APPROXIMATED WINNER DETERMINATION FOR A SERIES OF COMBINATORIAL AUCTIONS

Naoki Fukuta
Faculty of Informatics, Shizuoka University, 3 5 1, Johoku Hamamatsu Shizuoka, Japan

Takayuki Ito
Graduate School of Engineering, Nagoya Institute of Technology, Gokisocho, Showa-ku, Nagoya, Japan
Sloan School of Management, Massachusetts Institute of Technology, 5 Cambridge Center
Cambridge Massachusetts, MA, U.S.A.

Keywords: Combinatorial auctions, Winner determination, Approximation algorithm.

Abstract: In this paper, we propose approximated winner determination algorithms for iteratively conducted combinatorial auctions. Our algorithms are designed to effectively reuse last-cycle solutions to speed up the initial approximation performance on the next cycle. Experimental results show that our proposed algorithms outperform existing algorithms when a large number of similar bids are contained through iterations. Also, we propose an enhanced algorithm that effectively avoids the undesirable reuse of the last solutions in the algorithm without serious computational overheads.

1 INTRODUCTION

Combinatorial auctions(Cramton et al., 2006), one of the most popular market mechanisms, have a huge effect on electronic markets and political strategies. For example, Sandholm et al. (Sandholm et al., 2005)(Sandholm, 2007) proposed real markets using their innovative combinatorial auction algorithms. The FCC tried to employ combinatorial auction mechanisms to assign spectrums to companies (McMillan, 1994). Also (Cramton et al., 2006) shows other realistic examples that utilize combinatorial auction mechanisms.

We argue that demand exists to utilize combinatorial auction mechanisms that cannot be covered by existing approaches due to hard time constraints and the limitations of usable computational resources. Resource allocation for agents in ubiquitous computing environments is a good example for understanding the needs of the short-time approximation of combinatorial auctions. In such an environment, agents must provide specific services to their users using various available resources. However, in ubiquitous computing environments, since such resources as sensors and devices are typically limited, they do not satisfy all the needs of all agents. For various reasons including physical limitations and privacy concerns, most of the resources cannot be shared with other agents. Furthermore, agents will simultaneously use two or more resources to realize desirable services for users. Since agents provide services to their own users, agents might be self-interested. Therefore, a combinatorial auction mechanism is a good option for such situations since it provides effective resource allocation to self-interested agents.

In order to utilize combinatorial auctions on the above situation, we need to complete winner determination within a very short time. Consider 256 resources and 100 agents, where each agent places, for example, from 200 to 1,000 of combinations for the items as complex bids to an auction, they will be expanded to, from 20,000 to 100,000 of atomic *OR-bids*. Here, to avoid occupying a set of resources for a long time by a certain agent, we consider a resource allocation scenario based on fixed time slice assignment model. In the scenario, those resources are auctioned and allocated from a system to agents for a fixed time period. After a certain time has passed, those are once all returned to the system and then they are auctioned again for the next period. When an agent prefers to continue using the same resources at the next period, the agent will place a higher price for the resources to increase the possibility to win them.

In ubiquitous computing scenarios, since physical locations of users are always changing, resources should be reallocated in a certain period to catch up

Fukuta N. and Ito T. (2009).
APPROXIMATED WINNER DETERMINATION FOR A SERIES OF COMBINATORIAL AUCTIONS.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 400-407
DOI: 10.5220/0001659104000407

with those changes. For better usability, the resource reallocation time period will be 0.1 to several seconds depending on services provided. Furthermore, since we should complete whole resource allocation procedure that includes pricing and communication to devices for actual resource assignment, we must determine auction winners within an extremely short time period that is far less than the actual resource allocation period.

In general, the optimal winner determination problem of combinatorial auctions is NP-hard(Cramton et al., 2006). Thus, much work focuses on tackling the computational costs for winner determination (Fujishima et al., 1999)(Cramton et al., 2006)(Sandholm et al., 2005). Some works (Lehmann et al., 2002) (Zurel and Nisan, 2001) (Hoos and Boutilier, 2000) try to achieve *approximate* solutions in winner determination.

