Parallel Greedy Approximation on Large-Scale Combinatorial Auctions

Naoki Fukuta[1] and Takayuki Ito[2,3]
[1]*Shizuoka University,*
[2]*Nagoya Institute of Technology*
[3]*Massachusetts Institute of Technology*
[1,2]*Japan*
[3]*United States*

1. Introduction

Combinatorial auctions (Cramton et al., 2006) are auctions that allow bidders to place bids for a set of items. Combinatorial auctions provide suitable mechanisms for efficient allocation of resources to self-interested attendees (Cramton et al., 2006). Therefore, many works have been done to utilize combinatorial auction mechanisms for efficient resource allocation. For example, the FCC tried to employ combinatorial auction mechanisms for assigning spectrums to companies (McMillan, 1994).

On the other hand, efficient resource allocation is also becoming crucial in many computer systems that should manage resources efficiently, and combinatorial auction mechanisms are suitable for this situation. For example, considering a ubiquitous computing scenario, there is typically a limited amount of resources (sensors, devices, etc.) that may not cover all needs for all users. Due to certain reasons (physical limitations, privacy, etc.), most of the resources cannot be shared with other users. Furthermore, software agents will use two or more resources at a time to achieve desirable services for users. Of course, each software agent provides services to its own user, and the agent may be self-interested.

Tremendous research efforts have been done to improve many parts of combinatorial auctions. An example is recent efforts for winner determination problem. In general, the optimal winner determination problem of a combinatorial auction is NP-hard (Cramton et al., 2006) for the number of bids. Thus, much work focuses on tackling the computational costs for winner determination (Fujishima et al., 1999); (Cramton et al., 2006); (Sandholm et al., 2005). Also many efforts have been done for generic problem solvers that can be applied to solve winner determination problems.

However, in such ubiquitous computing scenarios, there is strong demand for completing an auction within a fine-grained time period without loss of allocation efficiency. In a ubiquitous computing scenario, the physical location of users may always be changing and that could be handled by the system. Also, each user may have multiple goals with different contexts, and those contexts are also dynamically changing. Therefore, resources should be re-allocated in a certain fine-grained period to keep up with those changes in a timely manner. For better usability, the time period of resource reallocation will be 0.1 to several

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

seconds depending on services provided there. Otherwise, resources will remain assigned to users who no longer need them while other users are waiting for allocation.
Also, in the above scenarios, it is very important to handle a large number of bids in an auction. Consider that if there are 256 resources and 100 agents, and each agent has 200 to 1000 bids, then there will be 20,000 to 100,000 bids for 256 items in an auction. However, it has been difficult to complete such a large-scale combinatorial auction within a very short time. Such hard time constraint even prevents algorithms to prepare a rich pre-processing to reach optimal results in (not very) short time.
Since greedy algorithm is so simple, it can be applied to such situations. However, a pure greedy algorithm typically provides lower optimality of results that are not satisfiable for applications. When we solve this issue, parallel greedy approach can be a good solution for this kind of problems. Furthermore, a simple greedy algorithm can be used to enforce results to satisfy desirable properties that are very important for both theoretical and practical reasons.
In this chapter, we describe how greedy algorithms can be effectively used in mechanism design, especially, on designing and implementing combinatorial auction mechanisms.

2. Combinatorial auctions and winner determination problem

2.1 Mechanism design and combinatorial auctions

An auction mechanism is an economic mechanism for efficient allocations of items to self-interested buyers with agreeable prices. When the auction mechanism is truthful, i.e., it guarantees incentive compatibility, the mechanism enforces the bidders to locate their bids with true valuations. In such auctions, since we have an expectation of obtaining bids with true valuations, we can allocate items to buyers efficiently even though some buyers may try to cheat the mechanisms out of gaining sufficient incomes from them. For example, Vickrey proposed an auction mechanism that has incentive compatibility (Vickrey, 1961). That is a basic difference from ordinary resource allocation mechanisms that have implicit assumptions of truth-telling attendees.
Combinatorial auction is an auction mechanism that allows bidders to locate bids for a bundle of items rather than single item (Cramton et al., 2006). Combinatorial auction has been applied for various resource allocation problems. For example, McMillan et al. reported a trial on an FCC spectrum auction (McMillan, 1994). Rassenti et al. reported a mechanism for an airport time slot allocation problem (Rassenti et al., 1982). Ball et al. discussed applicability of combinatorial auctions to airspace system resource allocations (Ball et al., 2006). Caplice et al. proposed a bidding language for optimization of procurement on freight transportation services (Caplice et al., 2004). Estelle et al. proposed a formalization on auctioning London Bus Routes (Cantillon & Pesendorfer, 2004). Hohner et al. presented an experience on procurement auctions at a software company (Hohner et al., 2003).
However, on emerging applications with such resource allocation problems, their problem spaces are larger, more complex, and much harder to solve compared to previously proposed applications. For example, Orthogonal Frequency Division Multiple Access (OFDMA) technology enables us to use a physically identical frequency bandwidth as virtually multiplied channels at the same time, and this causes the channel allocation problem to become more difficult (Yang & Manivannan, 2005). Also some recent wireless technologies allow us to use multiple channels on the same, or different physical layers (i.e, WiFi, WiMax, and Bluetooth at the same time) for attaining both peak speed and robust connectivity (Salem et al., 2006); (Niyato and Hossain, 2008). Furthermore, such resource

allocation should be done for many ordinary users rather than a fixed limited number of flights or companies. Also the contexts of users, which are dynamically changing through the time, should be considered in the allocation.
In this chapter, to maintain simplicity of discussion, we focus on utility-based resource allocation problems such as (Thomadakis & Liu, 1999), rather than generic resource allocation problems with numerous complex constraints. The utility-based resource allocation problem is a problem that aims to maximize the sum of utilities of users for each allocation period, but does not consider other factors and constraints (i.e., fair allocation (Sabrina et al., 2007); (Andrew et al., 2008), security and privacy concerns (Xie & Qin, 2007), uncertainty (Xiao et al., 2004), etc).
Also, throughout this chapter, we only consider auctions that are single-sided, with a single seller and multiple buyers to maintain simplicity of discussion. It can be extended to the reverse situation with a single buyer and multiple sellers, and the two-sided case. The two-sided case is known as the combinatorial exchange. In the combinatorial exchange mechanisms, multiple sellers and multiple buyers are trading on a single trading mechanism. About this mechanism, the process of determining winners is almost the same as single-sided combinatorial auctions. However, it is reported that the revenue division among sellers can be a problem. There are a lot of interesting studies on combinatorial exchange (Parkes et al, 2005).

2.2 Winner determination problem

An important issue on combinatorial auction is representation of bids. In this chapter, we use OR bid representation(Lehmann et al., 2006), a simplest one in major formalisms.
On OR bid representation, the winner determination problem on combinatorial auction WDP_{OR} is defined as follows (Cramton et al., 2006): The set of bidders is denoted by $N=\{1,...,n\}$, and the set of items by $M=\{m_1,...,m_k\}$. $|M|=k$. Bundle S is a set of items: $S \subseteq M$. We denote by $v_i(S)$, bidder i's valuation of the combinatorial bid for bundle S. An allocation of the items is described by variables $x_i(S) \in \{0,1\}$, where $x_i(S)=1$ if and only if bidder i wins bundle S. An allocation, $x_i(S)$, is feasible if it allocates no item more than once,

$$\sum_{i \in N} \sum_{S \ni j} x_i(S) \leq 1$$

for all $j \in M$.
The winner determination problem is the problem to maximize total revenue

$$\max_X \sum_{i \in N, S \subseteq M} v_i(S) x_i(S)$$

for feasible allocations $X \ni x_i(S)$.
Fig. 1 shows an example of WDP_{OR}. Consider there are three items *a*, *b*, and *c*, and three bidders *Alice*, *Bob*, and *Charles*. *Alice* bids 10 for *a*. *Bob* bids 20 for {*b*, *c*}. *Charles* bids 18 for {*a*, *b*}. The problem is to choose winners of this auction from those three bids. Here, to choose *Alice*'s and *Charles*'s, or *Bob*'s and *Charles*'s are infeasible allocation, since both *Alice*'s and *Charles*'s include item *a*, and both *Bob*'s and *Charles*'s include item *b*. The optimal allocation is *a* for *Alice*, and *b* and *c* for *Bob*.

