A Test Suite for the Evaluation of Mixed Multi-Unit Combinatorial Auctions

Meritxell Vinyals, Andrea Giovannucci, Jesús Cerquides,
Pedro Meseguer, Juan A. Rodriguez-Aguilar

PII: S0196-6774(08)00010-2
DOI: 10.1016/j.jalgor.2008.02.008
Reference: YJAGM 1445

To appear in: *Journal of Algorithms*

Received date: 22 October 2007
Revised date: 3 December 2007
Accepted date: 5 February 2008

Please cite this article as: M. Vinyals, A. Giovannucci, J. Cerquides, P. Meseguer, J.A. Rodriguez-Aguilar, A Test Suite for the Evaluation of Mixed Multi-Unit Combinatorial Auctions, *Journal of Algorithms* (2008), doi: 10.1016/j.jalgor.2008.02.008

A Test Suite for the Evaluation of Mixed Multi-Unit Combinatorial Auctions

Meritxell Vinyals [a] Andrea Giovannucci [a] Jesús Cerquides [b]
Pedro Meseguer [a] Juan A. Rodriguez-Aguilar [a]

[a] *IIIA, Artificial Intelligence Research Institute*
CSIC, Spanish National Research Council
Campus UAB, 08193 Bellaterra, Spain
{meritxell/andrea/pedro/jar}@iiia.csic.es

[b] *WAI, Dep. Matemàtica Aplicada i Anàlisi*
Universitat de Barcelona
Gran Via 585, 08191 Barcelona, Spain
cerquide@maia.ub.es

Abstract

Mixed Multi-Unit Combinatorial Auctions extend and generalize all the preceding types of combinatorial auctions. In this paper, we try to make headway on the practical application of MMUCAs by: (1) providing an algorithm to generate artificial data that is representative of the sort of scenarios a winner determination algorithm is likely to encounter; and (2) subsequently assessing the performance of an Integer Programming implementation of MMUCA in CPLEX.

1 Introduction

A combinatorial auction (CA) is an auction where bidders can buy (or sell) entire bundles of goods in a single transaction [1]. Selling goods in bundles has the great advantage of eliminating the risk for a bidder of not being able to obtain complementary goods at a reasonable price in a follow-up auction (think of a CA for a pair of shoes, as opposed to two consecutive single-item auctions for each of the individual shoes). The study of the mathematical, game-theoretical and algorithmic properties of CAs has recently become a popular research topic in AI. This is due not only to their relevance to important application areas such as electronic commerce or supply chain management, but also to the range of deep research questions raised by this auction model.

Central to CAs are the issues of *winner determination problem* (WDP) and *bidding*. Winner determination is the problem, faced by the auctioneer, of choosing which goods to award to which bidder so as to maximize its revenue. The (decision problem underlying the) WDP for standard CAs is known to be $\mathcal{NP}$-complete, with respect to the number of goods [2]. $\mathcal{NP}$-hardness can, for instance, be shown by reduction from the well-known Set Packing problem. Bidding is the process of transmitting one's valuation function over the set of goods on offer to the auctioneer through some *bidding language* [3]. Under an *OR-language*, if a particular bidder submits several atomic bids (a bundle together with a proposed price), then the auctioneer may accept any set of bids from that bidder for which the bundles do not overlap, and charge the sum of the specified prices. If we use an *XOR-language* instead, that means that only one of the atomic bids can be accepted. The advantage of an XOR-language is that it allows to express not only *complementarity* between goods (value of a bundle being greater than the values of its parts), like an OR-language, but also *substitutability* (value of a bundle being less than the sum of its parts). Although an XOR bidding language is known to be fully expressive [3,4], using this language in some types of CAs causes that finding a feasible solution is $\mathcal{NP}$-complete [5].

