Journal of Information Processing Vol.21 No.1 9–15 (Jan. 2013)

[DOI: 10.2197/ipsjjip.21.9]

Regular Paper

An Approach to VCG-like Approximate Allocation and Pricing for Large-scale Multi-unit Combinatorial Auctions

Naoki Fukuta[1,a)]

Received: January 27, 2012, Accepted: April 2, 2012

Abstract: A multi-unit combinatorial auction is a combinatorial auction that has some items that can be seen as indistinguishable. Although the mechanism can be applied to dynamic electricity auctions and various purposes, it is difficult to apply to large-scale auction problems due to its computational intractability. In this paper, I present an idea and an analysis about an approximate allocation and pricing algorithm that is capable of handling multi-unit auctions. The analysis shows that the algorithm effectively produces approximation allocations that are necessary in pricing. Furthermore, the algorithm can be seen as an approximation of VCG (Vickrey-Clarke-Groves) mechanism satisfying budget balance condition and bidders' individual rationality without having unrealistic assumptions on bidders' behaviors. I show that the proposed allocation algorithm successfully produced good allocations for those problems that could not be easily solved by ordinary LP solvers due to hard time constraints.

Keywords: combinatorial auction, approximation, resource allocation, pricing algorithm

1. Introduction

A multi-unit combinatorial auction is a combinatorial auction in that some items can be seen as indistinguishable. A combinatorial auction is an auction that allows bidders to place bids for a combination of items rather than a single item [1]. As described in many literatures [1], [2], [3], combinatorial auctions [1] that are one of the most popular market mechanisms, have a huge effect on electronic markets and political strategies. Combinatorial auctions provide suitable mechanisms for efficient allocation of resources to self-interested attendees [1]. Therefore, many works have been done to utilize combinatorial auction mechanisms for efficient resource allocation [1]. For example, the FCC (Federal Communications Commission) tried to employ combinatorial auction mechanisms for assigning spectrums to companies [3]. Also some possible applications have been investigated for the procurement of frieght transportation [4]. However, to make combinatorial auctions fully functional in such a situation, a special mechanism to calculate each winner's payment should be applied. VCG (Vickery-Clarke-Groves) mechanism is a major approach to calculate such payments [1], [5], [6]. In VCG, it repeatedly calculates winners of slightly different auction problems. Since winner determination for such an auction can be an NP-hard combinatorial optimization problem [1], it has been investigated to make such a mechanism computationally tractable.

Multi-unit combinatorial auctions are expected to be used on many problems that include quantitative or countable items [7], [8], [9], [10]. To overcome the winner determination problem, many approaches have been proposed. For example, Zurel et al. proposed a heuristic approach that combines approximation of LP and a local search algorithm [11]. Also a parallel greedy approach [12], performance analyses of algorithms [13], [14], a theoretical investigations [15], [16], and its possible enhancements [17] have been proposed. However, they are mainly focused on single-unit auction scenarios and they cannot be easily applied to multi-unit scenarios [18].

There are some approaches to solve winner determination problem specifically on multi-unit combinatorial auctions [19], [20]. However, they did not consider how their approximation algorithms can be extended to make efficient pricing [21]. The only work that considers this concern is Hafalir's work [21]. However, their work mainly focused on single-item multi-unit scenario but does not cover combinatorial and multi-unit auctions.

In this paper, I present an idea and an analysis about an approximation approach that employs an approximate allocation and pricing algorithm that is capable to handle multi-unit auctions efficiently. The pricing mechanism is based on the fast approximate allocation algorithm that behaves as an approximation of VCG mechanism and also satisfies budget balance condition and individual rationality without having an assumption called *single-minded bidders* [22]. I show that the proposed allocation algorithm successfully produced good allocations for those problems that could not be easily solved by ordinary LP (Linear Programming) solvers due to hard time constraints.

2. Preliminary

2.1 Multi-unit Combinatorial Auctions

Combinatorial auction is an auction that allows bidders to place bids for a combination of items rather than a single item [1]. The winner determination problem on single unit combinatorial auctions is defined as follows [1]: The set of bidders is denoted by

[1] Faculty of Informatics, Shizuoka University Hamamatsu, Shizuoka 432–8011, Japan

[a)] fukuta@cs.inf.shizuoka.ac.jp

$N = \{1, \ldots, n\}$, and the set of items by $M = \{m_1, \ldots, m_k\}$. $|M| = k$. Bundle S is a set of items: $S \subseteq M$. I 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, for all $j \in M$.

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

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

Note that I used simple *OR-bid* representation as the bidding language. Substitutability can be represented by a set of atomic *OR-bids* with dummy items [1].

When some items in auction can be replaceable each other, i.e., they are indistinguishable, the auction is called multi-unit auction. Multi-unit combinatorial auction is the case when some items are indistinguishable in a combinatorial auction [1].

2.2 Extending Lehmann's Approximation Approach

Lehmann's greedy algorithm [22] is a very simple but powerful linear algorithm for winner determination in 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 $b =< s, a >$. The greedy algorithm can be described as follows. (1) The bids are sorted by some criterion. In Ref. [22], 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, $c \geq 0$, possibly depending on the number of items, k. (2) A greedy algorithm generates an allocation. L is the sorted list in the first phase. It walks down the list L, and allocates items to bids whose items are still unallocated.