In this paper, we propose enhanced approximation algorithms of winner determination on combinatorial auctions that are suitable for the purpose of iterative reallocation of items mentioned above. Since the above-mentioned existing algorithms are *offline* algorithms, we need to re-calculate the winners when bids are added to or deleted from the auction even when the modification of bids is only slight. Intuitively, it could be helpful to reuse results of past similar auctions for faster approximation of the current auction. However, those algorithms did not consider reusing past approximated results for performance improvement since such reuse may cause serious performance down in certain cases. Our enhanced algorithms prepared mechanisms to reuse past approximation results but avoid such performance down with very small overhead.

2 PRELIMINARIES

2.1 Winner Determination Problem

In this paper, to keep simplicity of discussion, we only focus on utility-based resource allocation problems(Thomadakis and Liu, 1999), rather than generic resource allocation problems with numerous complex constraints. 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 (Andrew et al., 2008) , security and privacy concerns(Xie and Qin, 2008), uncertainty(Xiao et al., 2004), etc). Also we only consider a scenario that is based on fixed time slice assignment model.

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). When we apply combinatorial auction mechanism for utility-based resource allocation problems, the problem can be transformed to solve a winner determination problem on combinatorial auctions.

The winner determination problem on combinatorial auctions is defined as follows(Cramton et al., 2006) : The set of bidders is denoted by $N = 1,\ldots,n$, and the set of items by $M = \{1,\ldots,m\}$. $|M| = m$. Bundle S is a set of items : $S \subseteq M$. We denote by $v_i(S)$, bidder i's valuation of 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 the bundle S. An allocation, $x_i(S)$, is feasible if it allocates no item more than once, i.e.,for all $j \in M$,

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

The winner determination problem is the problem to maximize total revenue. For feasible allocations $X \ni x_i(S)$,

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

Here, we used simple *OR-bids* representation as our bidding language. Substitutability can be represented by a set of atomic *OR-bids* with dummy items(Cramton et al., 2006).

Even when we only focus on utility-based resource allocation problems, they enforce us to solve winner determination problem with really hard-time constraint for realizing fine-grained resource allocation. Here, we have to consider that, in such resource allocation procedures, we need to spend much time for pricing and communications for actual resource allocation protocols. Therefore, we need a fast winner determination algorithm for auctions with a large number of bids. In this paper, primarily we focus on solving this problem.

2.2 Lehmann's Greedy Winner Determination

Lehmann's greedy algorithm (Lehmann et al., 2002) is a very simple but powerful linear algorithm for winner determination on combinatorial auctions. Here, a bidder declaring $< s, a >$, with $s \subseteq M$ and $a \in \mathcal{R}_+$ will be said to put out a bid $b =< s, a >$. Two bids $b =< s, a >$ and $b' =< s', a' >$ conflict iff $s \cap s' \neq \emptyset$. The greedy algorithm can be described as follows: (1) The list of bids L is sorted by some criterion. In (Lehmann et al., 2002), a method to sort the list L by descending average amount per item is proposed.

More generally, they proposed sorting L by a criterion of the form $a/|s|^c$ for some number $c \geq 0$, which possibly depends on the number of items, m. (2) A greedy algorithm generates an allocation. L is the sorted list in the first phase. The algorithm walk down the list L, accepting bids if the items demanded are still unallocated and unconflicted.

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[1]. Also they showed that the mechanism is truthful when single-minded bidders are assumed and their proposed pricing scheme is used.

2.3 Hill-climbing Search

In (Fukuta and Ito, 2006)we proposed a preliminary idea of our hill-climbing approach, and in (Fukuta and Ito, 2007a) we showed our hill-climbing approach performs well when an auction has an enormous number of bids. In this section, we summarize our proposed algorithms.

Lehmann's greedy winner determination typically performs well and the lower bound of the optimality has been analyzed(Lehmann et al., 2002). A straightforward extension of the greedy algorithm is to construct a local search algorithm that continuously updates the allocation to increase optimality. Intuitively, one allocation corresponds to one state of a local search.

