 - 4 = 4$

So in next iteration we have to find estimate for another item, in this iteration we had found item 4, estimate is 4 so bid has changed by

bids	bid value
{1,2}	2
{1,2}	3
{2}	0

So in further iteration we find item 1&2 both have same estimate 1.5.Ties can be broken(arbitrarily) here I have taken in favour of lower index. So estimate for 1 is 1.5. Again bid will change accordingly as

bids	bid value
{2}	$2 - 2.5 = -0.5 \rightarrow 0$ from definition of heuristic $h2$
{2}	$3 - 1.5 = 1.5$

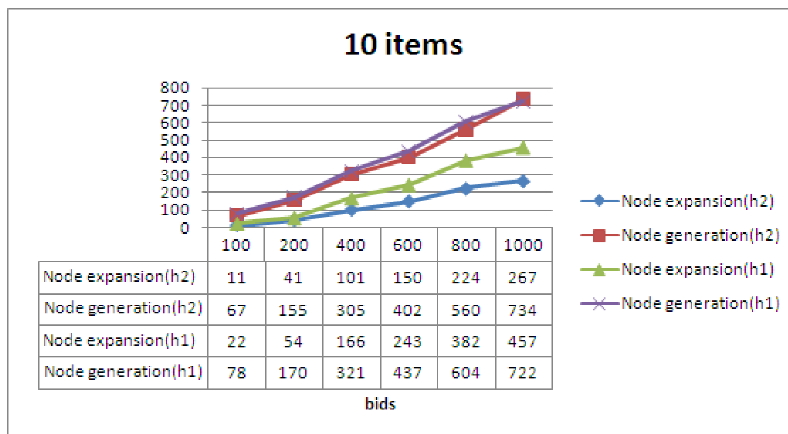
So $h_2(F) = 4(3) + 4(4) + 1.5(1) + 1.5(2) = 11$,estimate of heuristic $h2$

Actual solution is {1,2} & {3,4} and revenue is (2+8)=10,So heuristic estimate for heuristic $h2$ is much closer to the solution than heuristic $h1$.

11.5 Experimental setup

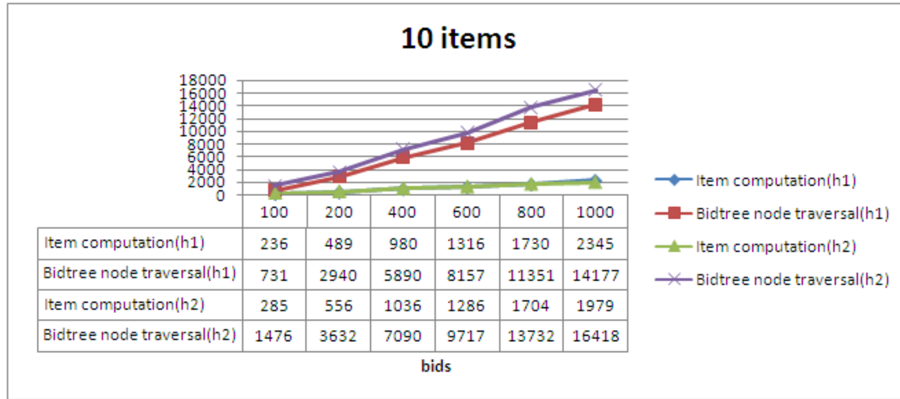
To understand the real impact of heuristic $h1$ and heuristic $h2$, we have set an experiment. In which we have used value added tree & ITEM TREE data structure and also used incremental heuristic computation. We have used the weighted random distribution and all the experiment is done on core 2 duo processor with 2 Ghz and 1 GB RAM in C with Weighted random bid distributions. Item size 10, 20, 30, 40 and 50 for bid size 100, 200, 400, 600, 800 and 1000 run 50 times for each item size and bid size combination & had taken average value for each combination.

