Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 36 (2009) 196–208
computers & operations research
www.elsevier.com/locate/cor

A model and heuristic algorithms for multi-unit nondiscriminatory combinatorial auction☆

Ali Haydar Özer, Can Özturan*

Department of Computer Engineering, Boğaziçi University, 34342 Bebek, Istanbul, Turkey

Available online 29 August 2007

Abstract

Single unit combinatorial auction problem (CAP) and its multi-unit extension have received a lot of attention recently. This paper introduces yet another variant of CAP which generalizes the multi-unit CAP further to allow bids on collections of items that can come from bidder defined classes of items. A bidder may be indifferent to some items with different brands, specifications or qualities and consider them as substitutable. In this case, these items can be put in a class and hence the bids can be made by referring to the items in such classes. This model enables the bidder to express his requests by using less number of bids in case he does not discriminate between different items. Because of this, we call this problem multi-unit nondiscriminatory combinatorial auction (MUNCA) problem. An integer programming formulation is given for this problem. Since this problem is NP-hard, two fast heuristic algorithms have also been designed. The heuristics give quite good solutions when compared to the optimal solution.

Keywords: Combinatorial auctions; Multi-unit auctions; Combinatorial bidding; Winner determination; Integer programming; Optimization

1. Introduction

As defined in [1], an *auction* is a market institution with an explicit set of rules for determining resource allocation and prices on the basis of bids from the market participants. In the traditional *sequential auction* model, each item or indivisible bundle of items in an auction is auctioned one at a time. Winner determination is done simply by picking the highest bidder for each item. Because of its simplicity, this model has been widely used throughout the history. Most of the auctions being conducted in the world use this model. English, Dutch, First-Price Sealed-Bid and Vickrey auctions are the four major forms of the sequential auction model [2]. These forms of auctions can be extended to multi-unit case in which more than one identical or equivalent objects are auctioned [3].

Sequential auctions are appropriate to use when the value of each item is unrelated to the values of other items for every bidder. However, there may be complementarities and substitutabilities between the items [4]. Assume that in an electronic equipment auction, several different brands of televisions and video recorders are to be auctioned. The valuation of a bidder on the bundle of a television and a video recorder can be higher than the sum of the valuations of the television and the video recorder alone. So there is a complementarity between the television and the video recorder for this bidder. Conversely, the valuation of the bidder on the bundle of two different brands of televisions

☆ This work has been funded by Boğaziçi University Scientific Projects (BAP) under the Grant number #04A104.
* Corresponding author. Tel.: +90 212 359 7225; fax: +90 212 287 2461.
E-mail addresses: ozeralih@boun.edu.tr (A.H. Özer), ozturaca@boun.edu.tr (C. Özturan).

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.08.003

can be lower than the sum of the separate valuations. In this case, there is a substitutability between the televisions of different brands for this bidder. Formally, *complementarity* between items i and j exists if $g_b(\{i, j\}) > g_b(\{i\}) + g_b(\{j\})$ where $g_b(S)$ is the gain of getting a set of items S for a bidder b and *substitutability* between the items i and j exists if $g_b(\{i, j\}) < g_b(\{i\}) + g_b(\{j\})$.

If there are complementarities between different items, sequential auctions may provide inefficient allocation. Although *parallel auction* model in which the items are auctioned simultaneously can increase the allocation efficiency by reducing uncertainty, this does not help us in solving *complementarity problem* that involves complementarities between the items [5]. *Combinatorial auction* model solves this problem by allowing bidders to submit bids on combinations of different items [6,7]. In this model, all items are available to the bidders and the bidders are free to express their own valuations of any combination of items. The combinatorial auction model is applicable to many real-world situations such as the auctions for radio spectrum rights [8], airport slot allocations [9], transportation services [10–12], course registrations [13], and commercial time slot allocations [14]. Among these, probably the most famous auctions are the Federal Communications Commission (FCC) electromagnetic spectrum auctions [15–18]. Since 1994, FCC has conducted more than 50 auctions in the form of simultaneous multiple-round (SMR) auctions. In an SMR auction, licenses that have complementarities are available for bidding in parallel. The auction is conducted in successive rounds and the lengths of the rounds are announced by FCC. After each round, results are processed and made public. Until the next round, bidders go over their bid strategies and adjust their bids if necessary. The auction ends when no new bid is submitted during a round. Although package bidding is not allowed in the SMR auctions, as announced by FCC, in auction #31 (Upper 700 MHz Band) package bidding would be allowed. This FCC auction, however, has not been conducted so far. Porter et al. [19] discusses the issues encountered in this FCC auction that played a role in its being not adopted. In general, among the issues cited by Porter et al. [19] are computational uncertainty and bidding complexity. This paper partly contributes to the resolution of these issues by presenting fast heuristic algorithms for the multi-unit auction problem and a compact bid representation.

In the combinatorial auction model, it is possible to solve the *substitutability* problem by introducing dummy items for each substitutable item [4]. The role of the dummy items is to allow bidders to express exclusive-or relationship between the bids. For instance, if a bidder wants to get one television of brand A or B, one dummy item should be introduced and two separate bids, one bid for the combination of the dummy item and the television of brand A and another bid for the combination of the dummy item and the television of brand B must be submitted by the bidder. Although this trick helps solving the substitutability problem, the number of bids increases combinatorially with the number of substitutable items.