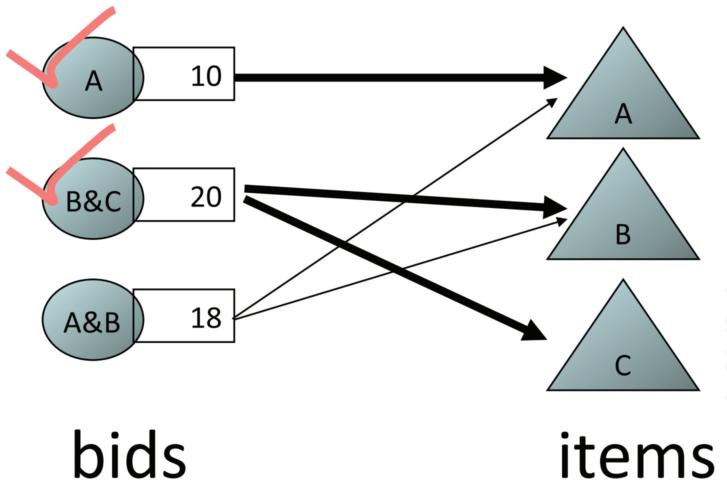


Fig. 1. Winner Determination Problem

Since the winner determination problem WDP_{OR} is a combinatorial optimization problem, it is generally NP-hard(Cramton et al., 2006). Furthermore, winner determination also plays important roles in other parts of combinatorial auction mechanism. For example, some combinatorial auction mechanisms (e.g., VCG, etc.) require many times of winner determination for slightly different bids for pricing mechanism. Therefore, it is strongly demanded to solve the problem in tractable way. In this chapter, we focus on solving this problem.

2.3 Lehmann's greedy winner determination

Lehmann et al. proposed a combinatorial auction mechanism that preserves truthfulness, a very important desirable property, while it uses a greedy approximation algorithm for its winner determination(Lehmann et al., 2002).

Lehmann's greedy algorithm (Lehmann et al., 2002) is a very simple but powerful linear algorithm for winner determination in combinatorial auctions. Here, we denote a bid b=<s,a>, such that $S \subseteq M$ and $a \in \mathcal{R}_+$. Two bids b=<s,a> and b'=<s',a'> conflict if and only if $s \cap s' \neq \emptyset$. The greedy algorithm can be described as follows. (1) The bids are sorted by some criterion. In (Lehmann et al., 2002), Lehmann et al. proposed sorting list L by descending average amount per item. More generally, they proposed sorting L by a criterion of the form $a/|s|^c$ for some number $c \geq 0$, possibly depending on the number of items, k. (2) A greedy algorithm generates an allocation. L is the sorted list in the first phase. Walk down the list L, allocates items to bids whose items are still unallocated.

Example: Assume there are three items a, b, and c, and three bidders *Alice*, *Bob*, and *Charles*. *Alice* bids 10 for a. *Bob* bids 20 for $\{b,c\}$. *Charles* bids 18 for $\{a,b\}$ (Fig. 2 Step1). We sort the bids by the criterion of the form $a/|s|^{0.5}$ (Fig. 2 Step2). *Alice*'s bid is calculated as $10/1^{0.5}$=10. *Bob*'s bid is calculated as $20/2^{0.5}$=14 (approximately). *Charles*'s bid is calculated as $18/2^{0.5}$=13 (approximately). The sorted list is now *Bob*'s bid <$\{b,c\}$,20>, *Charles*'s bid <$\{a,b\}$,18>, and *Alice*'s bid <$\{a\}$, 10>. The algorithm walks down the list (Fig. 2 Step3). At first, *Bob* wins $\{b,c\}$ for 20. Then, *Charles* cannot get the item because his bid conflicts with *Bob*'s bid. Finally, *Alice* gets $\{a\}$ for 10.

Lehmann's greedy algorithm provides a computationally tractable combinatorial auction. However, it has two remaining issues: (1)efficiency of item assignment, and (2)adjustment of good bid weighting parameter c. In the next section, we describe possible approaches for these issues.

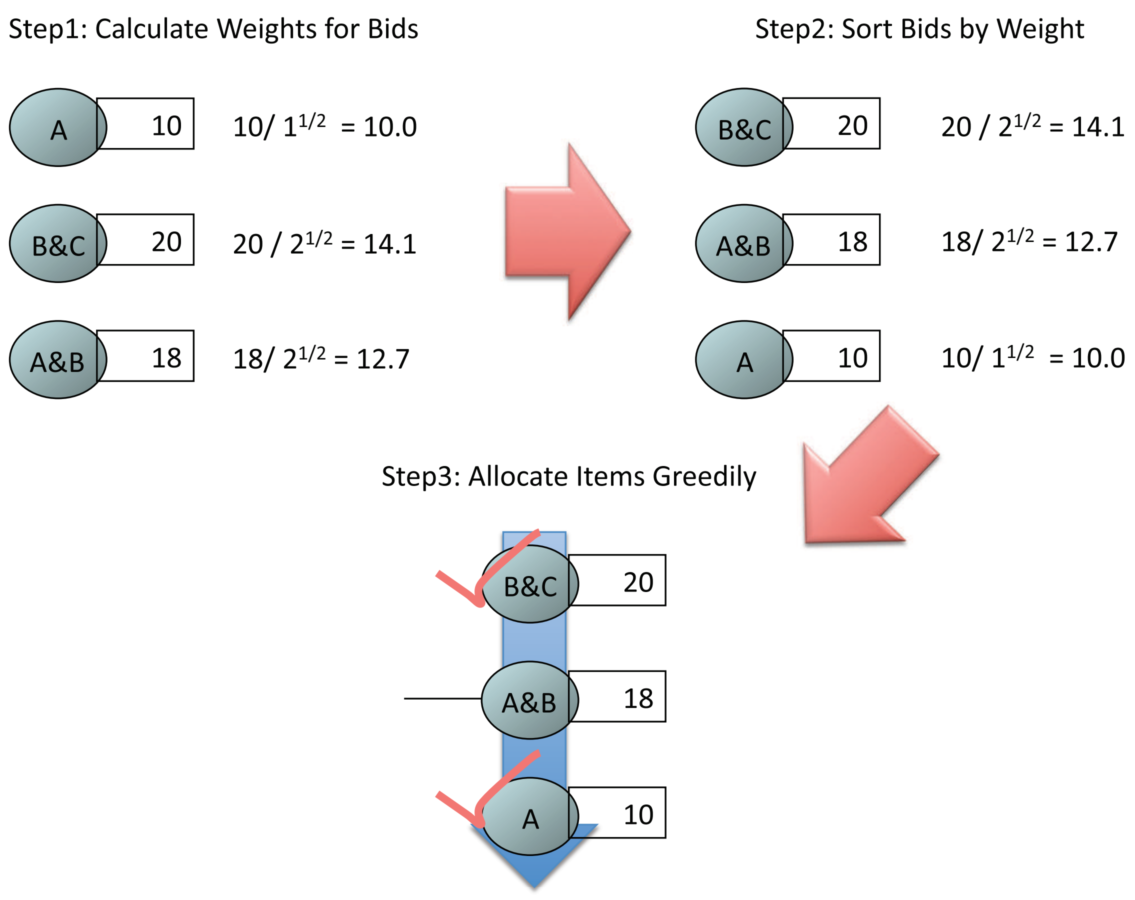

Step1: Calculate Weights for Bids
A 10 $10/ 1^{1/2} = 10.0$
B&C 20 $20 / 2^{1/2} = 14.1$
A&B 18 $18/ 2^{1/2} = 12.7$
Step2: Sort Bids by Weight
B&C 20 $20 / 2^{1/2} = 14.1$
A&B 18 $18/ 2^{1/2} = 12.7$
A 10 $10/ 1^{1/2} = 10.0$
Step3: Allocate Items Greedily
B&C 20
A&B 18
A 10

Fig. 2. Lehmann’s Greedy Allocation

3. Parallel greedy approximation

3.1 Incremental updating

In (Fukuta & Ito, 2006), we have shown that the hill-climbing approach performs well when an auction has a massively large number of bids. In this section, we summarize our proposed algorithms for incremental updating solutions.

Lehmann's greedy winner determination could succeed in specifying the lower bound of the optimality in its allocation (Lehmann et al., 2002). A straightforward extension of the greedy algorithm is to construct a local search algorithm that continuously updates the allocation so that the optimality is increased. Intuitively, one allocation corresponds to one state of a local search.

List 1 shows the algorithm. The inputs are *Alloc* and *L*. *L* is the bid list of an auction. *Alloc* is the initial greedy allocation of items for the bid list.