11.6 Experimantal result



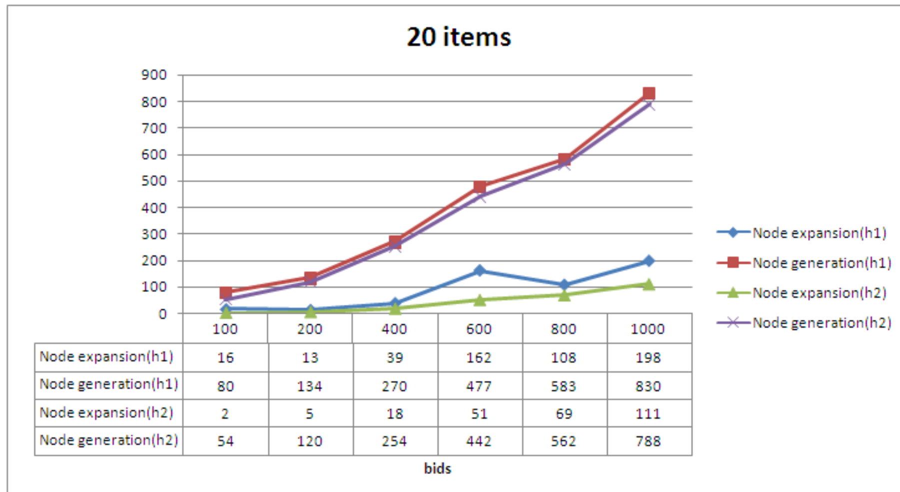
	100	200	400	600	800	1000
Node expansion(h2)	11	41	101	150	224	267
Node generation(h2)	67	155	305	402	560	734
Node expansion(h1)	22	54	166	243	382	457
Node generation(h1)	78	170	321	437	604	722

Figure 5.1



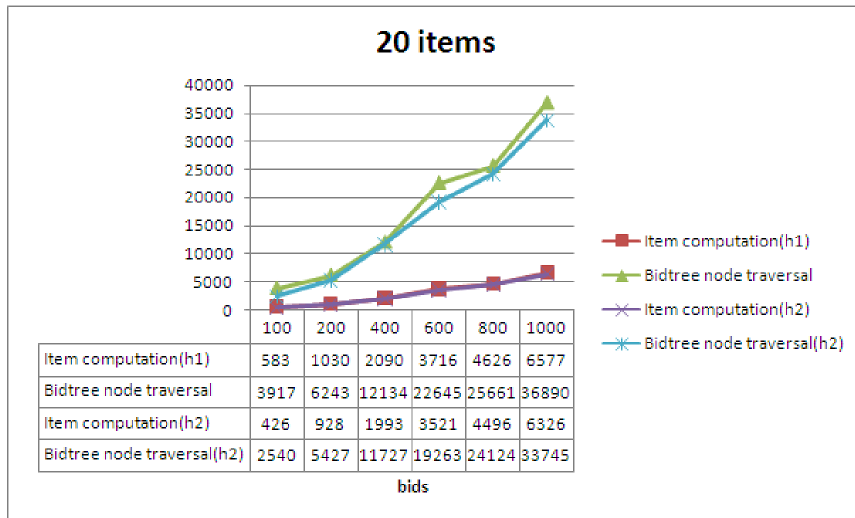
	100	200	400	600	800	1000
Item computation(h1)	236	489	980	1316	1730	2345
Bidtree node traversal(h1)	731	2940	5890	8157	11351	14177
Item computation(h2)	285	556	1036	1286	1704	1979
Bidtree node traversal(h2)	1476	3632	7090	9717	13732	16418

Figure 5.2



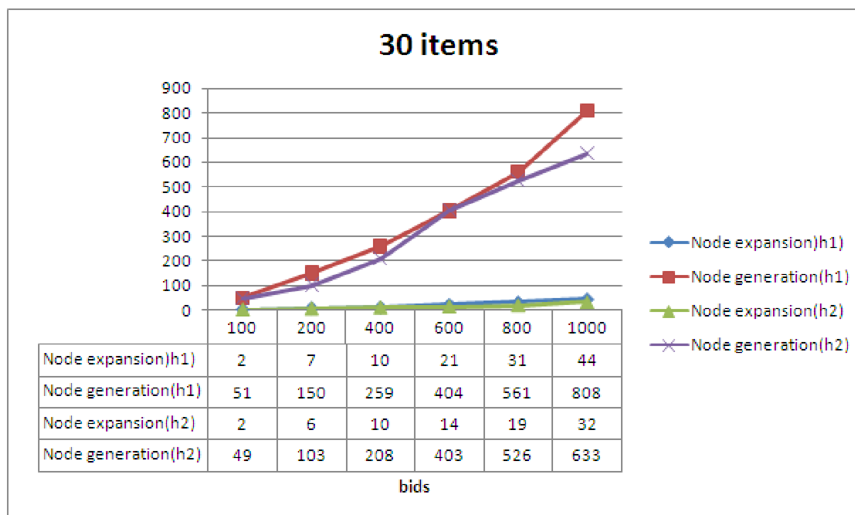
	100	200	400	600	800	1000
Node expansion(h1)	16	13	39	162	108	198
Node generation(h1)	80	134	270	477	583	830
Node expansion(h2)	2	5	18	51	69	111
Node generation(h2)	54	120	254	442	562	788