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.

```
1:  function LocalSearch(Alloc, L)
2:    RemainBids:= L∩¬Alloc;
3:    for each b ∈ RemainBids as sorted order
4:      if b conflicts Alloc then
5:        Conflicted:=Alloc∩¬consistentBids({b},Alloc);
6:        NewAlloc:= (Alloc∩¬Conflicted)∪{b};
7:        ConsBids:=
8:          consistentBids(NewAlloc,RemainBids);
9:        NewAlloc:=NewAlloc∪ConsBids;
10:     if price(Alloc) < price(NewAlloc) then
11:       return LocalSearch(NewAlloc,L);
12:   end for each
13:   return Alloc
```

Function *consistentBids* finds consistent bids for the set *NewAlloc* by walking down the list *RemainBids*. Here, since a new inserted bid will wipe out some bids that are conflicting with the inserted bid, free items will appear to be allocated to other bidders after the insertion. Function *consistentBids* tries to find out potential winner bids that do not conflict to the specified allocation.

2.4 Parallel Search for Multiple Weighting

The optimality of allocations obtained by Lehmann's algorithm (and the subsequent hill-climbing) deeply depends on which value was set to c in the bid weighting function. Lehmann et al. reported that $c = 1/2$ guarantees lower bound of approximation. However, the optimal values for each auction are varied from 0 to 1 depending on the auction problem.

In (Fukuta and Ito, 2006), an enhancement has been presented for local search algorithm to parallel search for different bid weighting strategies (e.g., doing the same algorithm for both $c = 0$ and $c = 1$) In the algorithm, the value of c for Lehmann's algorithm is selected from a pre-defined list. Selecting c from neighbors of $1/2$ is reasonable, namely, $C = \{0.0, 0.1, \ldots, 1.0\}$. The results are aggregated and the best one (with the highest revenue) is selected as the final result.

2.5 Other Approximation Approaches

Zurel and Nisan(Zurel and Nisan, 2001) proposed a very high performance approximate winner determination algorithm for combinatorial auctions. The main idea is a combination of approximated positive linear program algorithm for determining initial allocation and stepwise random updates of allocations.

Hoos(Hoos and Boutilier, 2000) proposed Casanova algorithm, and showed that a generic random walk SAT solver may perform well for approximation of combinatorial auctions. It is based on scoring each search state using the revenue-per-item of the corresponding allocation.

3 ENHANCED APPROXIMATION

3.1 Fast Partial Reallocation by Last Result

In the setting of the periodical resource re-allocation scenario, winner determination occurs when some bids are revised. Theoretically, we need to recalculate winners even if only one bid is changed in the

[1]Note that, in (Sandholm et al., 2005), Sandholm et,al. determined experimentally that $c \in [0.8, 1]$ yields best performance in their approach.

auction. However, in some cases, reusing the winners of previous auctions is useful when the change is small so that it has small effects to the next winner determination process. The following simple algorithm reuses the approximation result of the last cycle when recalculation is needed due to changes of the

Here, we assume that the bids won at the last cycle (*LastWinners*) and the all bids at the last cycle (*LastBids*) are known.

```
1:  Function PartialReallocationA(
2:              LastBids,LastWinners,CurrentBids)
3:    AddedBids :=
4:     CurrentBids ∩ (¬(LastBids ∩ CurrentBids));
5:    DeletedBids :=
6:     LastBids ∩ (¬(LastBids ∩ CurrentBids));
7:    Winners := LastWinners;
8:    foreach d ∈ DeletedBids
9:     if d ∈ Winners
10:      then Winners := Winners ∩ ¬{d};
11:   foreach a ∈ AddedBids
12:    foreach w ∈ Winners
13:     if w and a are bids placed for the exactly same items
14:        and price({w}) < price({a})
15:      then Winners := (Winners ∩ ¬{w}) ∪ {a};
16:   Winners := LocalSearch(Winners,CurrentBids);
17:   return Winners
```

First, the algorithm deletes winners that no longer valid due to deletion of bids. Then, some winner bids are replaced by newly added bids. Note that we only replace a bid when the bids are placed for exactly the same items, i.e., for two bids $b_i(X)$ and $b_j(Y)$, $X = Y$, to avoid the ordering problem of newly added bids. Modification of a bid through cycles is treated as a combined operation of the deletion of previous bid and the addition of the renewed bid.