The allocation algorithm can naturally be extended to multi-unit combinatorial auction problems. However, they did not mention about the applicability to multi-unit combinatorial auctions in their paper.

In Refs. [12], [13], and [17], it has shown that their hill-climbing approach outperforms SA [12], SAT-based algorithms [23], LP-based heuristic approximation approach [11], and a recent LP solver product in the setting when an auction has a massively large number of bids but the given time constraint is very hard. However, the algorithm is designed for single-unit combinatorial auction problems so it cannot be applied to multi-unit problems directly.

2.3 Winner Approximation and Pricing

It is crucial for a combinatorial auction mechanism to have proper pricing mechanism. In VCG (Vickery-Clarke-Groves) mechanism, prices that winners will pay will be given as follows [5]. A payment p_n for a winner n is calculated by

$$p_n = \alpha_n - \sum_{i \neq n, S \subseteq M} v_i(S) x_i(S)$$

Here, the right part of the right side of the equation denotes the sum of all bidding prices of won bids, excluding the bids that are placed by the bidder n. The left part of the right side of the equation, α_n is defined by

$$\alpha_n = max \sum_{i \neq n, S \subseteq M} v_i(S) x_i(S)$$

for a feasible allocation $X \ni x_i(S)$. This means that the α_n is the sum of all bidding prices of won bids when the allocation is determined as if a bidder n does not place any bids for the auction.

In Ref. [5], Nisan et al. showed that optimal allocations should be used for VCG-based pricing to make the auction incentive compatible (i.e., revealing true valuations is the best strategy for each bidders). Also, Lehmann et al. showed that VCG-based pricing with approximate winner determination will not make the auction incentive compatible even when it is assumed that all bidders are single-minded (i.e., each bidder can only place single bid at each auction) [22].

To overcome this issue, Lehmann et al. prepared a special pricing mechanism that can only be applied to their approximate greedy winner determination [22]. However, this pricing mechanism can only be applied to their allocation algorithm and cannot be applied to other approximation allocation algorithms. Also the mechanism is incentive compatible only when single-minded bidders are assumed [22]. Another approach has been proposed to realize tractable and incentive compatible combinatorial auctions in Ref. [24] based on partial search and LP-based approach. However, it requires much stronger assumptions called "known single minded bidders" [24].

The main problem in which VCG-based pricing is applied to approximation allocation of items is that there are the cases that: (1) the price for a won bid is rather higher than the bid price, and (2) the price for a won bid is less than zero, it means the bidder will win the items and also obtain some money rather than paying for it [5]. In the situation of (1), it violates individual rationality (i.e., the one will not pay a higher price than the placed bid when the one won the bundle of items). Also the situation of (2) is not preferable for both auctioneers and sellers.

3. Proposed Algorithm

In this section, I propose an approximation allocation algorithm for multi-unit combinatorial auctions, as follows.

The inputs are *Alloc*, *L*, and *Stocks*. *L* is the bid list of an auction. *Stocks* is the list of the number of auctioned units for each distinguishable item type. *Alloc* is the initial greedy allocation of items for the bid list.

```
1:  function LocalSearch(Alloc, L, Stocks)
2:    RemainBids:= L - Alloc;
3:    sortByLehmannC(RemainBids);
4:    for each b ∈ RemainBids
5:      RestStocks:=getRestStocks({b}, Stocks);
6:      AllocFromWinners:=greedyAlloc(RestStocks, Alloc);
7:      RestStocks:=
8:        getRestStocks(AllocFromWinners + {b}, RestStocks);
9:      AllocFromRest:=
10:       greedyAlloc(RestStocks, RemainBids − {b});
11:     NewAlloc:=
12:       {b} + AllocFromWinners + AllocFromRest;
13:     if price(Alloc) < price(NewAlloc) then
14:       return LocalSearch(NewAlloc,L,Stocks);
```

15: **end for each**
16: **return** *Alloc*

The function *sortByLehmannC*(*Bids*) has an argument *Bids*. The function sorts the list of bids *Bids* by descending order of Lehmann's weighted bid price. The result are directly stored (overwritten) to the argument *Bids*. The function *getRestStocks*(*Bids*, *Stocks*) has two arguments: *Bids* and *Stocks*. The function returns how many unit of items will remain after allocating the items in *Stocks* to the list of bids *Bids*. The function *greedyAlloc*(*Stocks*, *Bids*) has two arguments: *Stocks* and *Bids*. The function allocates the items in *Stocks* to the list of bids *Bids* by using Lehmann's greedy allocation, and then the winner bids are returned as the return value. The function *price* calculates the sum of bidding prices for bids specified in the argument.

The optimality of allocations got by Lehmann's algorithm (and the following hill-climbing) deeply depends on which value was set as the bid sorting criterion c. Again, in Ref. [22], 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, the optimal values for each auction are varied from 0 to 1 even if the number of items is constant. Therefore, an enhancement has been proposed for this kind of local search algorithms by using parallel searches for multiple sorting criterion c [12]. Although the proposed enhancement is primarily designed for single-unit combinatorial auctions, this approach can be applied to the above mentioned approximation algorithm for multi-unit combinatorial auctions. In the algorithm, the value of c for Lehmann's algorithm is selected from a predefined list. It is reasonable to select c from neighbors of $1/2$, namely, $C = \{0.0, 0.1, \ldots, 1.0\}$. The results are aggregated and the best one (i.e., that has the highest revenue) is selected as the final result [*1].