Figure 5.3



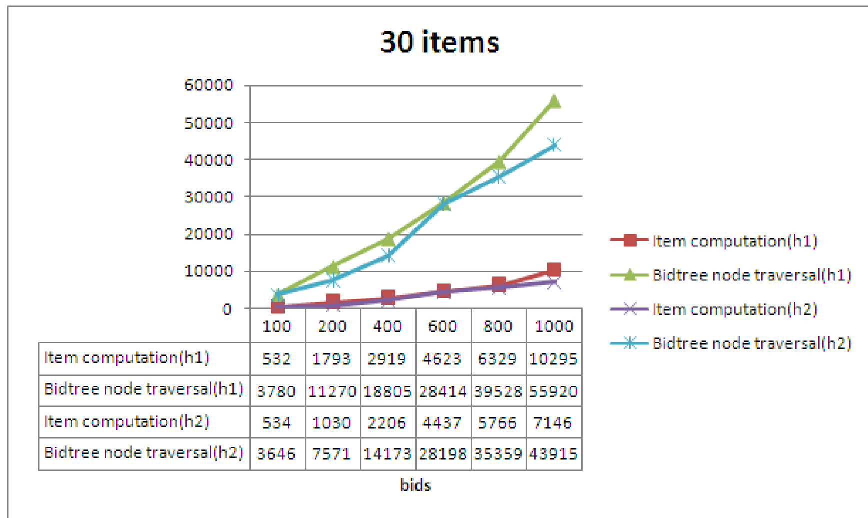
	100	200	400	600	800	1000
Item computation(h1)	583	1030	2090	3716	4626	6577
Bidtree node traversal	3917	6243	12134	22645	25661	36890
Item computation(h2)	426	928	1993	3521	4496	6326
Bidtree node traversal(h2)	2540	5427	11727	19263	24124	33745

Figure 5.4



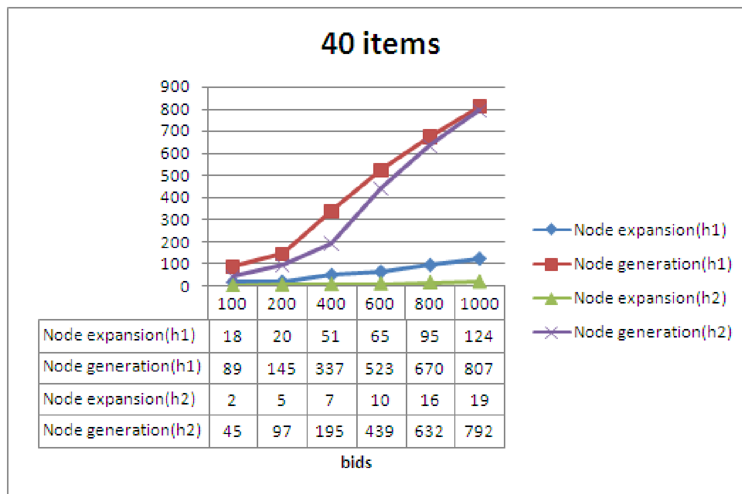
	100	200	400	600	800	1000
Node expansion)h1)	2	7	10	21	31	44
Node generation(h1)	51	150	259	404	561	808
Node expansion(h2)	2	6	10	14	19	32
Node generation(h2)	49	103	208	403	526	633

Figure 5.5



	100	200	400	600	800	1000
Item computation(h1)	532	1793	2919	4623	6329	10295
Bidtree node traversal(h1)	3780	11270	18805	28414	39528	55920
Item computation(h2)	534	1030	2206	4437	5766	7146
Bidtree node traversal(h2)	3646	7571	14173	28198	35359	43915