In [4] we introduce a generalization of the standard model of CA. This new auction model integrates direct and reverse auctions, i.e. the auctioneer can buy and sell goods within a single auction. It also incorporates the idea of *transformability* relationships between goods by allowing agents not only to bid for goods but also for *transformation services*, i.e. an agent may submit a bid offering to transform a certain set of goods into another set of goods. We call the resulting auction model *mixed multi-unit combinatorial auctions* (MMUCA). To illustrate the operation of MMUCA, consider as an example the assembly of a car's engine, whose structure is depicted in Figure 1 (a). Notice that each part in the diagram, in turn, is produced from further components or raw materials. For instance, a cylinder ring (part 8) is produced by transforming some amount of stainless steel with the aid of an appropriate machine. Therefore, there are several production levels involved in the making of a car's engine. A MMUCA allows to run an auction where bidders can bid over bundles of parts, bundles of transformations, or any combination of parts and transformations. Notice that the result of an MMUCA WDP algorithm would be an ordered sequence of bids making explicit how bidders coordinate to progressively transform goods until producing engines as final products. Therefore, an MMUCA would allow to assemble a supply chain from bids.

Despite its potential for application, and unlike CAs, little is known about the practical application of MMUCAs since no empirical results have been reported on any WD algorithms. These results are unlikely to come up unless researchers are provided with algorithms or test suites to generate artificial data that are representative of the auction scenarios a WD algorithm is likely

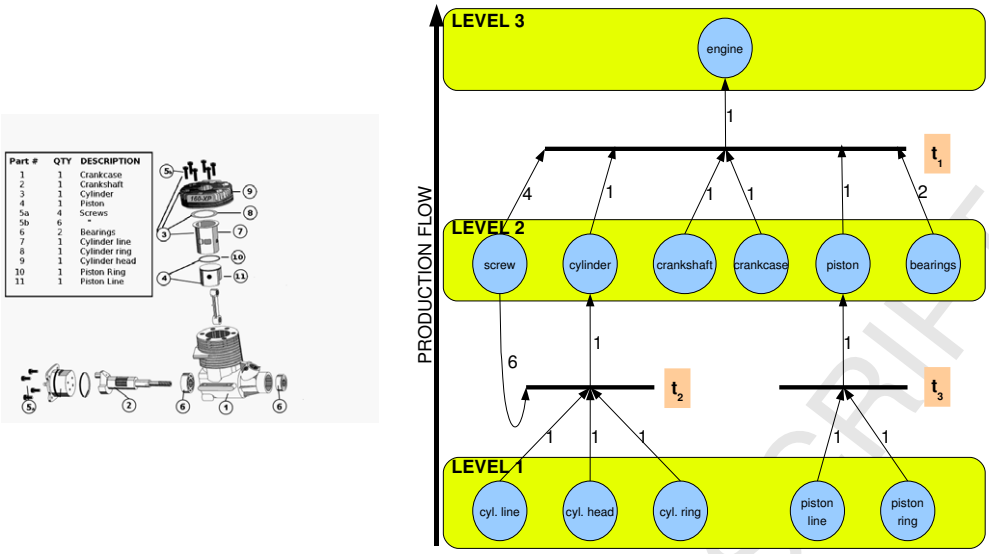


Fig. 1. (a) Components of a car engine. (b) Supply chain for a car's engine.

to encounter. Hence, WD algorithms could be accurately tested, compared, and improved. In this paper, we try to contribute to the practical application of MMUCAs in the following lines.

We provide an algorithm to generate artificial data sets that are representative of the sort of scenarios a WD algorithm is likely to encounter. Our algorithm takes inspiration on the structure of supply chains, whose formation has been identified as a very promising application domain for MMUCA [6,4]. In order to understand the basic operation of our algorithm, consider the sample of supply chain depicted in Figure 1 (b) illustrating the goods (represented as circles) and transformations (represented as horizontal bars) involved in the assembly of a car engine. A supply chain is composed of levels (or *tiers* if we employ the supply chain terminology). In Figure 1 (b) the supply chain has three levels. Each level contains goods that are subsequently transformed into other goods within another level in the supply chain. There are three transformations, namely t_1, t_2, t_3. While t_2 and t_3 transform goods from level 1 into goods in level 2, t_1 transforms goods from level 2 into goods in level 3. For instance, t_3 transforms 1 unit of piston line and 1 unit of piston ring into 1 piston. Within this setting, our generator allows to flexibly generate:

- *Supply chain structures with a varying number of levels.* Modelling from complex supply chains involving multiple levels (e.g. the making of a car) to simple supply chains involving a few parties.
- *Transformations of varying complexity.* Varying complexity on the input and output sides of the transformations involved in a supply chain. For instance, consider t_2, the production of cylinders in Figure 1 (b). It involves screws, cylinder lines, cylinder rings, and cylinder heads. Thus, on the input side

it requires more material goods than t_3 when producing pistons. However, both t_2 and t_3 produce a single output good.