To realize a pricing mechanism that receives little effect from the winners bid prices, I use the following algorithm. The inputs are *Alloc*, *L*, and *Stocks*. *L* is the bid list of an auction. *Stocks* is the list of the number of auctioned units for each distinguishable item type. *Alloc* is the initial allocation of items for the bid list that is obtained by the previously defined *LocalSearch* function.

1: **function** transformToSWPM(*Alloc*, *L*, *Stocks*)
2: *RemainBids*:= *L* - *Alloc*;
3: *sortByLehmannC*(*RemainBids*);
4: *clear*(*payment*);
5: **for each** $b \in Alloc$
6: *RestStocks*:=*getRestStocks*(*Alloc* − {*b*}, *Stocks*);
7: *AllocForB*:=*greedyAlloc*(*RestStocks*, *RemainBids*);
8: *NewAlloc*:=*Alloc*-{*b*} + *AllocForB*;
9: **if** *price*(*Alloc*) < *price*(*NewAlloc*) **then**
10: **return** transformToSWPM(*NewAlloc*,*L*,*Stocks*);
11: **else** $payment_b = price(NewAlloc) - price(Alloc - \{b\})$
12: **end for each**
13: **return** (*Alloc*,*payment*)

[*1] An analysis discussing the sensitivity of value c used in this kind of algorithms has been presented in Ref. [17].

The above algorithm computes the price to be paid for each winner bid. The payment price for a winner bid b is denoted by $payment_b$, and it's value is obtained by $price(NewAlloc) - price(Alloc - \{b\})$. When the obtained payment price is higher than the bidding price of the winner bid, the algorithm discards the winner bid and place the items to *AllocForB*. Finally, the algorithm produces modified allocations *Alloc* and their payment prices *payment* that satisfies budget constraints for bidders.

For simplicity of description, the above algorithm is written with single-minded bidders assumption. To extend the algorithm without the assumption can be realized by just replacing $\{b\}$ with the all bids that come from the bidder of $\{b\}$.

4. Evaluation

4.1 Experiment Settings

For the evaluation of winner determination performance on combinatorial auction, LeytonBrown et al. proposed CATS benchmark testsuite [7]. However, even if multi-unit auction is referred in Ref. [7], CATS suite does not include any data generation algorithm for multi-unit combinatorial auction. Therefore, I extended existing auction problem generation algorithm to support multi-unit auctions by the following way [*2].

Extending CATS standard dataset to multi-unit problems: Each auction problem generation algorithm in CATS generates artificial bids for a fixed size of items. The generated auction problems are single-unit combinatorial auction problems where each item in the auction has only one stock and these items are distinguished each other. When I consider each item has many stocks in a small size auction, the allocation problem could be rather much easier than that of single stocks since many conflicts (i.e., the situation that some bids placed to a set that includes an identical item) among bids can be automatically solved by allocating items to such conflicting bids. So, in the situation, many bids could win the items and only a limited number of bids might fail to win the items. However, when there are a huge number of bids in a single-unit combinatorial auction, the problem could be complex enough even when I assume there are a certain number of stocks for each item in the auction. Here, I extend the dataset produced by CATS workbench by adding number of stocks for each non-dummy item in an auction. I call this 'the number of stocks for each item' approach.

This representation is also useful for representing items that can be shared with a limited number of people. For example, when I represent a fact that a radio frequency band can be shared by three devices at a time, the stocks for the item (i.e., the number of shared users for the bandwidth) is set to 3 in the auction. Also this representation does not have to generate a large number of bids even when the number of stocks is large. Another representation could be based on a representation of indistinguishable relationships among items but this representation inevitably generates a large number of definitions for such relationships. Therefore, I use 'the number of stocks for each item' approach here.

The actual preparations of datasets have been done as follows. I used the bid distributions (i.e., the way to generate bids for

[*2] A preliminary analysis about this issue has been presented in Ref. [18].

items) that are defined and usable for generating the auction problems with a specified number of bids. Here I chose 20,000 bids and 100,000 bids for each auction so I chose the bid distribution L2, L3, L4, L6, L7, arbitrary, matching, paths, regions, and scheduling [*3]. I prepared 100 auction problems for each bid distribution for both the size of 20,000 bids and 100,000 bids in an auction. I used those settings to make the results comparable to other papers [13]. The bid distribution names were borrowed from Ref. [7] [*4]. Here, to keep the meaning of data generation algorithms, I chose fixed values for those stocks (e.g., every item has 4 stocks). I chose four fixed values, 2, 4, 16, and 256 for the number of stocks.

Note that, as mentioned before, in multi-unit auction problems, some bids that could be treated as dominated bids (e.g., having a higher-price bid for the same bundle of items) in single-unit auction problems could be winners of the auction. Therefore, I did not remove such bids in the bid generation process of the original single-unit auction problems by CATS.