Figure 5.6



	100	200	400	600	800	1000
Node expansion(h1)	18	20	51	65	95	124
Node generation(h1)	89	145	337	523	670	807
Node expansion(h2)	2	5	7	10	16	19
Node generation(h2)	45	97	195	439	632	792

Figure 5.7

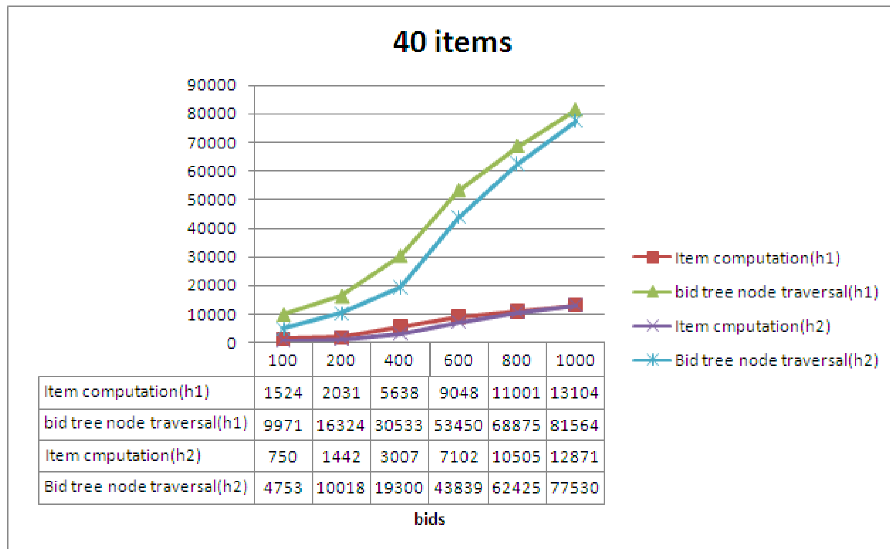


Figure 5.8

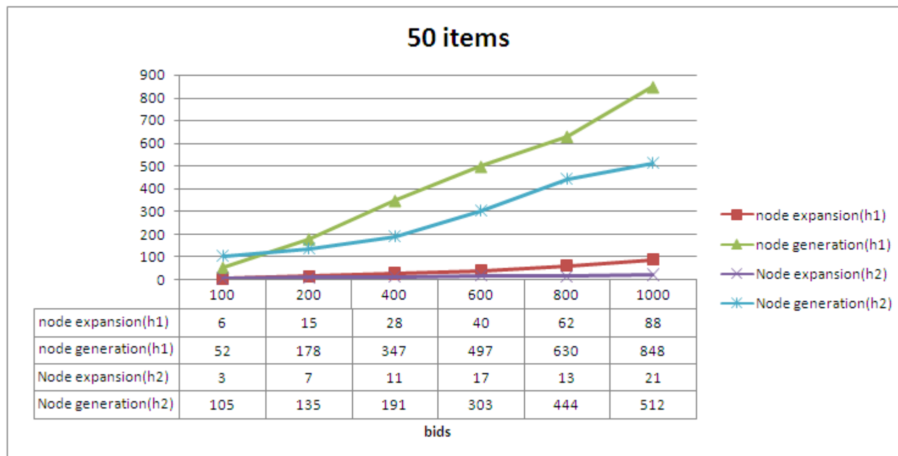
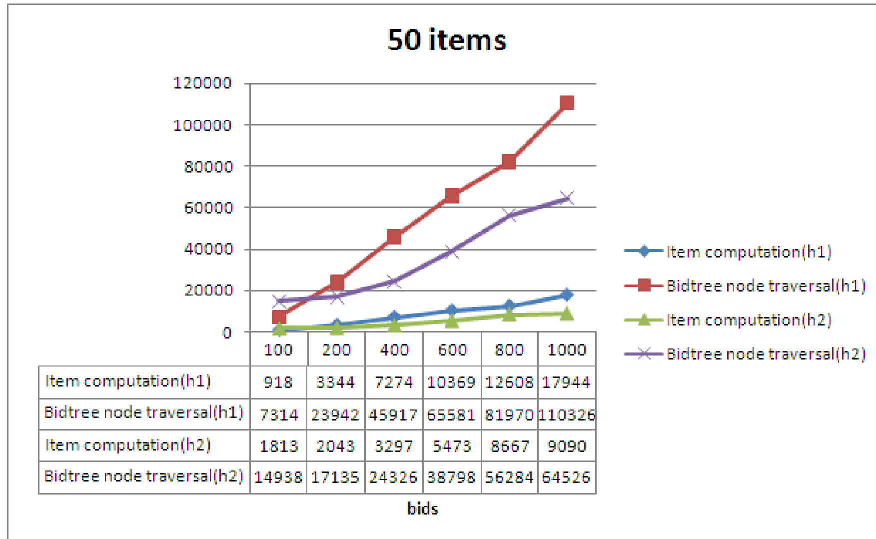