The function ***consistentBids*** finds consistent bids for the set ***NewAlloc*** by walking down the list ***RemainBids***. Here, a new inserted bid will wipe out some bids that conflict with the inserted bid. So there will be free items to allocate after the insertion. The function ***consistentBids*** tries to insert the other bids greedily for selling as many of the items as possible. When the total price for ***NewAlloc*** is higher than ***Alloc***, current allocation is updated to ***NewAlloc*** and the function continues updating from ***NewAlloc***. We call this as *Greedy Hill Climbing*(GHC) in this chapter.

```
function GreedyHillClimbingSearch($Alloc$, $L$)
  $RemainBids$:= $L$ - $Alloc$;
  for each $b \in RemainBids$ as sorted order
    if $b$ conflicts $Alloc$ then
      $Conflicted$:=$Alloc$ - $consistentBids(\{b\}, Alloc)$;
      $NewAlloc$:= $Alloc$ - $Conflicted$ + $\{b\}$;
      $ConsBids$:=
         consistentBids($NewAlloc$, $RemainBids$);
      $NewAlloc$:=$NewAlloc$+$ConsBids$;
    if $price(Alloc) < price(NewAlloc)$ then
      return GreedyHillClimbingSearch($NewAlloc$,$L$);
  end for each
  return $Alloc$
```

List. 1. Greedy Hill Climbing Algorithm

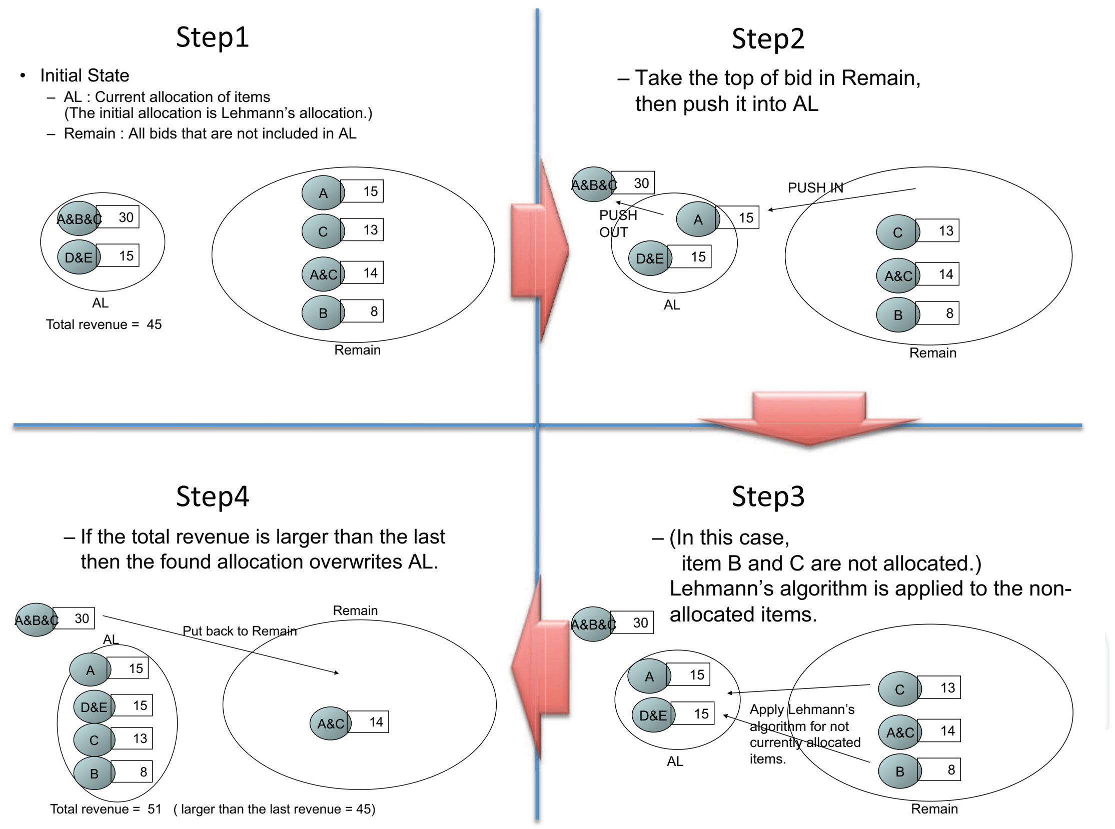

Step1
• Initial State
– AL : Current allocation of items (The initial allocation is Lehmann's allocation.)
– Remain : All bids that are not included in AL
A&B&C 30
D&E 15
AL
Total revenue = 45
A 15
C 13
A&C 14
B 8
Remain
Step2
– Take the top of bid in Remain, then push it into AL
A&B&C 30
PUSH OUT
PUSH IN
A 15
D&E 15
AL
C 13
A&C 14
B 8
Remain
Step3
– (In this case, item B and C are not allocated.) Lehmann's algorithm is applied to the non-allocated items.
A&B&C 30
A 15
D&E 15
AL
Apply Lehmann's algorithm for not currently allocated items.
C 13
A&C 14
B 8
Remain
Step4
– If the total revenue is larger than the last then the found allocation overwrites AL.
A&B&C 30
Put back to Remain
Remain
AL
A 15
D&E 15
C 13
B 8
A&C 14
Total revenue = 51 (larger than the last revenue = 45)

Fig. 3. Example of Greedy Hill Climbing

Example: Assume there are five items *a*, *b*, *c*, *d*, and *e*, and there are six bids, <{*a,b,c*},30>, <{*a*},15>, <{*c*},13>, <{*d,e*},15>, <{*a,c*},14>, and <{*b*},8>. We can calculate the values of Lehmann's criterion $a/|s|^{0.5}$ as 17.6, 15, 13, 10.7, 10, and 8, respectively. In this case, the initial allocation is Lehmann's greedy allocation <{*a,b,c*},30>, <{*d,e*},15> and the total revenue is 45. Here, the remaining list contains <{*a*},15>, <{*c*},13>, <{*a,c*},14>, and <{*b*},8> (Fig. 3,

Step1). In this algorithm, we pick <{*a*},15> since it is the top of the remaining list. Then we insert <{*a*},15> into the allocation and remove <{*a,b,c*},30>. The allocation is now <{*a*},15>, <{*d,e*},15> (Fig. 3, Step2). We then try to insert the other bids that do not conflict with the allocation (Fig. 3, Step3). Then, the allocation becomes <{*a*},15>, <{*b*},8>, <{*c*},13>,<{*d,e*},15>. The total revenue is 51, and is increased. Thus, the allocation is updated to it (Fig. 3, Step4). Our local algorithm continues to update the allocation until there is no allocation that has greater revenue. This could improve the revenue that Lehmann's greedy allocation can achieve.
To show the advantages of greedy incremental updating, we also prepared an ordinary Hill-Climbing local search algorithm. List.2. shows the algorithm. The difference to above is to choose *best* alternatives in each climbing step, instead of choosing it greedily. We call this as *Best Hill Climbing*(BHC) in this chapter.

```
**function** BestHillClimbingSearch($Alloc$, $L$)
  $MaxAlloc := \phi$
  $RemainBids$:= $L$ - $Alloc$;
  **for each** $b \in RemainBids$ as sorted order
    **if** $b$ conflicts $Alloc$ **then**
      $Conflicted$:=$Alloc$ - $consistentBids(\{b\}, Alloc)$;
      $NewAlloc$:= $Alloc$ - $Conflicted$ + $\{b\}$;
      $ConsBids$:=
         consistentBids($NewAlloc$, $RemainBids$);
      $NewAlloc$:=$NewAlloc$+$ConsBids$;
    **if** $price(MaxAlloc) < price(NewAlloc)$ **then**
      $MaxAlloc := NewAlloc$;
  **end for each**
  **if** $price(Alloc) < price(MaxAlloc)$ **then**
    **return** BestHillClimbingSearch($MaxAlloc$,$L$);
  **return** $Alloc$
```

List. 2. Best Hill Climbing Algorithm

3.2 Parallel search for multiple weighting strategies

The optimality of allocations got by Lehmann's algorithm (and the following hill-climbing) deeply depends on which value was set to *c* in the bid weighting function. Again, in (Lehmann et al., 2002), Lehmann et al. argued that *c=1/2* is the best parameter for approximation when the norm of the worst case performance is considered. However, optimal value for approximating an auction is varied from 0 to 1 depending on the auction problem.
For example, when we choose *c=1* in the example in section 3.1, we can get better results directly at the time of initial Lehmann's greedy allocation (Fig. 4).
In (Fukuta & Ito, 2006), we presented an initial idea of an enhancement for our incremental updating algorithm to parallel search for different bid weighting strategies (e.g, doing the same algorithm for both *c=0* and *c=1*).