- *Transformations representing different production structures.* The input and output goods from a transformation may come from different levels. For instance, in Figure 1 (b) t_2 transforms goods from level 1 into level 2 following the production flow. However, t_2 requires screws from level 2 and goods from level 1 to produce goods for level 2. As a particular case, notice that a transformation disassembling an engine in level 3 into its pieces in level 1 would cause a loop in the supply chain caused by the combination of assembly and disassembly processes. Notice that the fact that a transformation like, for instance, t_1 accepts input goods from level 1 allows to model that t_1 *overrides the intermediaries* producing goods for level 2.
- *Bids per level.* Different bid distributions may appear at different levels, to control the degree of *competition* in the market.

We employ such algorithm to generate artificial data and subsequently assess the performance of an Integer Programming (IP) implementation of a WDP solver for MMUCA in CPLEX.

The paper is structured as follows. In section 2 we provide some background on CAs, which is extended to MMUCAs in section 3. Next, in section 4 we analyze the required features of an artificial data set generator for MMUCAs whose algorithm is detailed in section 5. In sections 6 and 7, we analyze some empirical results, draw some conclusions and outline paths to future research.

2 Combinatorial Auctions

A *combinatorial auction* (CA) [1] is an auction where bidders can buy (or sell) entire bundles of goods in a single transaction. A CA is *single-unit* when there is a single copy of each item at auction (each item has multiplicity one). We say that a CA is *multi-unit* when there are multiple copies of some item(s). Although CAs are very complex computationally [7], the fact that bidders can express their preferences over bundles of goods may help both bidders and auctioneer to obtain better deals. In fact, buying items in bundles has the great advantage of eliminating the risk for a bidder of not being able to sell complementary items at a reasonable price in a follow-up auction (think of a CA to acquire a pair of shoes, as opposed to two consecutive single-item auctions for each of the individual shoes). Indeed, CAs may lead to more efficient allocations whenever complementarities among the goods at auction hold.

CAs have a high potential to be employed as an allocation mechanism in a wide variety of real-world domains. Thus, they have been proposed to be employed

for allocating loads to trucks in the transportation market [8], routes to buses [9], goods/services to buyers/providers in industrial procurement scenarios [10], airport arrival and departure slots [11], radio-frequency spectrum for wireless communications services [12], and supply chain formation [13].

Auction theory studies the formal properties of auctions as shown in the surveys of [14] and [15]. Nonetheless CAs have recently attracted the attention of economists and game theorists. Associated to auction theory is also the design of auction mechanisms, devoted to study *how* to run an auction in order to guarantee some economic properties such as, for instance, efficiency, incentive compatibility, individual rationality, etc. For CA mechanisms see [16], [17], [18], [19], [20], and [21].

2.1 Bidding Languages

Bidding is the process of transmitting one's valuation function over the set of goods on offer to the auctioneer (or rather *some* valuation function: the bidders are of course not required to reveal their true valuation). In principle, it does not matter how the valuation function is being encoded, as long as sender (bidder) and receiver (auctioneer) agree on the semantics of what is being transmitted, *i.e.* as long as the auctioneer can understand the message(s) sent by the bidder. Indeed, it is possible to fully specify an auction mechanism (allocation and pricing rules) without reference to a concrete bidding language. In practice, however, the choice of bidding language is of central importance. Since there are an exponential number of combinations of goods, a bidding language must allow bidders to express their bids in a compact way. In addition, the complexity of the problem of allocating the goods at auction to bidders depends on the choice of the bidding language (e.g. [22] contains an interesting discussion on the topic).

Early work on CAs has typically ignored the issue of bidding languages. The standard assumption used to be that if a particular bidder submits several atomic bids (a bundle together with a proposed price), then the auctioneer may accept any set of bids from that bidder for which the bundles do not overlap, and charge the sum of the specified prices. This is now sometimes called the *OR-language*. But other interpretations of a set of atomic bids are possible. For instance, we may take it to mean that the auctioneer may accept at most one bid per bidder; this is now known as the *XOR-language*.