Figure 5.9

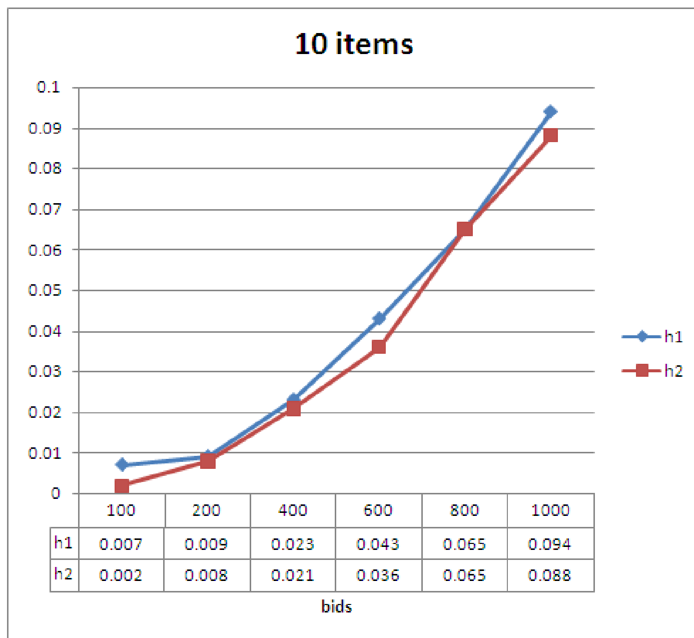


	100	200	400	600	800	1000
Item computation(h1)	918	3344	7274	10369	12608	17944
Bidtree node traversal(h1)	7314	23942	45917	65581	81970	110326
Item computation(h2)	1813	2043	3297	5473	8667	9090
Bidtree node traversal(h2)	14938	17135	24326	38798	56284	64526

Figure 5.10

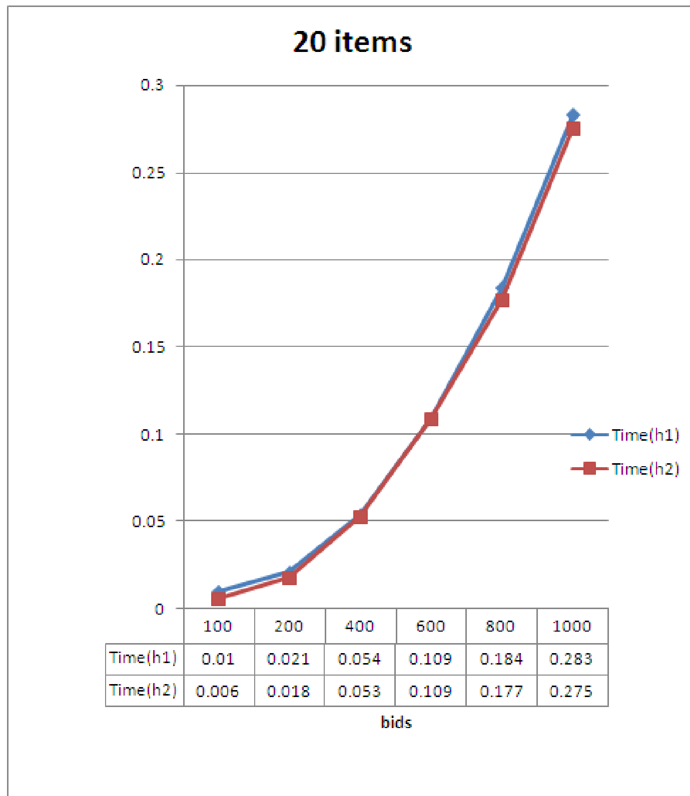
In this section we will analyse the result of experimentation. We had seen that computation in heuristic($h2$) is costlier as comparison to heuristic($h1$). The total time complexity of heuristic $h1$ estimate is $O(mn')$ whereas the total time complexity of heuristic $h2$ estimate is $O(mn' + m^2)$(n' is greatest number of bids corresponding to an item,m is maximum posssible items in a bid) for addition of a node in MAIN SEARCH. So heuristic $h1$ is computed significantly faster as comparison to heuristic $h2$, but the main advantage of heuristic $h2$ over $h1$ is less node generation and less node expansion which is main cause for overall perforamance of algorithm. So to experimentally study the effect of tighter bound heuristuic $h2$,we have done several experiments on various bid size(maximum possible bids) and various item size(maximum possible items in a bid).