Evaluating overheads to support multi-unit problems: It is said that the difficulty of winner determination in single-unit combinatorial auction problems is sometimes rather more difficult than that in multi-unit problems [2]. Therefore, some algorithms are focused on single-unit problems and they may behave better than an algorithm which also supports multi-unit problems. For comparison to existing winner determination algorithms that only support single-unit combinatorial auctions, I prepared a dataset with 20,000 bids in a single-unit combinatorial auction. The dataset was produced by CATS [7] with default parameters in 5 different distributions. They contain 100 trials for each distribution, and the number of bids does not include the number of dominated bids since they have been removed in this setting. Each trial is an auction problem with 256 items [*5].

Compared algorithms: In this analysis, I compared the following search algorithms: greedyL(C=0.5) uses Lehmann's greedy allocation algorithm [22] with parameter ($c = 0.5$). 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 [12]. HC-3 uses the best results of the hill-climbing search with parameter ($0 \leq c \leq 1$ in 0.5 steps) [12], [13]. MHC(c=0.5) and MHC-3 are the proposed multi-unit enabled algorithms extended from HC(c=0.5) and MHC-3, respectively. greedyO means a simple greedy allocation of the received bids by the input order. SA uses simulated annealing algorithm presented in Ref. [12]. I denote the Casanova algorithm [23] as casanova and Zurel's algorithm [11] as Zurel. Also I 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 within the specified time limit.

Comparison criteria: Since it is really difficult to obtain the maximum revenue for an auction problem, I have compared algorithms with the values computed by average revenue ratio [13]. I use the same approach to evaluate performances of algorithms on single-unit auction problems.

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, I use $ratioA_a(L)$ for my comparison of algorithms on single-unit auction problems.

Unfortunately, since Zurel's approximation algorithm is designed for single-unit auction problems, the same evaluation function cannot be applied.

Here, I use another approach that is based on the optimality ratio to the best one in the average on each bid distribution.

Let A be a set of algorithms, 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 $ratioM_a(L)$ for algorithm $a \in A$ for dataset L is defined as follows:

$$ratioM_a(L) = \frac{\sum_{p \in L} revenue_a(p)}{\max_{m \in A}(\sum_{p \in L} revenue_m(p))}$$

Here, I use $ratioM_a(L)$ for my comparison of algorithms on multi-unit auction problems. I also showed the actual computation time for obtaining the approximation allocations.

Evaluating pricing performance: In addition to above-mentioned comparisons, I compared the performance of the proposed pricing mechanism. Since the pricing mechanism itself may modify the allocations, I compared the algorithms in $ratio_B$, and execution time to complete allocations and pricing.

Experiment environment: I implemented algorithms in a C program for the following experiments. I also implemented the Casanova algorithm [23] in a C program. For Zurel's algorithm, I used Zurel's C++ based implementation that is shown in Ref. [11] with the suggested value for epsilon. Also I used CPLEX Interactive Optimizer 11.0.0 (32 bit) in the experiments. The experiments were done with above implementations to examine the performance differences among algorithms. The programs were run on a Mac with Mac OS X 10.4, a CoreDuo 2.0 GHz CPU, and 2 GBytes of memory.

4.2 Results

Table 1 shows the performance *ratioA* of approximate winner determination in single-unit auction problems. Here, I expect some computational overheads to support multi-unit auction problems. However, the overheads can be seen as very small compared with the results of HC-3 and MHC-3, HC(c=0.5) and MHC(c=0.5), respectively. In both cases, the obtained results within 100 msec are slightly lower than the results of single-unit optimized algorithms. However, the results within 1,000 msec are rather slightly higher than the results obtained by single-unit optimized algorithms. Note that this result does not mean that MHC-3 and MHC(c=0.5) are always better when the time increases.

[*3] The reason why there are some missing number (e.g., L1, and L5,) is mainly the difficulty of generating the necessary number of bids by such bid distributions.
[*4] For more details about each bid distribution, see Ref. [7].
[*5] For easiness of comparisons, here I used the same distributions that appeared in Ref. [13].

Table 1 Winner Determination Performance on Single-Unit Auctions (20,000 bids-256 items).