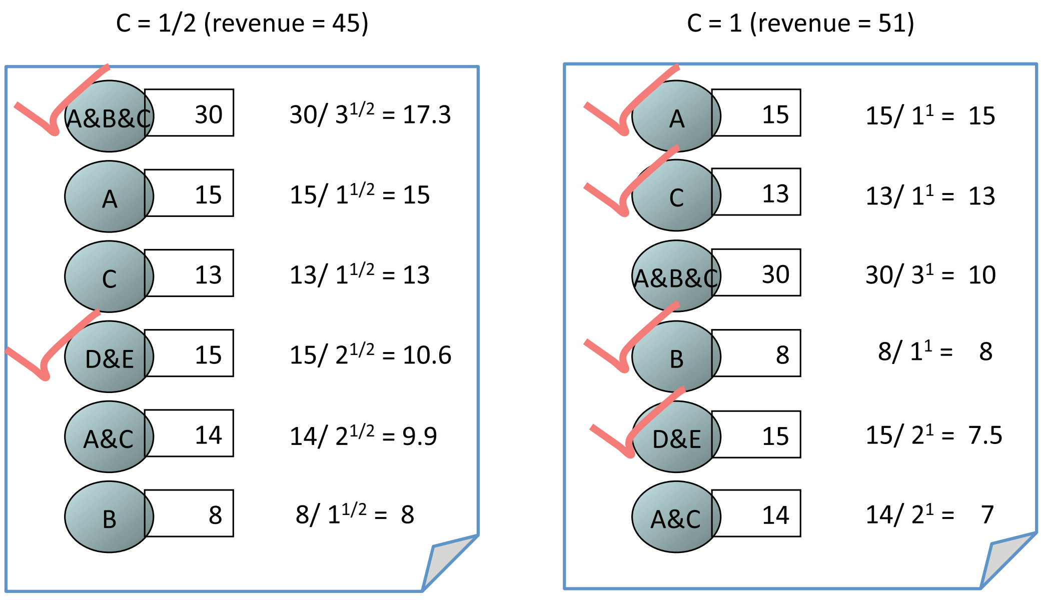


Fig. 4. Effects of Bid Weighting Strategy

3.3 Simulated annealing search

We also prepared a small extension of the shown algorithm to the simulated annealing local search(Fukuta & Ito, 2006). The algorithm is a combination of the presented hill-climbing approach and a random search based on the standard simulated annealing algorithm. We use a parameter that represents the temperature. The temperature is set at a high value at the beginning and continuously decreased until it reaches 0. For each cycle, a neighbour is randomly selected and its value may be less than the current value in some cases. Even in such a case, if a probability value based on the temperature is larger than 0, the state is moved to the new allocation that has less value. This could make us get off the local minimum.

We prepared this algorithm only for investigating how random search capability will improve the performance. Note that the proposed SA search may not satisfy our proposed features discussed later.

4. Experimental analysis

4.1 Experiment settings

In this section, we compare our algorithms to other approaches in various datasets. Details about other approaches are presented in section 5.

We implemented our algorithms in a C program for the following experiments. We also implemented the Casanova algorithm(Hoos & Boutilier, 2000) in a C program. However, for the following experiments, for Zurel's algorithm we used Zurel's C++ based implementation that is shown in (Zurel & Nisan, 2001). Also we used CPLEX Interactive Optimizer 11.0.0 (32bit) in our experiments.

The experiments were done with the above implementations to examine the performance differences among algorithms. The programs were employed on a Mac with Mac OS X 10.4, CoreDuo 2.0GHz CPU, and 2GBytes of memory. Thus, actual computation time will be much smaller when we employ parallel processor systems in a distributed execution environment.

We conducted several experiments. In each experiment, we compared the following search algorithms. greedy(c=0.5) uses Lehmann's greedy allocation algorithm with parameter (*c=0.5*). greedy-N uses the best results of Lehmann's greedy allocation algorithm for N different weighting parameters ($0 \leq c \leq 1$). *HC(c=0.5) uses a local search in which the initial allocation is Lehmann's allocation with *c=0.5* and conducts one of hill-climbing searchs (e.g., GHC or BHC) shown in the previous section. Similarly, *HC-N uses the best results of a hill-climbing search (e.g., GHC or BHC) for *N* different weighting parameters ($0 \leq c \leq 1$). For example, GHC-11 means the best result of greedy hill-climbing(GHC) with parameter c = {0, 0.1,...,0.9, 1}. SA uses the simulated annealing algorithm presented in (Fukuta & Ito, 2006). Also, we denote the Casanova algorithm as casanova and Zurel's algorithm as Zurel.
In the following experiments, we used 0.2 for the epsilon value of Zurel's algorithm in our experiments. This value appears in (Zurel & Nisan, 2001). Also, we used 0.5 for *np* and 0.15 for *wp* on Casanova, which appear in (Hoos & Boutilier, 2000). Note that we set *maxTrial* to 1 but *maxSteps* to ten times the number of bids in the auction.

4.2 Evaluation on basic auction dataset

In (Zurel & Nisan, 2001), Zurel et al. evaluated the performance of their presented algorithm with the data set presented in (de Vries & Vohra, 2003), compared with CPLEX and other existing implementations.
In (Fukuta & Ito, 2007a), we presented comparison of our algorithms, Casanova, and Zurel's algorithm with the dataset provided in (de Vries & Vohra, 2003). This dataset contains 2240 auctions with optimal values, ranging from 25 to 40 items and from 50 to 2000 bids. Since the data set is small, we omit details in this chapter.
We conducted detailed comparisons with common datasets from CATS benchmark(Leyton-Brown et al., 2000). Compared to deVries' dataset shown in (de Vries & Vohra, 2003), the CATS benchmark is very common and it contains more complex and larger datasets.
Fig. 5 shows the comparison of our algorithms, Casanova, and Zurel's algorithm with a dataset provided in the CATS benchmark (Leyton-Brown et al., 2000). The dataset has numerous auctions with optimal values in several distributions. Here we used ***varsize*** which contains a total of 7452 auctions with reliable optimal values in 9 different distributions[1]. Numbers of items range from 40 to 400 and numbers of bids range from 50 to 2000.
Since problems in the dataset have relatively small size of bids and items, we omitted the execution time since all algorithms run in very short time. Here, we can see that the performances of GHC-11 and SA are better than Zurel's on average optimality.
Note that those differences come from the differences of the termination condition on each algorithm. In particular, Casanova spent much more time compared with the other two algorithms. However, we do not show the time performance here since the total execution time is relatively too small to be compared.

[1] Since some of the original data seems corrupted or failed to obtain optimal values, we excluded such auction problems from our dataset. Also, we excluded a whole dataset of a specific bid distribution when the number of valid optimal values is smaller than the other half of the data. The original dataset provides optimal values of auction problems by two independent methods, CASS and CPLEX. Therefore, it is easy to find out such corrupted data from the dataset.

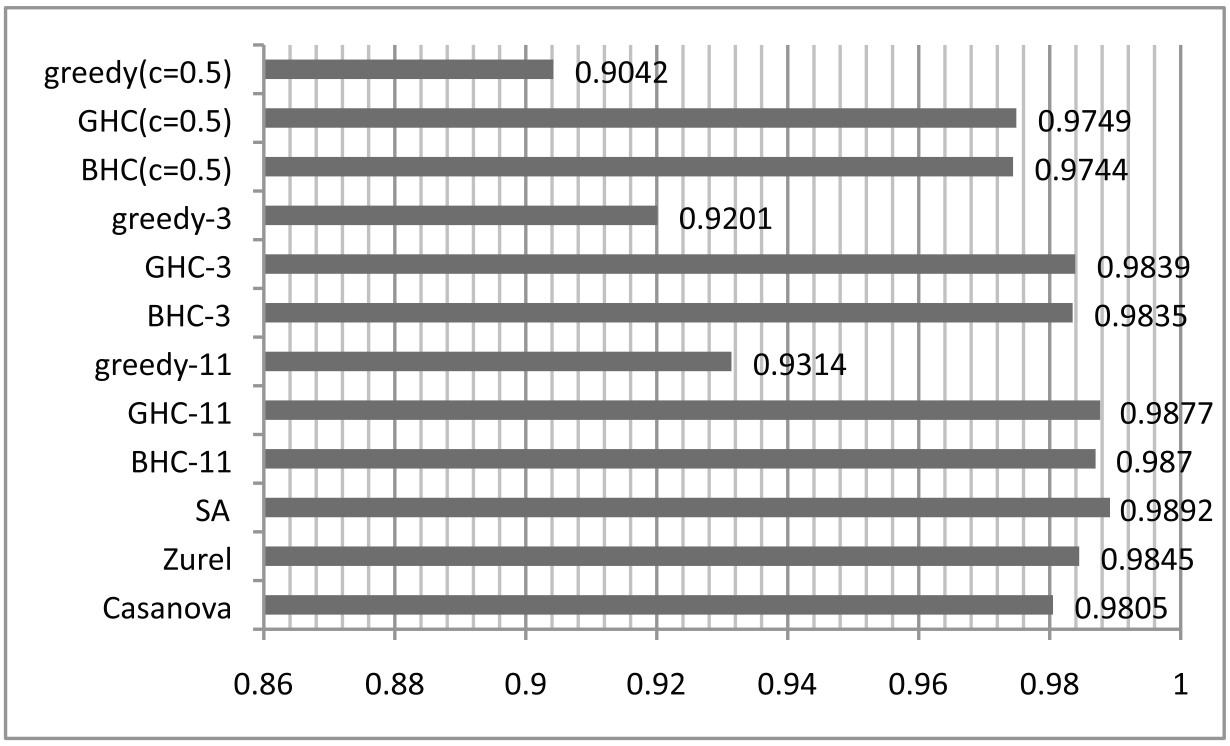


Fig. 5. Optimality on CATS-VARSIZE dataset

Here, we can see the performance of both greedy, GHC, and BHC increases when we use more threads to parallel search for multiple weightings. For example, the result of GHC-3 is better than GHC(c=0.5) and GHC-11 is slightly better in the average. It shows that our parallel approximation approach will increase the performance effectively even when the number of parallel executions is small.