The first systematic study of bidding languages is due to Nisan [3] (an early version appeared in 2000). Nisan's paper provides an excellent introduction to the topic and has clarified a number of issues that have previously remained somewhat fuzzy.

2.2 Winner Determination Problem

The winner determination problem (WDP) considers choosing which goods to award to which bidder so as to maximise the auctioneer revenue. WDP for CAs is a complex computational problem.

One of the fundamental issues limiting the applicability of CAs to real-world scenarios is the computational complexity associated to the winner determination problem. In particular, it has been proved that WDP is NP-complete [2]. General *integer programming* (IP) solvers [23] and special purpose algorithms (e.g. [5], [24],and [25]) have been employed to solve WDP, but it is well known that it does not exist a general solver that performs well in all situations. For an extended review on WDP and related issues refer to [26],[27], and [28].

3 Mixed Multi-unit Combinatorial Auctions

In this section we firstly summarise the work in [4], which introduces mixed multi-unit combinatorial auctions as a generalisation of the standard model of CA and discusses the issues of bidding and winner determination. Next, we describe the features of the proposed bidding language, designed to allow bidders to express several types of complex bids. Finally, we summarise the features of an efficient IP solver for the WDP of this new type of auction, fully described in [6].

3.1 Bidding Language

Let G be the finite set of all the types of goods. A *transformation* is a pair of multisets over G: $(\mathcal{I}, \mathcal{O}) \in \mathbb{N}^G \times \mathbb{N}^G$. An agent offering the transformation $(\mathcal{I}, \mathcal{O})$ declares that it can deliver $\mathcal{O}$ *after* having received $\mathcal{I}$. In our setting, bidders can offer any number of such transformations, including several copies of the same transformation. That is, agents will be negotiating over *multisets of transformations* $\mathcal{D} \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$. For example, $\{(\{\,\}, \{a\}), (\{b\}, \{c\})\}$ means that the agent in question is able to deliver a (no input required) and that it is able to deliver c if provided with b. Note that this is not the same as $\{(\{b\}, \{a, c\})\}$. In the former case, if another agent is able to produce b if provided with a, we can use the second transformation and get c; in the latter case this would not work.

In a MMUCA, agents negotiate over bundles of transformations. Hence, a *valuation* $v : \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)} \to \mathbb{R}$ is a (typically partial) mapping from multisets of

transformations to real numbers. Intuitively, $v(\mathcal{D}) = p$ means that the agent equipped with valuation v is willing to make a payment of p in return for being allocated all the transformations in $\mathcal{D}$ (in case p is a negative number, this means that the agent will accept the deal if it *receives* an amount of $|p|$). For instance, valuation $v(\{(\{line, ring, head, 6 \cdot screws, screwdriver\}, \{cylinder, screwdriver\})\}) = -10$ means that some agent can assemble a cylinder for \$10 when provided with a cylinder line, a cylinder ring, a cylinder head, six screws, and a screwdriver, and returns the screwdriver once done [1].

An *atomic bid* $b = (\{(\mathcal{I}^1, \mathcal{O}^1), \ldots, (\mathcal{I}^n, \mathcal{O}^n)\}, p, l)$ specifies a finite multiset of finite transformations, a price p, and a bid owner identifier l. To make the semantics of such an atomic bid precise, we need to decide whether or not we want to make a *free disposal* assumption. We can distinguish two types of free disposal. As to free disposal *at the bidder's side*, there are two possible free disposals sub-types: *good free disposal* and *transformation free disposal*. *Good free disposal* means that a bidder would always be prepared to accept more goods and give fewer goods away, without requiring a change in payment; whereas *transformation free disposal* means that it is allowed that some transformations are sold but not employed in the transformation process. As to free disposal *at the auctioneer's side*, we only have *good free disposal*, meaning that the auctioneer may accept more and give away fewer goods. Both these free disposals affect the definition of what constitutes a valid solution to the WDP.