3.2 Eliminating Undesirable Reallocations

Generally speaking, the performance of reusing the partial results of similar problems depends on the problem. Therefore, in some cases, reusing the last result may cause performance decreases. To avoid such a situation, we slightly modified our algorithm to switch the initial allocation by evaluating its performance.

Here, the modified algorithm simply compares the reused result with greedy allocation. Then, the better one is used as the seed of hill-climbing improvement. Note that both our reallocation and greedy allocation algorithms complete their executions in very short time. Therefore, computational overhead for them is expected to be negligible.

```
1:  Function PartialReallocationX(
2:              LastBids,LastWinners,CurrentBids)
3:    AddedBids :=
4:     CurrentBids ∩ (¬(LastBids ∩ CurrentBids));
5:    DeletedBids :=
6:     LastBids ∩ (¬(LastBids ∩ CurrentBids));
7:    Winners := LastWinners;
8:    foreach d ∈ DeletedBids
9:     if d ∈ Winners
10:      then Winners := Winners ∩ ¬{d};
11:   foreach a ∈ AddedBids
12:    foreach w ∈ Winners
13:     if w and a are bids placed for the exactly same items
14:        and price({w}) < price({a})
15:      then Winners := (Winners ∩ ¬{w}) ∪ {a};
16:   GreedyWinners := GreedySearch(CurrentBids);
17:   if price(Winners) ≤ price(GreedyWinners)
18:    then Winners := GreedyWinners;
19:   Winners := LocalSearch(Winners,CurrentBids);
20:   return Winners
```

4 EVALUATION

4.1 Experiment Settings

We implemented our algorithms in a C program for the following experiments. We also implemented the Casanova algorithm in a C program. For Zurel's algorithm, we used Zurel's C++ based implementation that is shown in (Zurel and Nisan, 2001). Also we used CPLEX Interactive Optimizer 11.0.0 (32bit) in our experiments[2]. The experiments were done with above implementations to examine the performance differences among algorithms. The programs were employed on a Mac with Mac OS X 10.4, a CoreDuo 2.0GHz CPU, and 2GBytes of memory.

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-3 uses the best results of Lehmann's greedy allocation algorithm with parameter ($0 \leq c \leq 1$ in 0.5 steps). HC(c=0.5) uses a local search in which the initial allocation is Lehmann's allocation with $c = 0.5$ and conducts the hill-climbing search shown in section 2.3.

[2]Although CPLEX is an optimizer that can obtain optimal results, it is reported in (Sandholm et al., 2005) that its anytime approximation performance is also good.

HC-3 uses the best results of the hill-climbing search with parameter ($0 \leq c \leq 1$ in 0.5 steps). We denote the Casanova algorithm as casanova and Zurel's algorithm as Zurel Also we denote results of 1st stage of Zurel's algorithm as Zurel-1st. Note that Zurel's algorithm does not produce any approximation result until completing its 1st stage. cplex is the result of CPLEX with the specified time limit.

In the following experiments, we used 0.2 for the epsilon value of Zurel's algorithm. This value appears in (Zurel and Nisan, 2001). Also we used 0.5 for *np* and 0.15 for *wp* on Casanova that appear in (Hoos and Boutilier, 2000). Note that we set *maxTrial* to 1 but *maxSteps* to ten times the number of bids in the auction.

We conducted detailed comparisons among our past presented algorithms and the other existing algorithms mentioned above. The details of the comparisons are shown in (Fukuta and Ito, 2007b) and (Fukuta and Ito, 2007a). In (Fukuta and Ito, 2007b) and (Fukuta and Ito, 2007a), we prepared datasets with 20,000 bids in an auction. The datasets were produced by CATS(Leyton-Brown et al., 2000) with default parameters in 5 different distributions. They contain 100 trials for each distribution. Each trial is an auction problem with 256 items and 20,000 bids.[3]

However, since CATS common datasets only provide static bids for an auction, we prepared extended usage for those datasets to include the dynamic changes of bids in an auction.