The combinatorial auction model provides economically better allocation of items at the expense of computational difficulty. Winner determination problem, combinatorial auction problem (CAP), as formulated by Rothkopf et al. [20] and Sandholm [5] is an instance of the weighted set packing problem (and also the weighted independent set problem) which is known to be NP-complete [20–22]. For unrestricted CAP, many optimum search algorithms have been proposed [4,5,23–26]. It must be noted that although some specialized algorithms may perform better in some test cases when compared to the commercial general purpose mixed integer programming (MIP) solvers such as CPLEX [27], in other cases they fall behind [23,24,26]. Therefore, general purpose MIP solvers can be considered as a good choice among the optimum solvers for CAP. For a detailed discussion about optimum search techniques see [6].

The single unit combinatorial auction model provides economically efficient allocations when the bidders are interested in bundles of items. However, it is inappropriate for situations where multiple instances of items are auctioned. Since a bidder is not interested in a particular item, he must bid separately to all combinations of the items he wants. For instance, if the bidder wants to get 100 keyboards and 100 mice out of 200 keyboards and 300 mice in a single unit combinatorial auction, he must bid $\binom{200}{100} \cdot \binom{300}{100}$ times. The multi-unit combinatorial auction (MUCA) model solves this problem by representing identical items as multiple units of a single item and allowing bidders to bid on instances of the items [28–31]. For further discussion on combinatorial auctions, the reader is referred to [7].

If multiple instances of items are to be auctioned, the MUCA model provides efficient bid representation. Although the substitutability problem between identical units of items is solved in this model, possible substitutabilities between different items are not considered. If a bidder does not differentiate two or more different items, the MUCA model becomes insufficient in representing such preferences. As an example, assume that in a MUCA, the items to be auctioned are 200 wired PS/2 keyboards, 400 wired USB keyboards, 300 wireless PS/2 keyboards and 100 wireless

USB keyboards. Consider the following two scenarios:

- A bidder wants to buy 100 PS/2 keyboards without differentiating wired and wireless models. In order to express this preference, one dummy item must be introduced and $\binom{2+100-1}{100} = 101$ bids must be placed.
- Likewise, if the bidder wants to buy 100 keyboards without differentiating any four keyboard types, one dummy item must be introduced and $\binom{4+100-1}{100} = 176,851$ bids must be placed.

In order to overcome this inefficiency in preference expression, we propose a new auction model called the *multi-unit nondiscriminatory combinatorial auction* (MUNCA) model. In this proposed model, bidders may encode their preferences of substitutability into bids easily by declaring a list of nondiscriminatory items. In the above example, the MUNCA model will enable the bidder to express his preferences by placing only one bid in each of the above scenarios.

The rest of the paper is organized as follows: In Section 2, the MUNCA model is defined and the winner determination problem of the MUNCA model is formulated. Since the MUNCA problem is known to be NP-hard (by a trivial reduction from the independent set problem [32]), two fast heuristic algorithms were developed in order to solve it. In Section 3, we present these heuristic solvers as well as a linear relaxation based greedy heuristic solver. In order to estimate the performance of our solvers an artificial test case generator was developed. Details of the generator can be found in Section 4. In Section 5, we present the performances of our solvers on various tests.

2. Multi-unit nondiscriminatory combinatorial auction model

In this section, we describe our MUNCA model by first defining the MUNCA problem formally and then presenting an integer programming (IP) formulation of this problem. We were motivated to develop the MUNCA model in the context of the recently emerging grid computing technologies. Grids will enable sharing of various resources such as computational power, data and devices in the so called virtual organizations. Models for grid economies [33,34] are needed that will facilitate assignment of resources to interested parties in the grid. An auction market is one of the models that is proposed for resource management and scheduling in grids. Harchol-Balter [35], for example, proposes the use of combinatorial auctions for resource scheduling on the TeraGrid [36] which is the world's largest supercomputing grid. In the computer industry, there are also planned and/or current practices of the use of auctions for resource allocation on grids [37,38]. Since in grids, we can have multi-unit resources with different brands (the best example being the presence of many processors from many different manufacturers), we were motivated to propose the MUNCA model in this paper. Even though our own motivation for developing the MUNCA model has originated from the grid computing field, we should note that application of this model is not just confined to this field. It is also applicable to other industry examples that we mentioned in Section 1. To exemplify our model, we present an example involving resource assignments in grids at the end of this section.

2.1. Definition of the MUNCA problem

Let $R = \{r_1, r_2, \ldots, r_m\}$ be a set of m different items and let $U = \{u_1, u_2, \ldots, u_m\}$ be a set of units of items where u_i is the number of available identical units of the item r_i $(1 \leqslant i \leqslant m, u_i \in \mathbb{Z}^+)$. Let $B = \{b_1, b_2, \ldots, b_n\}$ be a set of submitted bids. A bid consists of two parts: a list of subbids and an offered price to be paid if the bid is satisfied. There is a logical AND relationship between the subbids because a bid is satisfiable if *all* of its subbids are satisfiable. Similarly a subbid consists of two parts, a set of substitutable items and requested quantity of instances from this set. There is a logical OR relationship between the items in this set because a subbid is satisfiable if the requested number of instances can be supplied from the items in *any* subset of this set. So if the size of this set is more than one, it means that the bidder treats all of the listed items as equivalent.

Let v be the maximum number of subbids among all bids. Then, formally a bid $b_j = \{(\langle s_{j1}, q_{j1}\rangle, \langle s_{j2}, q_{j2}\rangle, \ldots, \langle s_{jt_j}, q_{jt_j}\rangle), p_j\}$ is defined as a combination of subbids and an offered price p_j where $s_{jk} \subseteq R$ is the set of requested substitutable items and q_{jk} is the requested quantity of instances for the set s_{jk} $(1 \leqslant j \leqslant n, 1 \leqslant k \leqslant t_j \leqslant v, p_j \in \mathbb{R}^+)$. The MUNCA problem is defined as the problem of finding a subset B_s of the bid set B and the corresponding allocation of items that maximizes the sum of the offered prices of the bids in B_s while preserving the item quantity limits in U.