	L2		L3		L4		L6		L7		average	
greedyL(c=0.5)	1.0002	(7.3)	0.9639	(6.8)	0.9417	(7.1)	0.9389	(6.4)	0.7403	(8.3)	0.9170	(7.2)
HC(c=0.5)-100 ms	1.0004	(100)	0.9742	(100)	0.9576	(100)	0.9532	(100)	0.8335	(100)	0.9438	(100)
HC-3-seq-100 ms	1.0004	(100)	0.9698	(100)	1.0000	(100)	0.9966	(100)	0.8723	(100)	0.9678	(100)
HC-3-para-100 ms	1.0004	(100)	0.9742	(100)	1.0001	(100)	0.9969	(100)	0.9438	(100)	0.9831	(100)
MHC(c=0.5)-100 ms	1.0004	(100)	0.9843	(100)	0.9755	(100)	0.9532	(100)	0.7980	(100)	0.9423	(100)
MHC-3-seq-100 ms	1.0004	(100)	0.9787	(100)	1.0009	(100)	0.9966	(100)	0.8238	(100)	0.9601	(100)
MHC-3-para-100 ms	1.0004	(100)	0.9844	(100)	1.0012	(100)	0.9969	(100)	0.9981	(100)	0.9784	(100)
HC(c=0.5)-1,000 ms	1.0004	(1,000)	0.9850	(1,000)	0.9757	(1,000)	0.9638	(1,000)	1.0102	(1,000)	0.9870	(1,000)
HC-3-seq-1,000 ms	1.0004	(1,000)	0.9795	(1,000)	1.0003	(1,000)	0.9975	(1,000)	1.0062	(1,000)	0.9968	(1,000)
HC-3-para-1,000 ms	1.0004	(1,000)	0.9850	(1,000)	1.0006	(1,000)	0.9985	(1,000)	1.0236	(1,000)	1.0016	(1,000)
MHC(c=0.5)-1,000 ms	1.0004	(1,000)	0.9973	(1,000)	0.9909	(1,000)	0.9638	(1,000)	0.9862	(1,000)	0.9877	(1,000)
MHC-3-seq-1,000 ms	1.0004	(1,000)	0.9914	(1,000)	1.0012	(1,000)	0.9975	(1,000)	0.9835	(1,000)	0.9947	(1,000)
MHC-3-para-1,000 ms	1.0004	(1,000)	0.9973	(1,000)	1.0012	(1,000)	0.9985	(1,000)	1.0166	(1,000)	1.0027	(1,000)
SA-1,200 ms	1.0004	(1,200)	0.9773	(1,200)	0.9594	(1,200)	0.9449	(1,200)	1.0083	(1,200)	0.9781	(1,200)
Zurel-1st	0.5710	(11,040)	0.9690	(537)	0.9983	(2,075)	0.9928	(1,715)	0.6015	(1,796)	0.8265	(3,433)
Zurel	1.0000	(13,837)	1.0000	(890)	1.0000	(4,581)	1.0000	(4,324)	1.0000	(3,720)	1.0000	(5,470)
casanova-10 ms	0.2795	(10)	0.0074	(10)	0.0282	(10)	0.0376	(10)	0.7142	(10)	0.2134	(10)
casanova-100 ms	0.7216	(100)	0.1631	(100)	0.1377	(100)	0.1921	(100)	0.7776	(100)	0.3984	(100)
casanova-1,000 ms	0.9951	(1,000)	0.6478	(1,000)	0.4095	(1,000)	0.4578	(1,000)	0.8787	(1,000)	0.6778	(1,000)
cplex-100 ms	0.0000	(288)	0.0000	(121)	0.0299	(111)	0.0000	(150)	0.0000	(119)	0.0060	(158)
cplex-333 ms	0.0000	(489)	0.0000	(393)	0.9960	(497)	0.9716	(354)	0.0000	(487)	0.3935	(444)
cplex-1,000 ms	0.0000	(1,052)	0.0000	(1,039)	0.9960	(1,143)	0.9716	(1,140)	0.0000	(2,887)	0.3935	(1,452)
cplex-3,000 ms	0.0000	(9,171)	0.9338	(3,563)	0.9964	(3,030)	0.9716	(3,077)	0.0000	(3,090)	0.5804	(4,386)

(each value in () is time in milliseconds)

Table 2 Detailed Winner Determination Performance on Multi-Unit Auctions (20,000 bids-256items, with dominated bids, stocks=16).

	MHC-3-para-100 ms		MHC-3-para-1,000 ms		greedyL(c=0.5)		greedyO		cplex-1,000 ms		cplex-3,000 ms	
L2	1.0000	(100)	1.0000	(1,000)	0.9992	(7.3)	0.4714	(1.8)	0.0000	(1,801)	0.0000	(3,547)
L3	0.9967	(100)	1.0000	(1,000)	0.9925	(6.3)	0.5222	(0.6)	0.0000	(1,143)	1.0000	(3,039)
L4	0.9949	(100)	0.9963	(1,000)	0.9200	(5.5)	0.5294	(0.5)	0.2399	(1,058)	0.0000	(1,661)
L6	0.9976	(100)	1.0000	(1,000)	0.9286	(7.8)	0.5185	(1.3)	0.0000	(1,102)	0.0000	(3,086)
L7	0.9784	(100)	1.0000	(1,000)	0.9272	(9.5)	0.4720	(0.7)	0.0000	(1,119)	0.0000	(3,043)
arbitrary	0.9897	(100)	1.0000	(1,000)	0.9276	(7.8)	0.8503	(0.9)	0.0000	(1,018)	1.0000	(3,056)
matching	0.9869	(100)	0.9882	(1,000)	0.9857	(6.0)	0.8705	(0.1)	0.9999	(711)	0.9595	(784)
paths	0.9945	(100)	1.0000	(1,000)	0.9824	(5.8)	0.7595	(0.5)	0.0056	(1,026)	0.0000	(4,495)
regions	0.9908	(100)	1.0000	(1,000)	0.9071	(7.5)	0.8039	(0.5)	0.0000	(1,017)	1.0000	(3,063)
scheduling	1.0000	(100)	1.0000	(1,000)	1.0000	(2.8)	0.9470	(0.4)	1.0000	(501)	0.3371	(501)
average	0.9930	(100)	0.9984	(1,000)	0.9570	(6.6)	0.6745	(0.7)	0.2245	(1,050)	0.4297	(2,627)

(each value in () is time in milliseconds)