Procedure: In each auction, the bid set is divided into k blocks by the order of bid generation (i.e., bid id). The bid set is modified totally k times and the modification is done in each second. In each 1-second period, a block is marked as hidden so that bids within these marked blocks are treated as *deleted bids*. For example, at the first period, the first block is marked as hidden so the remaining bids (second to kth blocks) are used for winner determination. After 1 second, the mark is moved to the second block (i.e., the first, and the third to kth blocks are used) and the winner determination process is restarted due to this change. Here, we can see it as the bids in the first block are newly added to the auction and the bids in the second block are deleted from the auction. This process is repeated until the mark has been moved to the kth block. Finally, all marks are cleared and the winner determination process is restarted with full bids in the auction. Ordinary algorithms should be completely restarted in each cycle. However, when we use our proposed reallocation algorithms, some intermediate results can be reused in the next cycle in the same auction.

Since the bid set in the $k+1$th cycle completely equals the bids of the auction, the results of the $k+1$th cycle can be compared to our previous experimental results.

[3]Due to difficulty of dataset preparation, we only prepared five distributions. Producing a dataset with other distributions is difficult in feasible time.

4.2 Time Performance

Table 1 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 equal 1 as follows.

Let A be a set of algorithms, $z \in A$ be the zurel's approximation algorithm, D 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 in a dataset, the average revenue ratio $rationA_a(D)$ for algorithm $a \in A$ for dataset D is defined as follows:

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

Here, we use $ratioA_a(D)$ for our comparison of algorithms.

The name of each distribution is taken from (Leyton-Brown et al., 2000). We prepared the cut-off results of Casanova and HC. For example, casanova-10ms denotes the results of Casanova within 10 milliseconds. Also we prepared a variant of our algorithm that has a suffix of -seq or -para. The suffix -seq denotes that the algorithm is completely executed sequentially that is equal to be executed on a single CPU computer. For example, greedy-3-seq denotes that the execution time is the sum of execution times spent by three threads. The suffix -para denotes that the algorithm is completely executed in a parallel manner, the three independent threads are completely executed in parallel. Here, we used ideal value for -para since our computer has only two cores in the CPU. The actual execution performance will be between -seq and -para.

Additionally, we added results with names AHC or XHC in the same table. They are the average approximated results of the $k+1$th cycle of auctions with our proposed algorithms `PartialReallocationA` and `PartialReallocationX`, respectively.

In most distributions, Zurel-1st takes more than one second but the obtained optimality is lower than greedy-3-seq. However, our proposed HC-3 performs better or slightly lower although their computation times are shorter than Zurel-1st and Zurel, excluding L3. Surprisingly, in most cases, the results of XHC-3-seq-100ms are better than HC-3-seq-1000ms while their spent computation time is only 1/10. This fact

Table 1: Time Performance of (k+1)th cycle on 20,000bids-256items (k=10).