2.2. IP formulation of the MUNCA problem

In order to formulate this problem as an IP problem, two new variables x and y must be introduced. In this formulation x_j is a 0–1 integer variable that represents whether a bid is satisfied (1) or not (0) and $y_{jk}^{(i)}$ is a natural number that represents how many units of item i are taken by kth subbid of bid j if it is satisfied ($1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, $1 \leqslant k \leqslant v$). Our formulation is given as

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \sum_{k=1}^{v} y_{jk}^{(i)} \leqslant u_i \quad (1 \leqslant i \leqslant m), \tag{2}$$

$$\left(\sum_{i=1}^{m} y_{jk}^{(i)} \right) - q_{jk} x_j = 0 \quad (1 \leqslant j \leqslant n, 1 \leqslant k \leqslant v), \tag{3}$$

$$y_{jk}^{(i)} = 0 \quad (1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n, 1 \leqslant k \leqslant v, r_i \notin s_{jk}), \tag{4}$$

$$x_j \in \{0, 1\} \quad (1 \leqslant j \leqslant n), \tag{5}$$

$$y_{jk}^{(i)} \in \mathbb{N} \quad (1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n, 1 \leqslant k \leqslant v). \tag{6}$$

In this formulation, the objective line in Eq. (1) ensures that the maximum revenue is gained from the auction. Eq. (2) constrains the sum of the requested quantity of item i by all subbids with the number of available units of item i. Eq. (3) constrains the sum of the quantities for each item inside a subbid to be equal to the requested quantity in that subbid if bid j is satisfied, otherwise $y_{jk}^{(i)}$ values are cleared to 0. In this equation, it is also ensured that if a bid is satisfied, then all the subbids inside that bid are also satisfied. In Eq. (4), the $y_{jk}^{(i)}$ value is set to 0 if item i is not requested by the kth subbid of the jth bid.

The MUNCA model supports OR-bids with dummy items like in the (OR^*) bidding language [4,39]. In this language, bidders can introduce dummy (phantom) items in their bids in order to represent different preferences in the form of exclusive-or bids. In [39], it is proven that OR^* bidding language is at least as powerful as *OR*, *XOR*, *OR-of-XOR* and *XOR-of-OR* bidding languages.

2.3. MUNCA problem example

In order to make the definition and the formulation clear, we will present a simple MUNCA example from Grid economy domain [32]. Assume that in a small computational grid, the following resources are available for rental for a predefined period of time:

- 10 Intel workstations (*intel*).
- 10 AMD workstations (*amd*).
- 20 Sun workstations (*sun*).
- A license server with 5 MATLAB licenses (*matlab*) and 5 CPLEX licenses (*cplex*).
- 10 GB (in 1 GB chunks) storage space in an external storage server (*storage*).

Given these resources, the first bidder requests 10 Intel workstations, 5 MATLAB licenses and 4 GB storage space for \$1000. The second bidder requests 10 Intel or AMD workstations and 5 CPLEX licenses for \$600 and the last bidder requests 30 workstations and 5 GB storage space for \$1500. So for this scenario, $R = \{intel, amd, sun, matlab, cplex, storage\}$, $U = \{10, 10, 20, 5, 5, 10\}$ and the submitted bids are:

- $b_1 = \{(\langle\{intel\}, 10\rangle, \langle\{matlab\}, 5\rangle, \langle\{storage\}, 4\rangle), 1000\}$.
- $b_2 = \{(\langle\{intel, amd\}, 10\rangle, \langle\{cplex\}, 5\rangle), 600\}$.
- $b_3 = \{(\langle\{intel, amd, sun\}, 30\rangle, \langle\{storage\}, 5\rangle), 1500\}$.

This example can be formulated as follows (for simplicity we omitted variables and constraints if $y_{jk}^{(i)} = 0$):

$$
\begin{aligned}
\text{maximize} \quad & 1000x_1 + 600x_2 + 1500x_3 \\
\text{subject to} \quad & y_{11}^{(1)} + y_{21}^{(1)} + y_{31}^{(1)} \leqslant 10, \\
& y_{21}^{(2)} + y_{31}^{(2)} \leqslant 10, \\
& y_{31}^{(3)} \leqslant 20, \\
& y_{12}^{(4)} \leqslant 5, \\
& y_{22}^{(5)} \leqslant 5, \\
& y_{13}^{(6)} + y_{32}^{(6)} \leqslant 10, \\
& y_{11}^{(1)} - 10x_1 = 0, \\
& y_{12}^{(4)} - 5x_1 = 0, \\
& y_{13}^{(6)} - 4x_1 = 0, \\
& y_{21}^{(1)} + y_{21}^{(2)} - 10x_2 = 0, \\
& y_{22}^{(5)} - 5x_2 = 0, \\
& y_{31}^{(1)} + y_{31}^{(2)} + y_{31}^{(3)} - 30x_3 = 0, \\
& y_{32}^{(6)} - 5x_3 = 0, \\
& x_1, x_2, x_3 \in \{0, 1\}, \\
& y_{jk}^{(i)} \in \mathbb{N} \quad (\text{for all } i, j, k).
\end{aligned}
$$

3. MUNCA problem solvers

We have developed four MUNCA problem solvers:

(i) Optimum solver (OPT),
(ii) Linear relaxation based greedy heuristic solver (LRS),
(iii) Greedy heuristic solver based on price per unit criteria (PS) and
(iv) Enhanced heuristic solver based on price per unit criteria (EPS).