Also we compared the performance on our greedy local updating approach (GHC) with ordinary best updating approach (BHC). Surprisingly, the average performances of GHC are slightly better than BHC, regardless of using parallel search. This is because the BHC approach is still heuristic one so it does not guarantee the choice is best for global optimization. Also we think we found a very good heuristic bid weighting function for our greedy updating.

4.3 Evaluation on large auction dataset

The CATS common datasets we used in Section 4.2 have a relatively smaller number of bids than we expected. We conducted additional experiments with much greater numbers of bids. We prepared additional datasets having 20,000 non-dominated bids in an auction. The datasets were produced by CATS (Leyton-Brown et al., 2000) with default parameters in 5 different distributions. In the datasets, we prepared 100 trials for each distribution. Each trial is an auction problem with 256 items and 20,000 bids[2].

[2] Due to the difficulty of preparing the dataset, we only prepared 5 distributions. For more details about the bid generation problem, see (Leyton-Brown et al., 2000). A preliminary result of this experiment was shown in (Fukuta & Ito, 2007b).

Fig. 6 (6a and 6b) shows the experimental result on the datasets with 20,000 bids in an auction focused on execution time of approximation. Due to the difficulty of attaining optimal values, we normalized all values as Zurel's results equaling 1 as follows.
Let A be a set of algorithms, $z \in A$ be the Zurel's approximation algorithm, L be a dataset generated for this experiment, and *revenue*$_a$*(p)* such that $a \in A$ be the revenue obtained by algorithm a for a problem p such that $p \in L$, the average revenue ratio *ratioA*$_a$*(L)* for algorithm $a \in A$ for dataset L is defined as follows:

$$ratioA_a(L) = \frac{\sum_{p \in L} revenue_a(p)}{\sum_{p \in L} revenue_z(p)}$$

Here, we use *ratioA*$_a$*(L)* for our comparison of algorithms.

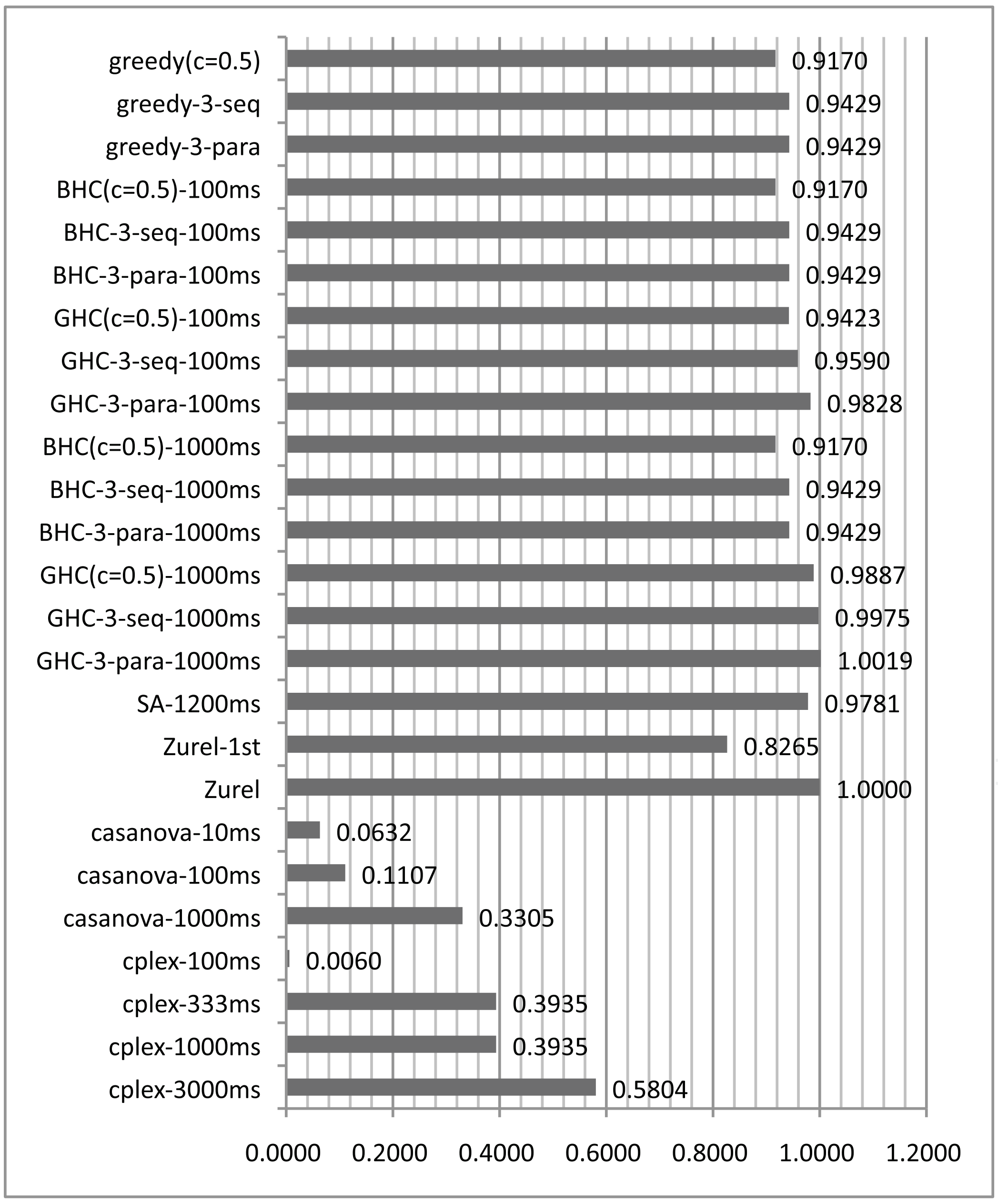


Fig. 6a. Time Performance on 20,000 bids- 256 items (Optimality Ratio)