	L2		L3		L4		L6		L7		average	
greedy(c=0.5)	1.0002	(23.0)	0.9639	(19.0)	0.9417	(23.0)	0.9389	(23.4)	0.7403	(22.1)	0.9170	(22.1)
greedy-3-seq	1.0003	(69.1)	0.9639	(59.2)	0.9999	(72.9)	0.9965	(67.8)	0.7541	(66.8)	0.9429	(67.2)
greedy-3-para	1.0003	(26.4)	0.9639	(20.9)	0.9999	(28.4)	0.9965	(26.0)	0.7541	(25.5)	0.9429	(25.4)
HC(c=0.5)-100ms	1.0004	(100)	0.9741	(100)	0.9576	(100)	0.9533	(100)	0.8260	(100)	0.9423	(100)
HC-3-seq-100ms	1.0004	(100)	0.9692	(100)	1.0000	(100)	0.9966	(100)	0.8287	(100)	0.9590	(100)
AHC-3-seq-100ms	1.0004	(100)	0.9690	(100)	1.0006	(100)	0.9974	(100)	1.0225	(100)	0.9980	(100)
XHC-3-seq-100ms	1.0004	(100)	0.9813	(100)	1.0005	(100)	0.9987	(100)	1.0217	(100)	1.0005	(100)
HC-3-para-100ms	1.0004	(100)	0.9743	(100)	1.0001	(100)	0.9969	(100)	0.9423	(100)	0.9828	(100)
AHC-3-para-100ms	1.0004	(100)	0.9741	(100)	1.0006	(100)	0.9977	(100)	1.0249	(100)	0.9995	(100)
XHC-3-para-100ms	1.0004	(100)	0.9820	(100)	1.0006	(100)	0.9988	(100)	1.0249	(100)	1.0013	(100)
HC(c=0.5)-1000ms	1.0004	(1000)	0.9856	(1000)	0.9771	(1000)	0.9646	(1000)	1.0157	(1000)	0.9887	(1000)
HC-3-seq-1000ms	1.0004	(1000)	0.9804	(1000)	1.0003	(1000)	0.9976	(1000)	1.0086	(1000)	0.9975	(1000)
AHC-3-seq-1000ms	1.0004	(1000)	0.9795	(1000)	1.0007	(1000)	0.9982	(1000)	1.0266	(1000)	1.0011	(1000)
XHC-3-seq-1000ms	1.0004	(1000)	0.9830	(1000)	1.0006	(1000)	0.9991	(1000)	1.0266	(1000)	1.0019	(1000)
HC-3-para-1000ms	1.0004	(1000)	0.9856	(1000)	1.0006	(1000)	0.9987	(1000)	1.0240	(1000)	1.0019	(1000)
AHC-3-para-1000ms	1.0004	(1000)	0.9847	(1000)	1.0008	(1000)	0.9990	(1000)	1.0272	(1000)	1.0024	(1000)
XHC-3-para-1000ms	1.0004	(1000)	0.9853	(1000)	1.0008	(1000)	0.9996	(1000)	1.0272	(1000)	1.0027	(1000)
Zurel-1st	0.5710	(11040)	0.9690	(537)	0.9983	(2075)	0.9928	(1715)	0.6015	(1796)	0.8265	(3433)
Zurel	1.0000	(13837)	1.0000	(890)	1.0000	(4581)	1.0000	(4324)	1.0000	(3720)	1.0000	(5470)
casanova-10ms	0.2583	(10)	0.0069	(10)	0.0105	(10)	0.0202	(10)	0.2577	(10)	0.0632	(10)
casanova-100ms	0.2583	(100)	0.0069	(100)	0.0105	(100)	0.0202	(100)	0.2577	(100)	0.1107	(100)
casanova-1000ms	0.5357	(1000)	0.1208	(1000)	0.0861	(1000)	0.1486	(1000)	0.7614	(1000)	0.3305	(1000)
cplex-100ms	0.0000	(288)	0.0000	(121)	0.0299	(111)	0.0000	(150)	0.0000	(119)	0.0060	(158)
cplex-333ms	0.0000	(489)	0.0000	(393)	0.9960	(497)	0.9716	(354)	0.0000	(487)	0.3935	(444)
cplex-1000ms	0.0000	(1052)	0.0000	(1039)	0.9960	(1143)	0.9716	(1140)	0.0000	(2887)	0.3935	(1452)
cplex-3000ms	0.0000	(9171)	0.9338	(3563)	0.9964	(3030)	0.9716	(3077)	0.0000	(3090)	0.5804	(4386)

(each value in () is time in milliseconds)

shows that our XHC-3 could effectively reuse the approximated results of previous cycles.

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 L4 and L6 have results for CPLEX. For them, CPLEX spends around 400 msec for the computation but the results are still lower than greedy-3. For L3, 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.

Table 2 shows the experimental result on the datasets with 100,000 bids in an auction focused on execution time of the approximation. The settings are identical as Table 1 excluding the difference of number of bids in an auction. Due to hard time constraint, results of -seq-100ms (sequential execution with a cutoff time of 100ms) are excluded from the table since they could not complete their execution within the cutoff time. Here, our proposed methods (AHC-3,XHC-3) clearly have a certain advantage of their performance time ratio. HC-3,AHC-3,and XHC-3 produced acceptable approximated results within 100 to 1000 msec that are 2 to 443 times faster than Zurel's approximation. Especially, in most cases, our AHC-3-para-100ms outperforms HC-3-seq-1000ms and HC-3-para-1000ms.

On above experiments, we used $k = 10$. Table 3 shows average time performance of our algorithms on $k = 2, 5, 10, 20, 40$, respectively. At same cutoff time, XHC-3 obtains higher or at least same performance compared to HC-3. Furthermore, XHC-3-para-100ms outperforms HC-3-para-1000ms when $k \geq 20$, while its computation time is 10 times shorter.