OPT solves the MUNCA problem optimally by calling the commercial CPLEX MIP solver for the IP problem described in Section 2.2.

Linear relaxation based methods are widely used for bounding NP-hard problems because of their simplicity and effectiveness. Although our main concerns and contributions in this work are OPT, PS and EPS solvers, we include the solutions of the relaxed problem generated by CPLEX and LRS in order to show how tight the linear relaxation based bounds for the MUNCA problem are. The following sections describe the details of LRS and the two price per unit criteria based PS and EPS heuristics. The solver package can be downloaded from [40].

3.1. Linear relaxation based greedy heuristic solver

LRS finds a feasible solution to the MUNCA problem by solving the linear relaxation of the IP formulation and running a greedy heuristic based on the results of this solution. LRS algorithm is based on the assumption that after solving the linear relaxation of the problem, bids with higher x_j values are more likely to be in the optimum solution (note that, in the relaxed problem, x_j is a continuous variable with $x_j \in [0, 1]$).

This algorithm consists of two phases, a ranking phase and an allocation phase. In the ranking phase, the linear relaxation of the problem is solved and then the bids are sorted according to x_j values in descending order. In the allocation phase, we start by adding the first bid in the sorted list to an empty set of winning bids and check whether this set is feasible or not. If the set is feasible, we continue adding the next bid in the list to the set. If the set is not feasible,

```
Algorithm LRS
Input: MUNCA problem instance
Output: winningBids

/* Ranking Phase */
solve the LP relaxation of MUNCA problem and get x_j values for each bid
sort the bids according to x_j in descending order and store in orderedBids

/* Allocation Phase */
winningBids = ∅
for j = 1 to n
    add orderedBids_j to winningBids
    if checkFeasibility(winningBids) = false then
        remove orderedBids_j from winningBids
end for
return winningBids
```

Fig. 1. The pseudocode for the LRS algorithm.

we remove the last added bid from the set and then add the next bid. Then, we check the feasibility of the set again. The procedure continues in this manner until all the bids in the sorted list are processed. After the end of the procedure, the winning bids set is returned. The pseudocode for the LRS algorithm is presented in Fig. 1. *checkFeasibility* function is explained in the following section.

3.2. Checking the feasibility of a given bid set

For all the presented greedy heuristics, checking the feasibility of a given set of bids is an essential step. Unlike in the MUCA model, checking feasibility in the MUNCA model is not so trivial because of the ORed items in the subbids. In order to check the feasibility of a given bid set $B = \{b_1, b_2, \ldots, b_n\}$, the following set of equations must be solved:

$$\sum_{j=1}^{n}\sum_{k=1}^{v} y_{jk}^{i} \leqslant u_i \quad (1 \leqslant i \leqslant m),$$

$$\left(\sum_{i=1}^{m} y_{jk}^{i}\right) - q_{jk} = 0 \quad (1 \leqslant j \leqslant n, 1 \leqslant k \leqslant v),$$

$$y_{jk}^{i} = 0 \quad (1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n, 1 \leqslant k \leqslant v, r_i \notin s_{jk}),$$

$$y_{jk}^{i} \in \mathbb{N} \quad (1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n, 1 \leqslant k \leqslant v).$$

We model this problem as a feasible network flow problem [41]. The network associated with this model is shown in Fig. 2. This network is constructed as follows: Let $N(V, A, l, u, b)$ denote our network with node set V, arc set A, lower bounds $l(v, w)$ and upper bounds $u(v, w)$ for each arc $(v, w) \in A$, and source/demand values $b(v)$ for each node $v \in V$. We begin constructing the network by representing each bid b_j as one node and drawing an arc from a source node s, to the bid node b_j with infinite upper bound $u(s, b_j) = +\infty$ and zero lower bound $l(s, b_j) = 0$ for $1 \leqslant j \leqslant n$. Then, for each node b_j with $1 \leqslant j \leqslant n$, we introduce s_{jk} nodes with $1 \leqslant k \leqslant t_j$ where t_j is the number of subbids inside the bid b_j, and draw the t_j arcs (b_j, s_{jk}). We impose fixed flow requirements of q_{jk} on these arcs, i.e., $u(b_j, s_{jk}) = l(b_j, s_{jk}) = q_{jk}$. These arcs with fixed flow values ensure that each subbid gets the requested number of instances. After that we add one node, r_i, for each item in R $(1 \leqslant i \leqslant m)$ and draw the arcs (s_{jk}, r_i) if item i is requested in subbid k of bid b_j. There are no flow constraints on these arcs, i.e., $u(s_{jk}, r_i) = +\infty$ and $l(s_{jk}, r_i) = 0$. These arcs ensure that only the requested items for each subbid are allocated by that subbid. Finally, we add one arc from each item node r_i to the sink node t with upper bound of u_i and lower bound of zero. This last set of arcs limits the number of units of items. Supply/demand values for internal nodes are set to zero, i.e., $b(v) = 0$ for $v \in V - \{s, t\}$. Supply value for the source node s is set as $b(s) = q_{\text{sum}} = \sum_{j=1}^{n}\sum_{k=1}^{t_j} q_{jk}$ amount of flow and the demand value for the sink node t is set to the negative of the supply of the source node, i.e., $b(t) = -q_{\text{sum}}$.