A suitable *bidding language* should allow a bidder to encode choices between alternative bids and the like [3]. Informally, an OR-combination of several bids means that the bidder would be happy to accept any number of the sub-bids specified, if paid the sum of the associated prices. An XOR-combination of bids expresses that the bidder is prepared to accept at most one of them. For the formal definition of WDP below, we restrict ourselves to bids in the XOR-language, which is known to be fully expressive for MMUCAs [4].

Bids in MMUCAs are composed of transformations. Each transformation expressses either an offer to buy, to sell, or to transform some good(s) into (an)other good(s). Thus, transformations are the building blocks composing bids. We can readily classify the types of transformations over which agents bid as follows:

(1) **Output transformations** are those with no input good(s). Thus, an O-transformation represents a bidder's offer to sell some good(s). Besides, an O-transformation is equivalent to a bid in a reverse CA.
(2) **Input transformations** are those with no output good(s). Thus, an I-transformation represents a bidder's offer to buy some good(s). Notice

[1] We use $6 \cdot screws$ as a shorthand to represent six identical elements in the multiset.

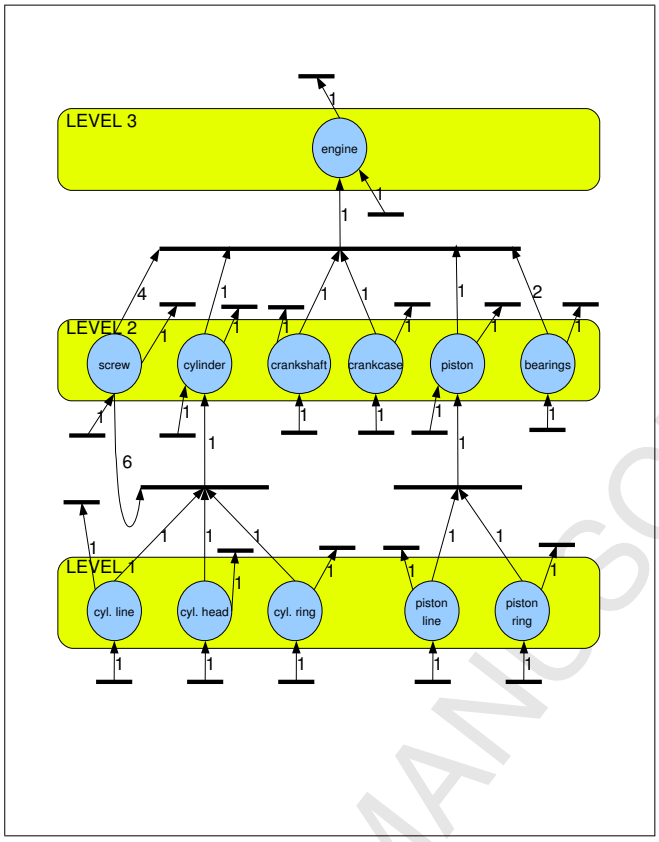


Fig. 2. All market transformations for a car's engine. It is indicated the number of units of each item required to execute each transformation.

that an I-transformation is equivalent to a bid in a direct CA.

(3) **Input-Output transformations** are those whose input and output good(s) are not empty. An IO-transformation stands for a bidder's offer to deliver some good(s) after receiving some other good(s): *I can deliver* $\mathcal{O}$ *after having received* $\mathcal{I}$. They can model a wide range of different processes in real-world situations (e.g. assembly, transformation, or exchange).

Figure 2 presents samples of each transformation type. Horizontal, black bars stand for transformations, circles stand for goods, and directed arrows from goods into or from transformations represent the goods input into or produced out of a transformation. Thus, for instance, we differentiate an I-transformation to consume a piston, an O-transformation to give away a piston, and an IO-transformation giving away a piston after receiving a piston ring and a piston line. Notice that any bid in a MMUCA results as a combination of transformations of the above-listed types.

3.2 Winner Determination Problem. Informal Definition

The *input* to WDP consists of a complex bid expression for each bidder, a multiset $\mathcal{U}_{in}$ of goods the auctioneer holds to begin with, and a multiset $\mathcal{U}_{out}$ of goods the auctioneer expects to end up with.