On above experiments, for direct comparison to other existing algorithms, we have shown results on final(e.g., $(k+1)$th) cycle in our procedure. We also confirmed performance improvement on intermediate cycle in our procedure. Table 4 shows average results on intermediate cycles for three algorithms (HC-3,AHC-3,and XHC-3). Here, since we do not have approximation results for those intermediate cycles on Zurel's algorithm, instead of using *ratioA*, we normalized all values as HC-3-para-1000msec equals 1. Since our algorithms improve results much more for latter cycles by cumulative reuse of the last cycle, we used first four cycles in this comparison. For results on $k = 5, 10, 20, 40$, we used an average value for first four intermediate cycles (e.g., from 2nd to 5th). Note

Table 2: Time Performance of (k+1)th cycle on 100,000bids-256items (k=10).

	L2		L3		L4		L6		L7		average	
HC-3-para-100ms	1.1098	(100)	0.9836	(100)	1.0003	(100)	1.0009	(100)	0.8688	(100)	0.9927	(100)
AHC-3-para-100ms	1.1098	(100)	0.9836	(100)	1.0003	(100)	1.0009	(100)	0.9941	(100)	1.0177	(100)
XHC-3-para-100ms	1.1098	(100)	0.9880	(100)	1.0003	(100)	1.0010	(100)	0.9939	(100)	1.0186	(100)
HC-3-para-1000ms	1.1098	(1000)	0.9880	(1000)	1.0003	(1000)	1.0010	(1000)	0.9814	(1000)	1.0161	(1000)
AHC-3-para-1000ms	1.1098	(1000)	0.9880	(1000)	1.0003	(1000)	1.0010	(1000)	0.9991	(1000)	1.0197	(1000)
XHC-3-para-1000ms	1.1098	(1000)	0.9889	(1000)	1.0003	(1000)	1.0011	(1000)	0.9990	(1000)	1.0198	(1000)
zurel-1st	0.8971	(74943)	0.9827	(2257)	0.9998	(5345)	0.9987	(4707)	0.7086	(8688)	0.9174	(19188)
Zurel	1.0000	(91100)	1.0000	(6036)	1.0000	(30568)	1.0000	(44255)	1.0000	(17691)	1.0000	(37930)
cplex-100ms	0.0000	(2022)	0.0000	(232)	0.0000	(143)	0.0000	(133)	0.0000	(852)	0.0000	(676)
cplex-333ms	0.0000	(2021)	0.0000	(559)	0.9998	(1084)	0.0000	(412)	0.0000	(852)	0.2000	(986)
cplex-1000ms	0.0000	(2021)	0.0000	(1045)	0.9998	(1085)	0.0000	(1328)	0.0000	(1285)	0.2000	(1353)
cplex-3000ms	0.0000	(3496)	0.0000	(3286)	0.9998	(5207)	0.9965	(3092)	0.0000	(15667)	0.3993	(6149)

(each value in () is time in milliseconds)

Table 3: Time Performance of (k+1)th cycle on 20,000bids-256items (k=2,5,10,20,40).

	k=2		k=5		k=10		k=20		k=40	
HC-3-para-100ms	0.9828	(100)	0.9828	(100)	0.9828	(100)	0.9828	(100)	0.9828	(100)
AHC-3-para-100ms	0.9952	(100)	0.9979	(100)	0.9995	(100)	1.0003	(100)	1.0009	(100)
XHC-3-para-100ms	0.9952	(100)	0.9998	(100)	1.0013	(100)	1.0021	(100)	1.0028	(100)
HC-3-para-1000ms	1.0019	(1000)	1.0019	(1000)	1.0019	(1000)	1.0019	(1000)	1.0019	(1000)
AHC-3-para-1000ms	1.0019	(1000)	1.0021	(1000)	1.0024	(1000)	1.0026	(1000)	1.0027	(1000)
XHC-3-para-1000ms	1.0019	(1000)	1.0025	(1000)	1.0027	(1000)	1.0031	(1000)	1.0035	(1000)

(each value in () is time in milliseconds)

that, only for results on $k = 2$, we used the results on 2nd cycle since we do not have other intermediate cycles when $k = 2$. Here, results of XHC-3 constantly better than AHC-3 and HC-3 and the differences are bigger when k is increased.