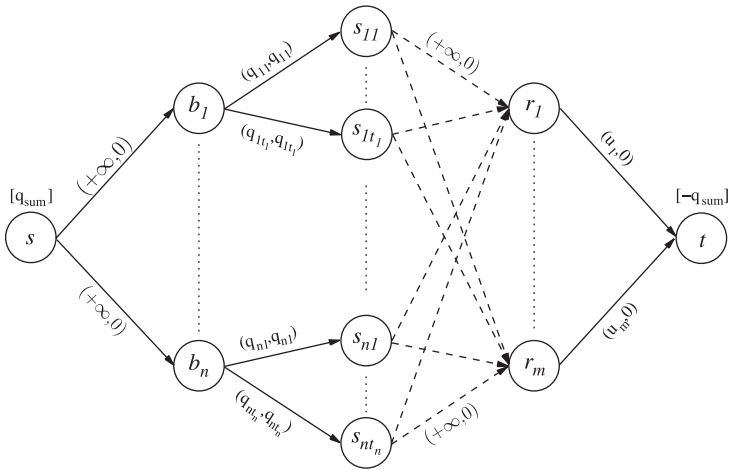

Fig. 2. Network flow diagram for checking the feasibility of a given set of bids.

Algorithm PS
Input: MUNCA problem instance
Output: $winningBids$

/* Ranking Phase */
for each bid j
  $qSum_j = 0$
  for each subbid k of bid j
    $qSum_j = qSum_j + q_{jk}$
  end for
  $h_j = \frac{p_j}{qSum_j}$
end for
sort the bids according to h_j in descending order and store in $orderedBids$

/* Allocation Phase */
$winningBids = \emptyset$
for $j = 1$ **to** n
  add $orderedBids_j$ to $winningBids$
  if checkFeasibility($winningBids$) = false **then**
    remove $orderedBids_j$ from $winningBids$
end for
return $winningBids$

Fig. 3. The pseudocode for the PS algorithm.

If there is a feasible flow in the constructed network N, then we can conclude that the given set of bids is also feasible. After solving this network flow problem, the amount of flow, f_{s_{jk},r_i}, between nodes s_{jk} and r_i gives the $y_{jk}^{(i)}$ values, i.e., $(y_{jk}^{(i)} = f_{s_{jk},r_i})$.

3.3. Greedy heuristic solver based on price per unit criteria

Like LRS, the PS algorithm is also based on greedy allocation of bids. However, this heuristic is based on the assumption that the bids with high prices and small number of requested items are most likely to be in the optimum solution. Combining these two criteria, the bids with higher price per unit of items should be preferred. Therefore, for sorting bids, price per unit criterion is used instead of linear relaxation solution.

```
Algorithm EPS
Input: MUNCA problem instance
Output: maxBids

maxPrice = 0;
for β = 0.9 to 1.1 step 0.05 /* and_factor */
   for α = 0.9 to 1.1 step 0.05 /* or_factor */

      /* Ranking Phase */
      for each bid j
         qSum_j = 0
         for each subbid k of bid j
            qSum_j = qSum_j + q_jk · α^(|s_jk|−1)
         end for
         h_j = p_j / (qSum_j · β^(t_j − 1))
      end for
      sort the bids according to h_j in descending order and store
         in orderedBids

      /* Allocation Phase */
      winningBids = ∅
      for j = 1 to n
         add orderedBids_j to winningBids
         if checkFeasibility(winningBids) = false then
            remove orderedBids_j from winningBids
      end for
      if total_price(winningBids) > maxPrice then
         maxPrice = total_price(winningBids)
         maxBids = winningBids
      end if
   end for
end for
return maxBids
```

Fig. 4. The pseudocode for the EPS algorithm.

This algorithm also consists of two phases, the ranking phase and the allocation phase. In the ranking phase of the algorithm, we first calculate heuristic value of each bid j using the following formula:

$$h_j = \frac{p_j}{\sum_{k=1}^{t_j} q_{jk}}. \tag{7}$$

Then, we sort the bids according to these h_j heuristic values in descending order. The rest of the algorithm, i.e., the allocation phase, is the same as that of the LRS algorithm and is described in Section 3.1. The pseudocode for the PS algorithm is presented in Fig. 3.

3.4. Enhanced heuristic solver based on price per unit criteria

In the PS algorithm, the bids with higher price per unit are favored. Price per unit is a general criterion that can also be applied to other combinatorial auction types. It does not benefit from any information specific to the MUNCA model such as the information related to the subbids of bids. In the EPS heuristic, the assumption in the PS heuristic is extended so that the bids containing small number of subbids but containing large number of items in their subbids are favored. To facilitate this favoring, we change the ranking scheme by introducing two new factors called or_factor and and_factor.

In the EPS algorithm, for each or_factor (α) and and_factor (β) pair, the heuristic values of the bids are calculated using the new ranking formula:

$$h_j = \frac{p_j}{\sum_{k=1}^{t_j}(q_{jk} \cdot \alpha^{|s_{jk}|-1}) \cdot \beta^{t_j-1}}. \tag{8}$$

Assuming the or_factor is less than one and the and_factor is greater than one, the heuristic value of the bid increases if the number of requested item types ($|s_{jk}|$) is increased. In contrast, higher numbers of subbids (t_j) cause the heuristic value to decrease. The effects of the or_factor and the and_factor increase geometrically with $|s_{jk}|$ and t_j.

After the calculation of the heuristic values, the bids are sorted according to these values in descending order and the allocation procedure is applied. Among the solutions found for each or_factor and and_factor pair, the best solution is returned.

This algorithm increases the chance of finding better solution by changing the or_factor and the and_factor in the range of [0.9, 1.1] with 0.05 increments. The ranges of [0.9, 1) and (1, 1.1] for the or_factor and the and_factor, respectively, are for the instances for which the assumption of EPS holds. The rest of the range, although it seems to contradict with the assumption of this heuristic, is for the exceptional problem instances.