We have taken weighted random distribution for bid creation. From Figure 5.1 ,5.3, 5.5, 5.7, 5.9 we see that node expansion & node generation in heuristic $h2$ is lower as comparison to heuristic $h1$.It means that heuristic $h2$ is prunning more bids in depth search tree in MAIN SEARCH.Item computation & bidtree node traversal is also having difference between $h1$ and $h2$ but not so significant than node expansion,it means more pruning is done,but performance wise it is not giving any advantage over heuristic $h1$.We can see from plot for performance measurement in Figure 6.1, 6.2, 6.3, 6.4, and 6.5. We see that both heuristic is taking almost same time where as in heuristic $h2$ node expansion & node generation is less. So bottleneck is heuristic computation time in heuristic $h2$ .Due to costlier heuristic computation,it tradeoffs the advantages of heuristic $h2$ over $h1$.



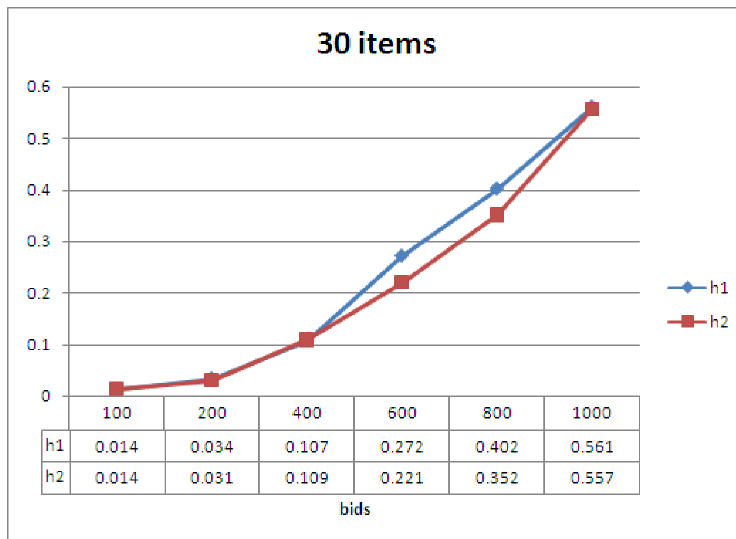
	100	200	400	600	800	1000
h1	0.007	0.009	0.023	0.043	0.065	0.094
h2	0.002	0.008	0.021	0.036	0.065	0.088

Figure 6.1



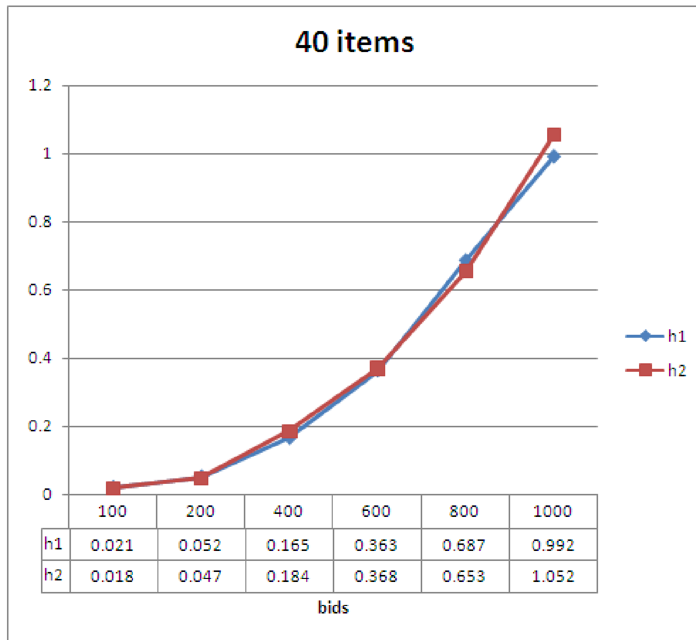
	100	200	400	600	800	1000
Time(h1)	0.01	0.021	0.054	0.109	0.184	0.283
Time(h2)	0.006	0.018	0.053	0.109	0.177	0.275