5 RELATED WORK

There have been a lot of works on the optimal algorithms for winner determination in combinatorial auctions(de Vries and Vohra, 2003). Recently, Dobzinski et, al. proposed improved approximation algorithms for auctions with submodular bidders(Dobzinski and Schapira, 2006). Lavi et, al, reported an LP based algorithm that can be extended to support the classic VCG(Lavi and Swamy, 2005). Those researches are mainly focused on theoretical aspects. In contrast to those papers, we rather focus on experimental and implementation aspects. Those papers did not present experimental analysis about the settings with large number of bids we presented in this paper. Also, Guo(Guo et al., 2005) proposed local-search based algorithms for large number of bids in combinatorial auction problems. However, they did not present experiments with such a huge number of bids we used in our experiments.

CPLEX is a well-known, very fast linear programming solver system. In (Zurel and 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 and 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 paper, 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 better than CPLEX in our settings. 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 solutions. In case we need optimal solutions, it is good choice to solve the same problem by both our approach and CPLEX in parallel.

The above approaches are based on offline algorithms and therefore there are no considerations about addition and deletion of bids in their approximation processes. Although our algorithms are not strict online algorithms, it is possible to reuse the last results when bids are modified and recalculation is necessary.

Table 4: Time Performance of intermediate cycles on 20,000bids-256items (k=2,5,10,20,40).

	k=2	k=5	k=10	K=20	K=40
HC-3-para-100ms	0.9889	0.9847	0.9829	0.9826	0.9818
AHC-3-para-100ms	0.9889	0.9805	0.9838	0.9874	0.9897
XHC-3-para-100ms	0.9892	0.9917	0.9943	0.9951	0.9966

(values are normalized as HC-3-para-1000msec equals 1)

6 CONCLUSIONS

In this paper, we proposed enhanced approximation algorithms for combinatorial auctions that are suitable for the purpose of iterative reallocation of items. Our proposed algorithms effectively reuse the last solutions to speed up initial approximation performance. The experimental results showed that our proposed algorithms outperform existing algorithms in some aspects. However, we found that in some cases reusing the last solutions may worsen performance compared to ordinary approximation from scratch. We proposed an enhanced algorithm that effectively avoids the undesirable reuse of the last solutions in the algorithm. We showed this is especially effective when a non-negligible number of existing bids are deleted from the last cycle.

REFERENCES

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

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

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

Dobzinski, S. and Schapira, M. (2006). An improved approximation algorithm for combinatorial auctions with submodular bidders. In *Proc. of the seventeenth annual ACM-SIAM symposium on Discrete algorithm(SODA2006)*, pages 1064–1073. ACM Press.

Fujishima, Y., Leyton-Brown, K., and 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 (IJCAI1999)*, pages 548–553.

Fukuta, N. and 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.

Fukuta, N. and 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.

Fukuta, N. and 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.

Guo, Y., Lim, A., Rodrigues, B., and Zhu, Y. (2005). A non-exact approach and experiment studies on the combinatorial auction problem. In *Proc. of the 38th Hawaii International Conference on System Sciences(HICSS2005)*, page 82.1.

Hoos, H. H. and Boutilier, C. (2000). Solving combinatorial auctions using stochastic local search. In *Proc. of the Proc. of 17th National Conference on Artificial Intelligence (AAAI2000)*, pages 22–29.

Lavi, R. and Swamy, C. (2005). Truthful and near-optimal mechanism design via linear programming. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS2005)*, pages 595–604.

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

Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *Proc. of ACM Conference on Electronic Commerce (EC2000)*, pages 66–76.

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

Sandholm, T. (2007). Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3):45–58.

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

Thomadakis, M. E. and 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.

Xiao, L., Chen, S., and Zhang, X. (2004). Adaptive memory allocations in clusters to handle unexpectedly large data-intensive jobs. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):577–592.

Xie, T. and Qin, X. (2008). Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems*, 19(5):682–697.

Zurel, E. and 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.