When the or_factor and the and_factor are set to 1, this algorithm gives the same result as that of the PS algorithm. Therefore, the EPS heuristic is guaranteed to give better solutions than the PS heuristic. The complexity of the algorithm is $O(25 \cdot CPS)$, where CPS is the complexity of the PS algorithm. The pseudocode for the EPS heuristic is presented in Fig. 4. As seen in this algorithm, the bodies of the two nested for-loops are roughly the same as that of PS. Each of the two for-loops is repeated 5 times, leading to 25 iterations that execute the body of the loops.

4. MUNCA problem test case generator

We were not able to locate any public real-world data that we could use in order to test our MUNCA model. Therefore, we have coded a test case generator in order to prepare an artificial test suite for measuring the performance of the algorithms. The generator is capable of producing test cases for full-factorial testing in which all possible combinations of all factors can be tested. It supports uniform, normal and exponential distribution types and uses GNU Scientific Library (GSL) [42] for generating pseudo-random numbers.

The configuration parameters of the generator are:

- *number_of_instances*: defines how many set of instances will be generated,
- *m*: defines how many items will be generated,
- *u*: defines the number of available units for each item,
- *n*: defines how many bids will be generated for each problem instance,
- *t*: defines how many subbids will be generated for each bid,
- *s*: defines the size of requested subset of items for each subbid,
- s_jk_method: defines the method for generating requested items for subbids,
- *q*: defines the requested number of items for each subbid inside all bids,
- or_factor: defines the price factor for ORed item requests inside a subbid,
- and_factor: defines the price factor for ANDed subbids inside a bid,
- $price\_stdev$: defines the standard deviation factor used for calculating the bid price.

The algorithm employed by the generator is straightforward and its source code can be obtained from [40]. However, the methods for determining items inside a subbid and the price of a bid need further explanation.

There are two methods for generating requested items for subbids. The first method (called *uniform random method*) chooses items of subbids randomly from all available items. The second method (called *neighborhood method*) chooses the first item of a subbid randomly among all items and chooses the remaining items from the neighbors (in terms of item index) of the chosen item. For instance, let $m = 10$ and for a subbid let $s = 5$ and $index\ of\ chosen\ item = 4$, then the list of requested items are {2,3,4,5,6}. The motivation for the neighborhood method is as follows: Substitutable items are very likely to be indexed consecutively or closely. Since substitutable items are also very likely to be chosen inside the subbids, then our method is justified for choosing items from the neighbors of the chosen item. For instance,

Price Determination Algorithm
Input: or_factor, and_factor, price_stdev
Output: price for given bid

for each item type i
  $unitPrice_i = uniform(0,1)$
end for
for each bid j
  $bidPrice_j = 0$
  for each subbid k of bid j
    $subbidPrice_k = 0$
    for each item type l in subbid k
      $subbidPrice_k = subbidPrice_k + q_{jk} \cdot unitPrice_l \cdot (u_l / \sum_{z \in s_{jk}} u_z)$
    end for
    $subbidPrice_k = subbidPrice_k \cdot \alpha^{|s_{jk}|-1}$
    $bidPrice_j = bidPrice_j + subbidPrice_k$
  end for
  $bidPrice_j = bidPrice_j \cdot \beta^{t_j - 1}$
  $bidPrice_j = normal(\mu = bidPrice_j, \sigma = (bidPrice_j \cdot price\_stdev/100))$
end for

Fig. 5. The pseudocode for determining the price of a bid.

auctions containing items with different brands and auctions for the resources in computational grid in which computers reside in a high bandwidth network are indexed consecutively.

Assignment of proper prices for bids is quite important for generating realistic test cases. In the generator, after determining the number of items, a uniform random number between 0 and 1 is chosen as the price of one unit of each item. In order to determine the price of a bid, we first find the price of each subbid in the bid. Raw price of a subbid is determined by multiplying requested quantity of items with the *weighted average price* of ORed items inside the subbid. In order to favor substitutability between the items, we multiply the raw subbid price with $\alpha^{|s_{jk}|-1}$ where α is the or_factor and $|s_{jk}|$ is the number of items inside the subbid. This produces the price of the subbid. Then, we sum up the prices of subbids inside a bid and in order to disfavor complementarities between subbids, we multiply this value with β^{t_j-1} and produce the raw bid price where β is the and_factor and t_j is the number of subbids inside the bid. Finally, we draw a normally distributed random number with mean the raw bid price, and standard deviation calculated using $price\_stdev$ parameter. We assign this number as the price of the bid. The pseudocode for determining the price of a bid is presented in Fig. 5.

5. Experimental Results

We have prepared a test suite that consists of 1692 problem instances based on uniform, normal and exponential distributions. The problem instances are generated using 39 different configuration files. In order to be able to compare the quality of the heuristics with the optimum solution, small to mid-size problem instances are generated. In the test suite, the number of items, *m*, varies between 10 and 100 and the number of bids, *n*, varies between 50 and 500. The test suite and the configuration files can be obtained from [40].