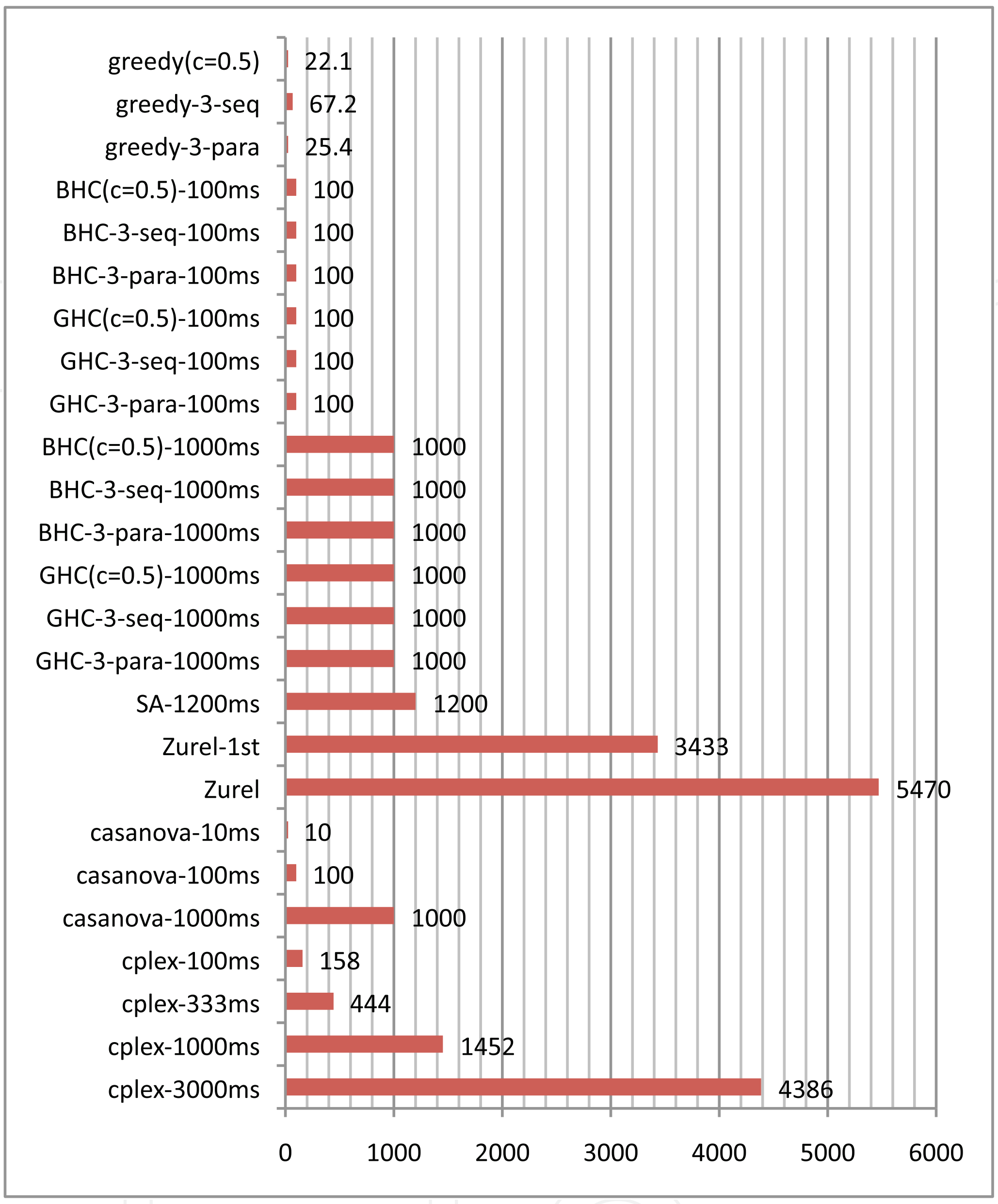


Fig. 6b. Time Performance on 20,000 bids - 256 items (Elapsed Time[msec])

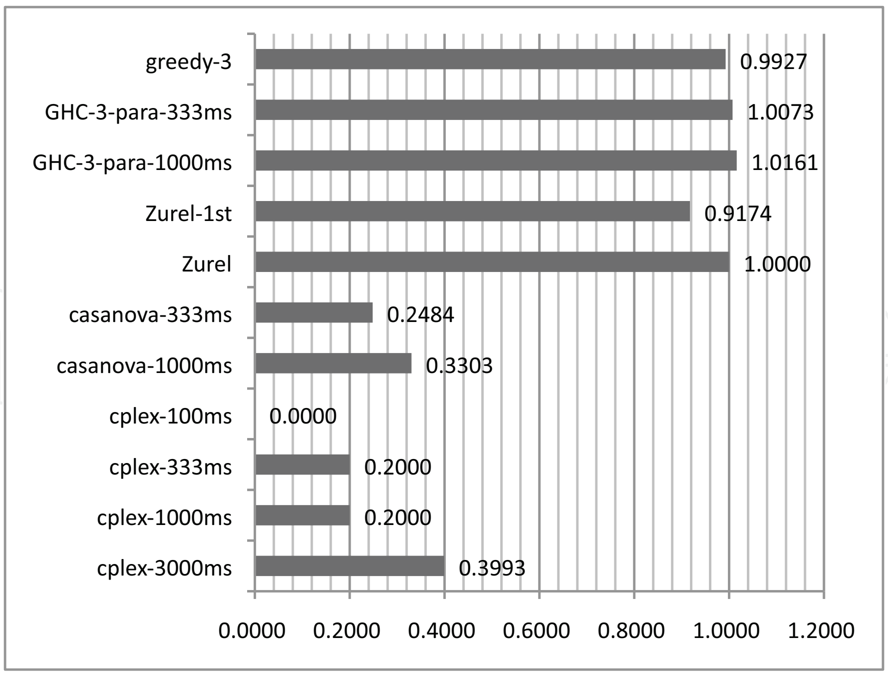
We prepared cut-off results for Casanova and HC. For example, casanova-10ms denotes the result of Casanova within 10 milliseconds. Here, for faster approximation, we used greedy-3, GHC-3, and BHC-3 but did not use greedy-11, GHC-11, and BHC-11. Here, greedy-3 uses the best results of Lehmann's greedy allocation algorithm with parameter ($0 \leq c \leq 1$ in 0.5 steps). GHC-3 and BHC-3 use the best results of the local updating with parameter ($0 \leq c \leq 1$ in 0.5 steps). Also, we prepared a variant of our algorithm that has a suffix of -seq or -para. The suffix -seq denotes the algorithm is completely executed in a sequence that is equal to one that can be executed on a single CPU computer. For example, greedy-3-seq denotes that the execution time is just the sum of execution times of three threads. The suffix -para denotes the algorithm is completely executed in a parallel manner, and the three independent threads are completely executed in parallel. Here, we used the ideal value for -para since our computer has only two cores in the CPU. The actual execution performance

will be between -seq and -para. Also, we denote the initial performance of Zurel's algorithm as Zurel-1st. Here, Zurel-1st is the result at the end of its first phase and no winners will be approximately assigned before it. cplex is the result of CPLEX with the specified time limit.
On most distributions in Fig. 6, Zurel-1st takes more than 1 second but the obtained *ratioA* is lower than greedy-3-seq. Furthermore, the average *ratioA* of GHC-3-para-1000ms is higher than Zurel while its computation time is less than both Zurel and Zurel-1st.
In Fig. 6, BHC could not get any update within the time limit so there is no update from greedy. Here, although SA performs better than greedy(C=0.5), it could not outperform GHC(C=0.5) in any case. Therefore, we can see that both *best-updating* and *random-updating* approaches are not sufficient enough for extremely short time approximation, although the *greedy-updating* approach makes a good performance in the same situation.
In many settings of CPLEX, the values are 0. This is because CPLEX could not generate initial approximation result within the provided time limit. Only datasets for two bid distributions have non-zero results for CPLEX. However, CPLEX spends around 400 msec for the computation but the results are still lower than greedy-3. On a dataset for another bid distribution, CPLEX could prepare results in 3.8 sec of computation, however, the result is still lower than greedy-3. This is because the condition we set up gave extremely short time limit so therefore CPLEX could not generate sufficient approximation results in such hard time constraint.
Fig. 7 shows the experimental result on the dataset with 100,000 bids in an auction focused on the early anytime performance. While GHC-3 and Zurel's algorithm are competitive in Fig. 6, it is clear that our proposed GHC-3 outperforms Zurel's algorithm in any time performance in Fig. 7. Note that, for Zurel's algorithm, the time needed to attain initial allocations increased approx. six times when the number of bids becomes five times larger than that of Fig. 6. However, while our GHC-3-para-1000ms only takes the same execution time (i.e, 1000 msec) for larger dataset, its average *ratioA* is higher than Zurel. Note that the GHC-3-para-333ms has still higher *ratioA* value than Zurel while its average computation time is 100 times less. We argue that our algorithm has an advantage when the number of bids increases.