Figure 6.2



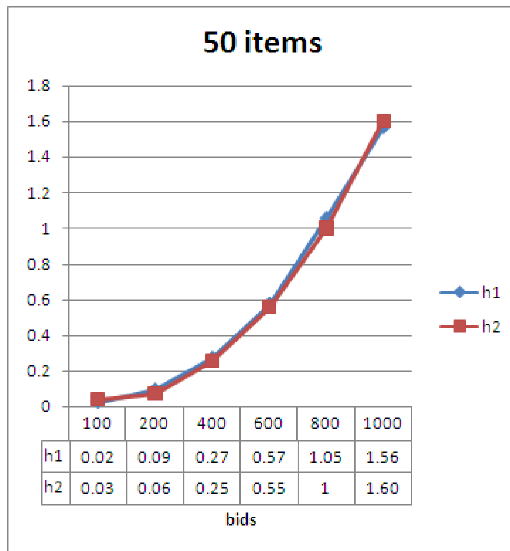
	100	200	400	600	800	1000
h1	0.014	0.034	0.107	0.272	0.402	0.561
h2	0.014	0.031	0.109	0.221	0.352	0.557

Figure 6.3



	100	200	400	600	800	1000
h1	0.021	0.052	0.165	0.363	0.687	0.992
h2	0.018	0.047	0.184	0.368	0.653	1.052

Figure 6.4



	100	200	400	600	800	1000
h1	0.02	0.09	0.27	0.57	1.05	1.56
h2	0.03	0.06	0.25	0.55	1	1.60

Figure 6.5

12 Conclusion

As we have seen that performance of heuristic depends not only on node generation & node expansion in winner determination in combinatorial auction but also on heuristic computation time. So for suggesting a heuristic function we should consider its time complexity also,but it does not mean that computationally intensive heuristic will not give better performance. As we have seen that heuristic $h2$ time complexity $O(m^2 + mn')$ is comparable in performance with heuristic $h1$ of time complexity $O(mn')$.So,if we will be able to reduce time complexity of heuristic $h2$ by $O(m^{2-\epsilon} + mn')$,for $\epsilon > 0$, it is possible that we can get better performance as compare to heuristic $h1$ in winner determination in Combination Auction.

References

[1] T. Sadholm, S. suri, A. Giplin, D. Levine, CABOB : A Fast optimal algorithm for winner determination in Combinatorial Auctions Vol. 51 , No. 3, March 2005, PP. 374-390 ISSN 0025-1999 — EISSN 1526-5501 — 05 — 5103 — 0374.

[2] T.Sadholm, S.suri, BOB: Improved winner determination in combinatorial auctions And generalisations 0004-3702/03/ @ 2003 Elsevier Science doi: 10.1016 / S0004 - 3702(03) 00015-8.

[3] T. Sadholm, Algorithm for optimal winner determination in combinatorial Auctions 0004-3702/01/2001 Elsevier science PII : S0004-3702(01)00159-X.

[4] J. Hastad, clique is hard to approximate within $n^{1-\epsilon}$, Act Math. 182(1999) 105-142.

[5] M . A. Weiss, Data structures and Algorithm Analysis in C, 2nd edition, Addison-Wesley, reading, MA, 1999.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to Algorithms MIT Press, Cambridge, MA, 1990.

[7] R. E. Korf, Depth-first iterative-deepening : An optimal admissible tree search, Artificial Intelligence 27 (1) (1985) 97109.

[8] M.H. Rothkopf, A. Peke, R. M. Harstad, Computationally manageable combinatorial auction, Management sci.44(8) (1998) 1131-1147.

[9] M. J. Osborne, An introduction to game theory, Indian edition, ISBN-13:978-0-19-512895-6.