The tests are conducted using a dedicated AMD Athlon 64 3200+ based workstation with 1 GB memory. For presenting quality of the heuristic solutions, goodness values relative to the optimum solution is used. Goodness values of solutions for each configuration are calculated using the following formula:

$$\frac{Solution\ Of\ Solver\ s\ for\ Distribution\ d}{Optimum\ Solution\ for\ Distribution\ d} \times 100, \tag{9}$$

where *s* is either LRS, PS or EPS and *d* is either uniform, normal or exponential. Running time values are recorded using wall clock time in seconds. Maximum running time for the optimum solver (CPLEX MIP solver) is set to 3 h. For 1596 out of the 1692 instances, CPLEX finds optimum solutions. For 93 instances, CPLEX finds integer feasible

Table 1
The linear relaxation based upper bounds and the goodness values of the solvers relative to the optimum solutions (%)

Distribution	# Tests	Lin. rel. based upper bound		LRS		PS		EPS	
		Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
Uniform	545	106.88	10.34	89.39	13.14	97.67	3.34	99.30	1.31
Normal	507	105.70	6.75	92.81	9.96	96.71	4.24	98.99	1.60
Exponential	544	130.35	39.79	75.26	21.76	97.15	4.37	99.05	1.72
Overall	1596	114.51	26.83	85.66	17.59	97.19	4.02	99.12	1.56

Table 2
The running times of the solvers (s)

Distribution	OPT		LRS		PS		EPS	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
Uniform	288.57	930.35	1.44	2.02	0.07	0.03	1.59	0.91
Normal	515.31	1435.60	1.23	1.66	0.06	0.03	1.56	0.77
Exponential	207.72	758.45	1.88	2.03	0.07	0.04	1.82	0.97
Overall	333.04	1077.71	1.52	1.93	0.07	0.04	1.66	0.89

solutions which are not necessarily optimal and for the remaining three instances, CPLEX cannot find any integer feasible solution in 3 h. In this experiment, only the results of optimally solved instances are evaluated. The mean goodness values of the heuristic solvers and the corresponding standard deviations can be seen in Table 1. Also in this table, the solutions of the relaxed problem generated by CPLEX, i.e., the upper bounds, are included in order to show how tight the linear relaxation based bounds for MUNCA problem are. The running times of the solvers are presented in Table 2.

The linear relaxation based upper bounds for the test cases are approximately 15% higher than the optimum solutions on average. It is observed that, for the instances generated using uniform and normal distributions, the upper bounds are relatively tighter than the instances generated using exponential distribution. Likewise the linear relaxation based solver, LRS, produces solutions approximately 15% lower than the optimum solutions. The distribution based results of LRS resemble the results of the linear relaxation based upper bounds. For the uniform and normal distribution based instances, the results are relatively better, with 7–10% loss on average, than the exponential distribution based instances with approximately 25% loss.

Examining the goodness values of the PS and EPS heuristics, average performance of the PS heuristic is quite promising with 97% relative to the optimum solver with a standard deviation of approximately 4%. By the nature of its design, EPS heuristic is guaranteed to give results that are at least as good as that of PS. It is observed that extra search done in EPS pays off and on average EPS produces results better than 99% of the optimum. The corresponding standard deviation is less than 2%. Note that the distribution based performances of PS and EPS, unlike that of LRS, are quite stable and do not differ considerably.

In terms of running times, the PS heuristic runs quite fast and finds solutions in 0.07 s on average. As expected from the complexity analysis, the mean running time of the EPS heuristic is 1.66 s with standard deviation of less than 1 s. Comparing the running times of EPS and LRS, mean running times of both heuristics are very close proving that EPS heuristic is not much slower than the LRS heuristic. If we compare running times of the optimum solver and our heuristics in general, we see that our heuristics are considerably faster than the optimum solver. Considering the fact that our test instances were generally small sized, we can expect that for larger sized instances, the difference between the running times of our heuristics and the optimum solver will widen even further.

6. Conclusion

In this paper, we have presented a new combinatorial auction model that extends the well-known MUCA model. When bidders are indifferent to some items in an auction, the MUCA model becomes inefficient in representing the

preferences of the bidders. The MUNCA model overcomes this inefficiency by allowing the bidders to value any combination of items according to their preferences. After defining the MUNCA model, we formulated the winner determination problem as an integer program. Since this problem is a generalization of the maximum independent set problem, the winner determination problem of the MUNCA model is NP-hard. Although most of the problem instances of moderate size are solvable optimally by using general purpose MIP solvers, we developed two fast greedy heuristic solvers, namely, the PS and the EPS solvers for difficult problem instances. Using artificial test data, the performances of these solvers were compared with the optimum and the linear relaxation based heuristic solvers. Our two heuristic solvers were able to obtain quite good results in the tests with 97% and 99% average performances relative to the optimum solution, respectively.