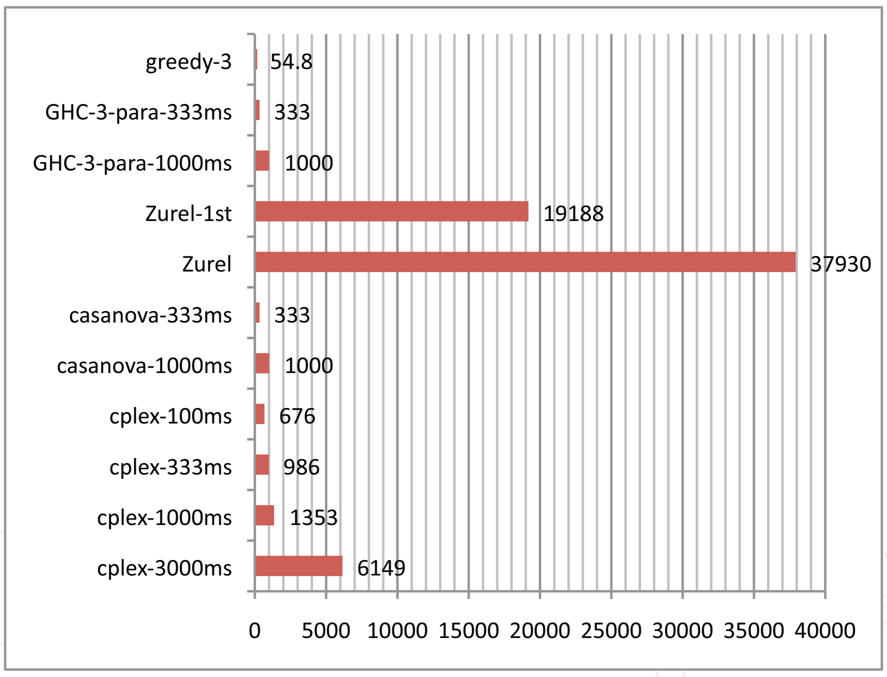
5. Related work

5.1 Approaches for optimization problems

There are really many approaches to optimization problems. Linear programming is one of the well-known approaches in this area. The winner determination problem on combinatorial auctions can be transformed into a linear programming problem. Therefore, it is possible to use a linear programming solver for the winner determination problem.
CPLEX is a well-known, very fast linear programming solver system. In (Zurel & Nisan, 2001), Zurel et al. evaluated the performance of their presented algorithm with many data sets, compared with CPLEX and other existing implementations. While the version of CPLEX used in (Zurel & Nisan, 2001) is not up-to-date, the shown performance of Zurel's algorithm is approximately 10 to 100 times faster than CPLEX. In this chapter, we showed direct comparisons to the latest version of CPLEX we could prepare. Our approach is far better than latest version of CPLEX for large-scale winner determination problems. Therefore, the performance of our approach is competitive enough with CPLEX or other similar solver systems. This is natural since Zurel's and our approaches are specialized for combinatorial auctions, and also focus only on faster approximation but do not seek optimal



Optimality Ratio



Elapsed Time[msec]

Fig. 7. Time Performance on 100,000bids - 256items

solutions. In case we need optimal solutions, it is good choice to solve the same problem by both our approach and CPLEX in parallel. This could improve anytime performance but guarantee obtaining optimal solutions. Even in such case, our approach should spend very small computation overhead.

Random-walk search is also a strong approach for approximating combinatorial optimization problems. There have been many algorithms proposed based on random-walk search mechanisms. In (Hoos & Boutilier, 2000), Casanova was proposed, which applies a

random walk SAT approach for approximating the winner determination problem in combinatorial auctions. In this chapter, we showed that our approach outperforms Casanova when the time constraint is very hard but the problem space is really large.
Simulated Annealing (SA) is another similar approach. We prepared an SA-based extension for our approach and we confirmed it increases the performance when the problem size is relatively small. However, SA needs random-walk in the early stage of its search and it decreases performance on short-time approximation.
Genetic Algorithm is another similar approach. In (Avasarala et al., 2006), Avasarala et al. proposed an approach for the winner determination problem on combinatorial auctions. However, in (Avasarala et al., 2006), they noticed that their algorithm is not effective for approximation in short time but is effective for obtaining higher optimal solutions with enough computation time. Random-walk searching is really effective approximation approach for combinatorial optimization problems. However, it is not effective when there are such hard time constraints. We focused on solving problems that are hard for such random-walk search approaches.

5.2 Approaches to obtain optimal solutions

There have been a lot of works on obtaining optimal solutions for winner determination in combinatorial auctions (de Vries & Vohra, 2003). For example, CABOB (Sandholm et al., 2005) and CASS (Fujishima et al., 1999) have been proposed by aiming to get the optimal allocations.
In (Hoos & Boutilier, 2000), it is shown that the Casanova algorithm outperforms approximation performance of CASS on winner determination. In this chapter, we showed that our approach outperforms Casanova in settings of a very large number of bids in an auction. Therefore, our approach should also outperform CASS in the same settings.
In (Sandholm et al., 2005), Sandholm et al. showed that CABOB outperforms CPLEX in several settings. However, according to our comparison, our algorithm should outperform CABOB in our settings. We argue that our approach is rather complementary to those algorithms that are seeking exact optimal solutions. It is not fair to compare their approximation performances when one guarantees obtaining optimal solutions but the other does not. Our approximation approach only covers large size problem settings that can only be handled by specialized approximation algorithms. Our approach does not contribute to advances in developing algorithms to obtain optimal solutions directly.

5.3 Other greedy approaches

Some researchers have noticed the better performance of simple greedy and incremental approaches for very large-scale problems. For example, (Sandholm, 2002) noticed the ease of approximation on very large auction problems. In (Lehmann et al., 2002), Lehmann et al. mentioned that a simple greedy approach obtains very high results when the auction problem is rather huge.
Also in (Kastner et al., 2002), Kastner et al. mentioned a potential capability of a simple incremental search approach to apply to very large auction problems and discussed the sensitivity for the number of bids in an auction. However, there is little mentioned about a detailed comparison of actual performances for several different types of datasets. In (Kastner et al., 2002), they only presented their preliminary experimental results on a dataset that is based on a single bid distribution.

Guo et al. (Guo et al., 2005) proposed similar local-search based algorithms and they argued that their approach is good for the settings of a large number of bids in a combinatorial auction problem. However, in (Guo et al., 2005), they presented very limited experimental results and little analysis or comparison to other high performance algorithms. Also in (Guo et al., 2005), they did not propose an idea that is similar to our multiple bid-weighting search. We argue that this multiple weighting search approach is very effective and that it distinguishes our approach from others. Also, we showed a detailed analysis of our experiments based on datasets generated by possible different bid distributions. We also showed direct comparisons to Zurel's approach presented in (Zurel & Nisan, 2001).

5.4 Other approaches

When we have some assumptions about models for valuation of bids, we can utilize those assumptions for better approximation. Dobzinski et al. proposed improved approximation algorithms for auctions with submodular bidders (Dobzinski & Schapira, 2006). Lavi et al, reported an LP-based algorithm that can be extended to support the classic VCG (Lavi & Swamy, 2005). Those studies mainly focused on theoretical aspects. In contrast to those papers, we rather focus on experimental analysis and implementation issues. Those papers did not present experimental analysis of the settings with a large number of bids as we presented in this chapter.

Using sequential auctions (Boutiler et al., 1999) is another approach to overcome the communication cost problem. Koenig et al. proposed a multiple-round auction mechanism that guarantees the upper bound of communication cost as fixed size *k*, that is independent from the number of agents or items in the auction (Koenig et al., 2007). Although our algorithm itself can approximate winners within a very short time with a huge number of updated bids, the communication cost problem remains.

6. Discussion

Lehmann's mechanism preserves truthfulness of the auction. However, since greedy incremental updating approach breaks *monotonicity*, an important property to provide truthfulness of auctions, the resulting auction will not be truthful. Detailed discussions and a counter example for *monotonicity* is presented in (Fukuta & Ito, 2007c). Therefore, another *monotonicity* has been proposed to approach this issue.

In real world auctions, often we open the winners and their bidding prices after the auction is finished. When we employ an approximated algorithm for winner determination, a loser who might be a winner in the optimal allocation could know the winner's bidding price in an approximate allocation after the auction finishes. In some cases, this loser had placed a higher price than the winner's for the same or a subset of the bundle. This would result in *unacceptable* allocations for bidders.

We believe that the above issue should be considered to make our mechanism acceptable by participants in the real world. Therefore, Winner-Price-Monotonicity and Weak-Winner-Price-Monotonicity are proposed to avoid *unacceptable* allocations(Fukuta & Ito, 2007a).

Definition 1. (Winner-Price-Monotonicity: WPM) *For two non-empty bundles B and B', if $B \subseteq B'$ and $v_i(B) > v_j(B')$, then j must not win bundle B'.*

Definition 2. (Weak-Winner-Price-Monotonicity: Weak-WPM) *For non-empty bundle B, if $v_i(B) > v_j(B)$, then j must not win bundle B.*

Here, proofs for following propositions are shown in (Fukuta & Ito, 2007a).

Proposition 1. *Our proposed winner determination algorithms, except for the simulated annealing-based algorithm, produce allocation W_{fin} that satisfies WPM when the algorithm reaches an end.*

Proposition 2. *In terms of any allocations that are achieved during computation (as an anytime algorithm), our proposed winner determination algorithms, except for the simulated annealing-based algorithm, satisfy Weak-WPM.*

It is a big merit to guarantee WPM and/or Weak-WPM at the algorithm level when we use it where slightly different combinatorial auctions are conducted iteratively. It seems easy to satisfy WPM and/or Weak-WPM by using any approximated winner determination algorithms by adding a pre-processing that removes all dominated bids from the bidset before starting the approximation. However, we should consider its computational overhead. For simplicity, consider a case $B = B'$ instead of $B \subseteq B'$. Let *n* be the number of items and *m* be the number of items in an auction. When *m* is very small, it is easy to look up the highest bids of each bundle by using a hash algorithm. In this case, the computational order is *O(n)*. However, it consumes a great deal of memory (of course it can be smaller than 2^m but at least additional *O(n)* of working space), and it is actually very difficult to determine good hash functions for a smaller hash table size without loss of computational speed. It is a serious problem when the memory is almost completely used up for storing the data of a large number of bids. Sometimes its computational order might reach $O(n^2)$, which is greater than that of typical good approximation algorithms. For example, the computational order of Lehmann's greedy algorithm is *O(n log n)* when we use one of the *O(n log n)* sorting algorithms on it. Furthermore, when we consider the deletion of a bid, we have to determine the highest price bid that has been made obsolete by the deleted bid, or recalculate such pre-processing for all bids again. Considering a case $B \subseteq B'$ will make the problem more difficult. Since our algorithms guarantee Weak-WPM and WPM for the produced results, there is no need to prepare such additional pre-processing.