Table 3 Average Winner Determination Performance on Multi-Unit Auctions (20,000 bids-256 items, with dominated bids, stocks=2, 4, 16, 256).

stocks	MHC-3-para-100 ms		MHC-3-para-1,000 ms		greedyL(c=0.5)		greedyO		cplex-1,000 ms		cplex-3,000 ms	
2	0.9669	(100)	0.9845	(1,000)	0.9122	(6.6)	0.5998	(0.4)	0.2016	(1,168)	0.6161	(2,864)
4	0.9774	(100)	0.9879	(1,000)	0.9286	(6.8)	0.6227	(0.6)	0.2952	(1,217)	0.5523	(2,658)
16	0.9930	(100)	0.9984	(1,000)	0.9570	(6.6)	0.6745	(0.7)	0.2245	(1,050)	0.4297	(2,627)
256	0.9982	(100)	0.9992	(1,000)	0.9942	(7.7)	0.8366	(1.4)	0.5000	(766)	0.5396	(1,728)

(each value in () is time in milliseconds)

Rather it depends on the settings and their differences are quite small.

Table 2 shows the performance *ratioM* of approximate winner determination in multi-unit auction problems that are extended from single-unit problems. Note that the values are represented as *ratioM*, and each auction problem has 20,000 bids for 256 kind of items. In Table 2, the shown result is the detailed performance in each bid distributions when the number of stocks for each auction problem is 16. **Table 3** shows the average results for all bid distributions when the number of stocks for each auction problem is 2, 4, 16, or 256.

Here, since I did not increase the number of bids while the number of stocks increases, the auction problems tend to be easy to be solved when the number of stocks becomes larger. Hence, in some bid distributions, both CPLEX and the proposed approach obtained the same (i.e., optimal) results and in some other cases CPLEX can obtain better results compared to the proposed approach. However, since in other bid distributions CPLEX could not even obtain any intermediate approximation results within the specified time, the greedy-based approaches (i.e., MHC-3 and greedyL(c=0.5)) obtained totally good results in the average.

Table 4 shows the results in the setting of 100,000 bids in each auction. In Table 4, It can be seen that in many bid distributions, CPLEX could not obtain intermediate approximation results since the size of each problem is much larger than the case of 20,000 bids. Also it can be seen that the optimality on MHC-3 is still high even when that of greedyL(c=0.5) has dropped on this setting.

Table 5 shows the performance *ratioM* of approximate winner determination when the proposed pricing mechanism is applied for each approximate allocation obtained by the shown approximate winner determination algorithms. The used dataset

Table 4 Average Performance on Multi-Unit Auctions (100,000 bids-256 items, with dominated bids, stocks=2, 4, 16, 256).

n.of stocks	MHC-3-para-100 ms		MHC-3-para-1,000 ms		greedyL(c=0.5)		greedyO		cplex-1,000 ms		cplex-3,000 ms	
2	0.9848	(100)	0.9943	(1,000)	0.9202	(40.2)	0.5850	(1.8)	0.0011	(1,788)	0.1119	(3,550)
4	0.9865	(100)	0.9934	(1,000)	0.9325	(40.6)	0.6044	(2.3)	0.0000	(1,784)	0.1104	(3,538)
16	0.9948	(100)	0.9987	(1,000)	0.9491	(40.8)	0.6447	(2.8)	0.0000	(1,785)	0.1093	(3,597)
256	0.9989	(100)	1.0000	(1,000)	0.9778	(44.0)	0.7275	(3.5)	0.0000	(1,787)	0.1143	(3,560)

(each value in () is time in milliseconds)

Table 5 Detailed Pricing Performance on Multi-Unit Auctions (20,000 bids-256 items, with dominated bids, stocks=16).

	MHC-3-para-100 ms		greedyL(c=0.5)		greedyO	
L2	1.0000	(157)	0.9994	(57)	0.6932	(6,057)
L3	1.0000	(744)	0.9988	(764)	0.7064	(36,503)
L4	1.0000	(414)	0.9705	(23,761)	0.8664	(66,774)
L6	1.0000	(292)	0.9497	(7,207)	0.7380	(34,904)
L7	1.0000	(475)	0.9771	(364)	0.7886	(1,091)
arbitrary	1.0000	(13,273)	0.9577	(4,071)	0.8883	(6,483)
matching	1.0000	(19,633)	0.9996	(22,137)	0.9718	(118,207)
paths	1.0000	(95,337)	0.9969	(84,245)	0.9889	(49,906)
regions	1.0000	(26,031)	0.9731	(14,288)	0.9461	(18,883)
scheduling	1.0000	(140)	1.0000	(51)	0.9663	(58)
average	1.0000	(15,650)	0.9823	(15,695)	0.8554	(33,886)

(each value in () is time in milliseconds)

is the same that is used in Table 2, but the results for CPLEX are omitted due to its low performance on the experiment setting. Although the actual execution time for the pricing mechanism deeply depends on the number of winners in each auction problem, the average total execution time on MHC-3-para-100 ms is somewhat faster than that on greedyO, and also it is slightly faster than that on greedyL(c=0.5). Furthermore, the performance *ratioM* of MHC-3-para-100 ms is higher than the others. This shows that the combination of MHC-3-para-100 ms and the proposed pricing mechanism can work better than other combinations on the experiment setting.