References

[1] McAfee RP, McMillan J. Auctions and bidding. Journal of Economic Literature 1987;25(2):699–738.
[2] Vickrey W. Counterspeculation auctions and competitive sealed tenders. The Journal of Finance 1961;16(1):8–37.
[3] Krishna V. Auction theory. San Diego, CA, USA: Academic Press; 2002.
[4] Fujishima Y, Leyton-Brown K, Shoham Y. Taming the computational complexity of combinatorial auctions: optimal and approximate approaches. In: IJCAI '99: Proceedings of the sixteenth international joint conference on artificial intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc; 1999. p. 548–53.
[5] Sandholm T. Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence 2002;135(1-2):1–54.
[6] In: Cramton P, Shoham Y, Steinberg R, editors. Combinatorial auctions. Cambridge, MA, USA: MIT Press; 2006.
[7] de Vries S, Vohra RV. Combinatorial auctions: a survey. INFORMS Journal on Computing 2003;15(3):284–309.
[8] Jackson C. Technology for spectrum markets. PhD thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA; 1976.
[9] Rassenti SJ, Smith VL, Bulfin RL. A combinatorial auction mechanism for airport time slot allocation. The Bell Journal of Economics 1982;13(2):402–17.
[10] Caplice CG. An optimization based bidding process: a new framework for shipper–carrier relationships. PhD thesis, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA; 1996.
[11] Ledyard JO, Olson M, Porter D, Swanson JA, Torma DP. The first use of a combined-value auction for transportation services. Interfaces 2002;32(5):4–12.
[12] Sheffi Y. Combinatorial auctions in the procurement of transportation services. Interfaces 2004;34(4):245–52.
[13] Graves RL, Schrage L, Sankaran JK. An auction method for course registration. Interfaces 1993;23(5):81–92.
[14] Jones JL. Incompletely specified combinatorial auction: an alternative allocation mechanism for business-to-business negotiations. PhD thesis, Warrington College of Business Administration, University of Florida, Gainesville, FL, USA; 2000.
[15] The Federal Communications Commission website. URL: ⟨http://www.fcc.gov/⟩; 2007 (accessed on July 30, 2007).
[16] Cramton P. Spectrum auctions. In: Cave M, Majumdar S, Vogelsang I, editors. Handbook of telecommunications economics. Amsterdam, The Netherlands: Elsevier Science; 2002. p. 605–39.
[17] McMillan J. Selling spectrum rights. Journal of Economic Perspectives 1994;8(3):145–62.
[18] Milgrom P. Putting auction theory to work: the simultaneous ascending auction. The Journal of Political Economy 2000;108(2):245–72.
[19] Porter D, Rassenti S, Roopnarine A, Smith V. Combinatorial auction design. Proceedings of the National Academy of Sciences 2003;100(19):11153–7.
[20] Rothkopf MH, Pekeč A, Harstad RM. Computationally manageable combinational auctions. Management Science 1998;44(8):1131–47.
[21] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of np-completeness. San Francisco, CA, USA: WH Freeman and Co; 1979.
[22] Karp R. Reducibility among combinatorial problems. In: Miller R, Thatcher J, editors. Complexity of computer computations. New York, NY, USA: Plenum Press; 1972. p. 85–103.
[23] Andersson A, Tenhunen M, Ygge F. Integer programming for combinatorial auction winner determination. In: ICMAS '00: Proceedings of the fourth international conference on multiagent systems ICMAS-2000. Washington, DC, USA: IEEE Computer Society; 2000. p. 39–46.
[24] Günlük O, Ladànyi L, de Vries S. A branch-and-price algorithm and new test problems for spectrum auctions. Management Science 2005;51(3):391–406.
[25] Sandholm T, Suri S. Bob: improved winner determination in combinatorial auctions and generalizations. Artificial Intelligence 2003;145(1–2): 33–58.
[26] Sandholm T, Suri S, Gilpin A, Levine D. Cabob: a fast optimal algorithm for winner determination in combinatorial auctions. Management Science 2005;51(3):374–90.
[27] ILOG CPLEX. URL: ⟨http://www.ilog.com/products/cplex/⟩; 2007 (accessed on July 30, 2007).
[28] Bartal Y, Gonen R, Nisan N. Incentive compatible multi unit combinatorial auctions. In: TARK '03: Proceedings of the 9th conference on theoretical aspects of rationality and knowledge. New York, NY, USA: ACM Press; 2003. p. 72–87.
[29] Gonen R, Lehmann D. Optimal solutions for multi-unit combinatorial auctions: branch and bound heuristics. In: EC'00: Proceedings of the 2nd ACM conference on electronic commerce. New York, NY, USA: ACM Press; 2000. p. 13–20.
[30] Leyton-Brown K, Shoham Y, Tennenholtz M. An algorithm for multi-unit combinatorial auctions. In: Proceedings of the seventeenth national conference on artificial intelligence and twelfth conference on innovative applications of artificial intelligence. California, CA, USA: AAAI Press/The MIT Press; 2000. p. 56–61.

[31] Sandholm T, Suri S, Gilpin A, Levine D. Winner determination in combinatorial auction generalizations. In: AAMAS '02: Proceedings of the first international joint conference on autonomous agents and multiagent systems. New York, NY, USA: ACM Press; 2002. p. 69–76.
[32] Özer AH. Combinatorial auction based resource co-allocation model for grids. MSc thesis, Computer Engineering Department, Boğaziçi University, Istanbul, Turkey; 2004.
[33] Buyya R, Abramson D, Venugopal S. The grid economy. Proceedings of the IEEE, vol. 93. New York, NY, USA: IEEE; 2005. p. 698–714.
[34] Z. Tan, Market-based grid resource allocation using a stable continuous double auction. PhD thesis, Department of Computer Science, University of Manchester, Manchester, UK; 2007.
[35] Harchol-Balter M. Auction based scheduling for the TeraGrid. URL: ⟨http://www.cs.cmu.edu/∼harchol/⟩; 2007 (accessed on July 30, 2007).
[36] TeraGrid. URL: ⟨http://www.teragrid.org⟩; 2007 (accessed on July 30, 2007).
[37] Sun opens processor auction house. URL: ⟨http://theregister.co.uk/2005/02/03/sun_grid_two⟩; 2005 (accessed on July 30, 2007).
[38] Tycoon: market based resource allocation. URL: ⟨http://www.hpl.hp.com/research/tycoon/⟩; 2007 (accessed on July 30, 2007).
[39] Nisan N. Bidding and allocation in combinatorial auctions. In: EC '00: Proceedings of the 2nd ACM conference on electronic commerce. New York, NY, USA: ACM Press; 2000. p. 1–12.
[40] Software and supplementary materials for MUNCA problem. URL: ⟨http://www.cmpe.boun.edu.tr/∼ozer/pubs/munca/⟩ alternatively ⟨http://www.ahozer.com/en/pubs/munca/⟩; 2007 (accessed on July 30, 2007).
[41] Ahuja RK, Magnanti TL, Orlin JB. Network flows: theory algorithms and applications. New Jersey, NJ, USA: Prentice-Hall; 1993.
[42] GNU Scientific Library. URL: ⟨http://www.gnu.org/software/gsl/⟩; 2007 (accessed on July 30, 2007).