7. Conclusions

In this chapter, we presented how greedy approach can be used in combinatorial auctions. When we have hard time constraint and a large scale problem, greedy approach works very well compared to other approaches. Two different greedy approaches can be combined to improve performance. Also it is good idea to combine parallel search approach for greedy approximation algorithm. Furthermore, greedy-based approach is also helpful to keep the result of algorithm a certain desirable property, while other random search algorithms could not.

For further reading about combinatorial auctions, (Cramton et al., 2006) is a best book for both researchers and practitioners. For further reading about the shown approach, see (Fukuta & Ito, 2007a); (Fukuta & Ito, 2007b) for detailed performance analysis, and see (Fukuta & Ito, 2006); (Fukuta & Ito, 2007c); (Fukuta & Ito, 2007a) for theoretical issues and further discussions.

8. References

Andrew, L. L.H.; Hanly, S. V. & Mukhtar, R. G. (2008). Active queue management for fair resource allocation in wireless networks. *IEEE Transactions on Mobile Computing*, pages 231-246, Feb. 2008.

Avasarala, V.; Polavarapu, H.; & Mullen, T. (2006). An approximate algorithm for resource allocation using combinatorial auctions. *In Proc. of The 2006 WIC/IEEE/ACM International Conference on Intelligent AgentTechnology (IAT2006)*, pages 571-578, 2006.

Ball, M. O.; Donohue, G. L. & Hoffman, K. (2006). Auctions for allocation of airspace system resources. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions,* chapter 20, pages 507-538. The MIT Press, 2006.

Boutiler, C.; Goldszmidt, M.; & Sabata, B. (1999). Sequential auctions for the allocation of resources with complementarities. *In Proc. of International Joint Conference on Artificial Intelligence(IJCAI1999)*, pages 527-534, 1999.

Cantillon, E. & Pesendorfer, M. (2004). Combination bidding in multi-unit auctions. *Working Paper of Harvard Business School and London School of Economics*, 2004.

Caplice, C.; Plummer, C. & Sheffi, Y. (2004). Bidder behavior in combinatorial auctions for transportation services. *Working Paper of Massachusetts Institute of Technology Center for Transportation and Logistics*, 2004.

Cramton, P.; Shoham, Y. & Steinberg, R. (2006). *Combinatorial Auctions*. The MIT Press, 2006.

de Vries, S. & Vohra, R. V. (2003). Combinatorial auctions: A survey. *International Transactions in Operational Research*, 15(3):284-309, 2003.

Dobzinski, S. & Schapira, M. (2006). An improved approximation algorithm for combinatorial auctions with submodular bidders. I*n SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1064-1073. ACM Press, 2006.

Fujishima, Y.; Leyton-Brown, K. & Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approarches. *In Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI99)*, pages 548-553, 1999.

Fukuta, N. & Ito, T. (2007a). Periodical resource allocation using approximated combinatorial auctions. *In Proc. of The 2007 WIC/IEEE/ACM International Conference on Intelligent Agent Technology (IAT2007)*, pages 434-441, 2007.

Fukuta, N. & Ito, T. (2007b). Short-time approximation on combinatorial auctions – a comparison on approximated winner determination algorithms. *In Proc. of The 3rd International Workshop on Data Engineering Issues in E-Commerce and Services (DEECS2007)*, pages 42-55, 2007.

Fukuta, N. & Ito, T. (2007c). Toward a large scale e-market: A greedy and local search based winner determination. *In Proc. of The 20th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE2007)*, pages 354-363, 2007.

Fukuta, N. & Ito, T. (2006). Towards better approximation of winner determination for combinatorial auctions with large number of bids. *In Proc. of The 2006 WIC/IEEE/ACM International Conference on Intelligent Agent Technology (IAT2006)*, pages 618-621, 2006.

Guo, Y. ; Lim, A. ; Rodrigues, B. & Zhu, Y. (2005). A non-exact approach and experiment studies on the combinatorial auction problem. *In Proc. of HICSS2005*, page 82.1, 2005.

Hohner, G.; Rich, J.; Ng, E.; Reid, G.; Davenport, A.; Kalagnanam, J.; Lee, H. S. & An, C. (2003). Combinatorial and quantity discount procurement auctions with mutual benefits at mars, incorporated. *Interfaces*, 33:23-35, 2003.

Hoos, H. H. & Boutilier, C. (2000). Solving combinatorial auctions using stochastic local search. *In Proc. of the AAAI2000*, pages 22-29, 2000.

Kastner, R.; Hsieh, C.; Potkonjak, M. & Sarrafzadeh, M. (2002). On the sensitivity of incremental algorithms for combinatorial auctions. *In Proc. International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS2002)*, pages 81-88, 2002.

Koenig, S.; Tovey, C.; Zheng, X. & Sungur, I. (2007). Sequential bundle-bid single-sale auction algorithms for decentralized control. *In Proc. of International Joint Conference on Artificial Intelligence(IJCAI2007)*, pages 1359-1365, 2007.

Lavi, R. & Swamy, C. (2005). Truthful and near-optimal mechanism design via linear programming. I*n 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 595-604, 2005.

Lehmann, D.; Mu¨ller, R. & Sandholm T. (2006). The winner determination problem. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 20, pages 507-538. The MIT Press, 2006.

Lehmann, D.; O'Callaghan, L. I. & Shoham, Y. (2002). Truth revelation in rapid, approximately efficient combinatorial auctions. *Journal of the ACM,* 49:577-602, 2002.

Leyton-Brown, K.; Pearson, M. & Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. *In Proc. of EC 2000*, pages 66-76, 2000.

McMillan, J. (1994). Selling spectrum rights. *The Journal of Economic Perspectives*, 1994.

Niyato, D. & Hossain, E. (2008). A noncooperative gametheoretic framework for radio resource management in 4g heterogeneous wireless access networks. *IEEE Transactions on Mobile Computing*, pages 332-345, March 2008.

Parkes, D. C.; Cavallo, R.; Elprin, N. ; Juda, A.; Lahaie, S.; Lubin, B.; Michael, L.; Shneidman, J. & Sultan, H. (2005). Ice: An iterative combinatorial exchange. *In The Proc. 6th ACM Conf. on Electronic Commerce (EC'05)*, 2005.

Rassenti, S. J.; Smith, V. L. & Bulfin, R. L. (1982). A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, 13:402-417, 1982.

Sabrina, F.; Kanhere, S. S. & Jha, S. K. (2007). Design, analysis, and implementation of a novel low complexity scheduler for joint resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, pages 749-762, June 2007.

Salem, N. B.; Buttyan, L.; Hubaux, J.-P. & Jakobsson, M. (2006). Node cooperation in hybrid ad hoc networks. *IEEE Transactions on Mobile Computing*, pages 365-376, April 2006.

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

Sandholm, T.; Suri, S.; Gilpin, A.; & Levine, D. (2005). Cabob: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science*, 51(3):374-390, March 2005.

Thomadakis, M. E. & Liu, J.-C. (1999). On the efficient scheduling of non-periodic tasks in hard real-time systems. *In Proc. of IEEE Real-Time Systems Symp.*, pages 148-151, 1999.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, XVI:8-37, 1961.

Xiao, L.; Chen, S.; & Zhang, X. (2004). Adaptive memory allocations in clusters to handle unexpectedly large data-intensive jobs. *IEEE Transactions on Parallel and Distributed Systems*, pages 577-592, July 2004.

Xie, T. & Qin, X. (2007). Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems*, Sep. 2007.

Yang, J. & Manivannan, D. (2005). An efficient fault-tolerant distributed channel allocation algorithm for cellular networks. *IEEE Transactions on Mobile Computing*, pages 578-587, Nov. 2005.

Zurel, E. & Nisan, N. (2001). An efficient approximate allocation algorithm for combinatorial auctions. *In Proc. of the Third ACM Conference on Electronic Commerce (EC2001)*, pages 125-136, 2001.