5. Conclusions

In this paper, I introduced a mechanism that employs an approximate allocation algorithm that is capable of handling huge size multi-unit auctions. The proposed pricing mechanism was built based on the fast approximate allocation algorithm that behaves as an approximation of VCG and also satisfies budget balance conditions and individual rationality without having single-minded bidders assumption. Throughout these experiments, I showed that the proposed allocation algorithm MHC-3 successfully produced good allocations for those problems that cannot be easily solved by ordinary LP solvers due to hard time constraints. Furthermore, the proposed combination of MHC-3 allocation algorithm and the VCG-like approximate pricing algorithm is even faster than other combinations with simple greedy allocation algorithms.

Limitations in the proposed approach include the low revenue problem for the sellers which also appears in VCG [1], [22], and the lack of detailed theoretical considerations for incentive compatibility. In Ref. [22], Lehmann et al. pointed out that an incentive compatible auction protocol should satisfy four requirements: *Exactness, Monotonicity, Critical, and Participation*. The proposed algorithm does not have enough theoretical investigations to satisfy *Monotonicity*, i.e., it guarantees that a winner should be still a winner even when its bidding price is increased in the same auction. There is a proof that a similar greedy allocation algorithm does not satisfy *Monotonicity* [15]. Furthermore, on an online scenario, an auction mechanism does not always satisfy incentive compatibility even when it employs VCG-based pricing with optimal winner allocations [25]. Obtaining an approximation allocation algorithm which guarantees *Monotonicity* by using sensitivity analysis or other related approaches (e.g., Ref. [26] etc.) is one of future work.

Acknowledgments The work was partly supported by Grants-in-Aid for Young Scientists (B) 22700142.

References

[1] Cramton, P., Shoham, Y. and Steinberg, R. (Eds.): *Combinatorial Auctions*, The MIT Press (2006).
[2] Sandholm, T., Suri, S., Gilpin, A. and Levine, D.: CABOB: A Fast Optimal Algorithm for Winner Determination in Combinatorial Auctions, *Management Science*, Vol.51, No.3, pp.374–390 (2005).
[3] McMillan, J.: Selling Spectrum Rights, *The Journal of Economic Perspectives*, Vol.8, No.3, pp.145–162 (1994).
[4] Caplice, C. and Sheffi, Y.: Combinatorial Auctions for Truckload Transportation, *Combinatorial Auctions*, Cramton, P., Shoham, Y. and Steinberg, R. (Eds.), chapter 21, pp.539–571, The MIT Press (2006).
[5] Nisan, N. and Ronen, A.: Computationally feasible VCG mechanisms, *Proc. ACM Conference on Electronic Commerce* (*EC2000*), pp.242–252 (2000).
[6] Parkes, D.C. and Shneidman, J.: Distributed Implementations of Vickrey-Clarke-Groves Mechanisms, *Proc. International Joint Conference on Autonomous Agents and Multi Agent Systems* (*AAMAS2004*), New York, USA, pp.261–268 (2004).
[7] Leyton-Brown, K., Pearson, M. and Shoham, Y.: Towards a Universal Test Suite for Combinatorial Auction Algorithms, *Proc. ACM Conference on Electronic Commerce* (*EC2000*), pp.66–76 (2000).
[8] de Vries, S. and Vohra, R.V.: Combinatorial Auctions: A Survey, *International Transactions in Operational Research*, Vol.15, No.3, pp.284–309 (2003).
[9] Fukuta, N. and Ito, T.: Toward Combinatorial Auction-based Better Electric Power Allocation on Sustainable Electric Power Systems, *Proc. International Workshop on Sustainable Enterprise Software* (*SES2011*), pp.392–399 (online), DOI: 10.1109/CEC.2011.64 (2011).
[10] Fukuta, N. and Ito, T.: An Approach to Sustainable Electric Power Allocation Using a Multi-Round Multi-Unit Combinatorial Auction, *Proc. International Workshop on Multi-agent Smart computing* (*MASmart2011*), pp.67–81 (2011).
[11] Zurel, E. and Nisan, N.: An Efficient Approximate Allocation Algorithm for Combinatorial Auctions, *Proc. 3rd ACM Conference on Electronic Commerce* (*EC2001*), pp.125–136 (2001).
[12] Fukuta, N. and Ito, T.: Towards Better Approximation of Winner Determination for Combinatorial Auctions with Large Number of Bids, *Proc. 2006 WIC/IEEE/ACM International Conference on Intelligent Agent Technology* (*IAT2006*), pp.618–621 (2006).
[13] Fukuta, N. and Ito, T.: Fine-grained Efficient Resource Allocation Using Approximated Combinatorial Auctions–A Parallel Greedy Winner Approximation for Large-scale Problems, *Web Intelligence and Agent Systems: An International Journal*, Vol.7, No.1, pp.43–63 (2009).
[14] Fukuta, N. and Ito, T.: Periodical Resource Allocation Using Approximated Combinatorial Auctions, *Proc. 2007 WIC/IEEE/ACM International Conference on Intelligent Agent Technology* (*IAT2007*), pp.434–441 (2007).
[15] Fukuta, N. and Ito, T.: Toward A Large Scale E-Market: A Greedy and Local Search based Winner Determination, *Proc. 20th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (*IEA/AIE2007*), pp.354–363 (2007).
[16] Fukuta, N. and Ito, T.: Winner Price Monotonocity for Approxi-

mated Combinatorial Auctions, *Proc. IEEE/WIC/ACM IAT International Workshop on Electronic Commerce, Business, and Services* (*ECBS2008*), pp.533–537 (2008).
[17] Fukuta, N. and Ito, T.: An Experimental Analysis of Biased Parallel Greedy Approximation for Combinatorial Auctions, *International Journal of Intelligent Information and Database Systems*, Vol.4, No.5, pp.487–508 (online), DOI: 10.1504/IJIIDS.2010.035773 (2010).
[18] Fukuta, N.: Toward a VCG-like Approximate Mechanism for Large-scale Multi-unit Combinatorial Auctions, *Proc. IEEE/ACM/WIC International Conference on Intelligent Agent Technologyb* (*IAT2011*), pp.317–322 (2011).
[19] Leyton-Brown, K., Shoham, Y. and Tennenholtz, M.: An Algorithm for Multi-Unit Combinatorial Auctions, *Proc. 17th National Conference on Artificial Intelligence* (*AAAI2000*), pp.56–61 (2000).
[20] Gonen, R. and Lehmann, D.: Optimal Solutions for Multi-Unit Combinatorial Auctions: Branch and Bound Heuristics, *Proc. ACM Conference on Electronic Commerce* (*EC2000*), pp.13–20 (2000).
[21] Hafalir, I., Ravi, R. and Sayedi, A.: Sort-Cut: A Pareto Optimal and Semi-Truthful Mechanism for Multi-Unit Auctions with Budget-Constrained Bidders (2009). July 16 2009, CMU Working Paper.
[22] Lehmann, D., O'Callaghan, L.I. and Shoham, Y.: Truth Revelation in Rapid, Approximately Efficient Combinatorial Auctions, *J. ACM*, Vol.49, pp.577–602 (2002).
[23] Hoos, H.H. and Boutilier, C.: Solving Combinatorial Auctions using Stochastic Local Search, *Proc. 17th National Conference on Artificial Intelligence* (*AAAI2000*), pp.22–29 (2000).
[24] Múalem, A. and Nisan, N.: Truthful Approximation Mechanisms for Restricted Combinatorial Auctions, *Games and Economic Behavior*, Vol.64, No.2, pp.612–631 (2008).
[25] Hajiaghayi, M.T., Kleinberg, R. and Parkes, D.C.: Adaptive Limited-Supply Online Auctions, *Proc. ACM Conference on Electronic Commerce* (*EC2004*), pp.71–80 (2004).
[26] Lai, J.K. and Parkes, D.C.: Monotone Branch and Bound Search for Restricted Combinatorial Auctions, *Proc. ACM Conference on Electronic Commerce* (*EC2012*), pp.705–722 (2012).

Appendix

A.1 Worst-case Computational Complexity of `transformToSWPM`

Here, I give a brief anaylsis about the computational complexity of `transformToSWPM`. For simplicity of discussion, the single-unit case is assumed where the number of items be k and the number of bids be N. In the function `transformToSWPM`, for each winner, it consumes $O(kN)$ of pricing computation cost since the greedy allocation [*6] takes this cost. Then, let there be w winner bids in the given allocation but no re-allocation of winners (i.e., by the line 10 of the algorithm) will happen there, its worst-case computation cost is $O(wkN)$. Here, the worst case to compute `transformToSWPM` with re-allocation of winners is the case that initially it has only one winner and then re-allocations have been done on the pricing for the last winner with only one increment of the number of winners [*7]. Therefore, its worst-case computation cost is $O(kN+2kN+3kN+...+k^2N) = O(\frac{k(k+1)}{2} \cdot kN)$ $= O(k^3N)$. When the number of the final winners is known as w, the worst-case computational cost is $O(w^2kN)$. In a multi-unit case (i.e., the number of items is k but there is l of distinguishable item types where $k > l$, it should be at most equal but typically less than the cost in the single-unit case since some items are indistinguishable so that it should have less opportunities to make re-allocations of winners.

[*6] This does not include the sorting cost of bids by a certain criteria since it should arleady be done at the initial winner allocation process.
[*7] This is because the greedy allocation produces at most only once a better result without an increment of the number of winners. Also this is a brief proof of the computability of `transformToSWPM`.

Naoki Fukuta received his B.E. and M.E. from Nagoya Institute of Technology in 1997 and 1999 respectively. He received his Doctor of Engineering from Nagoya Institute of Technology in 2002. Since April 2002, he has been working as a research associate at Shizuoka University. Since April 2007, he has been working as an assistant professor. In 2012, he received the IPSJ Yamashita SIG Research Award. His main research interests include Mobile Agents, SemanticWeb, Konwledge-based Software Engineering, Logic Programming, Applications of Auction Mechanisms, and WWW-based Intelligent Systems. He is a member of ACM (Association for Computing Machinery), IEEE-CS (IEEE Computer Society), JSAI (Japanese Society for Artificial Intelligence), IPSJ (Information Processing Society of Japan), IEICE (Institute of Electronics, Information, and Communication Engineers), JSSST (Japan Society of Software Science and Technology), and ISSJ (Information Systems Society of Japan).