In standard CAs, a solution to the WDP is a set of atomic bids to accept. In our setting, however, the *order* in which the auctioneer "uses" the accepted transformations matters. For instance, if the auctioneer holds a to begin with, then checking whether accepting the two bids $Bid_1 = (\{(\{a\}, \{b\})\}, 10, id_1)$ and $Bid_2 = (\{(\{b\}, \{c\})\}, 20, id_2)$ is feasible involves realising that we have to use Bid_1 before Bid_2. Thus, a *solution* for WDP will be a *sequence of transformations*. A *valid* solution has to meet two conditions:

(1) *Bidder constraints:* The multiset of transformations in the sequence has to *respect the bids* submitted by the bidders. This is a standard requirement. For instance, if a bidder submits an XOR-combination of transformations, at most one of them may be accepted. With no transformation free disposal, if a bidder submits an offer over a bundle of *transformations*, all of them must be employed in the transformation sequence, whereas in the case of transformation free disposal any number of the transformations in the bundle can be included into the solution sequence, but the price to be paid is the total price of the bid.
(2) *Auctioneer constraints:* The sequence of transformations has to be *implementable:* (a) check that $\mathcal{U}_{in}$ is a superset of the input set of the first transformation; (b) then update the set of goods held by the auctioneer after each transformation and check that it is a superset of the input set of the next transformation; (c) finally check that the set of items held by the auctioneer in the end is a superset (the same set in the case of no good free disposal) of $\mathcal{U}_{out}$.

An *optimal* solution is a valid solution that maximises the sum of prices associated with the atomic bids selected.

3.3 Connected Component IP Solver (CCIP)

In this section we summarise CCIP, a mapping of the MMUCA WDP into an IP formulation, as thoroughly discussed in [6]. In section 3.3.1 we outline the intuitions underlying CCIP, whereas in section 3.3.2 we provide a detailed IP formulation. For further details, we recommend the interested reader to consult [6] and [29].

Position	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	t_0	t_2			t_1		t_4				
Sequence 2	t_0	t_1	t_2	t_4							
Sequence 3	t_2	t_1	t_0	t_4							
Solution template	t_0	t_1	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_5	t_9	t_{10}	t_6 t_7	t_6 t_7	t_8

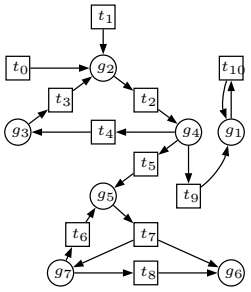
Table 1
Partial sequences of transformations.

3.3.1 Foundations

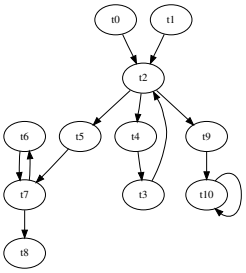
Consider that after receiving a bunch of bids, we draw the relationships among goods and transformations, as shown in Figure 3 (a). There, we represent goods at trade as circles, transformations as squares, a transformation input goods as incoming arrows and its output goods as outgoing arrows. Thus, for instance, transformation t_0 offers one unit of good g_2 and transformation t_2 transforms one unit of g_2 into one unit of g_4. Say that the auctioneer requires $\mathcal{U}_{out} = \{g_2, g_3\}$. Row 1 in table 1 stands for a valid solution sequence. Indeed, it stands for a valid solution sequence because at each position, enough input goods are available to perform the following transformation. Notice too that likewise row 1, row 2 also stands for a valid solution sequence because even though they differ in the ordering among transformations, both use exactly the same transformations, and both have enough goods available at each position. However, row 3 in table 1 is not a valid sequence, although it contains the same transformations, because t_2 lacks of enough input goods (g_2) to be used.

In Figure 3 (a), it is clear that transformations that have no input goods can be used prior to any other transformation. Thus, transformations t_0 and t_1 can come first in the solution sequence. Moreover, we can *impose* that t_0 comes before t_1 because swapping the two would yield an equivalent solution. If we now consider transformations t_2, t_3, t_4, we observe that: (i) they *depend* on the output goods of t_0 and t_1; and (ii) we cannot impose an arbitrary order among them because they form a cycle and then they can feed with input goods one another (they depend on one another). However, no permutation of the three can be discarded for the valid solution sequence. Furthermore, whatever their order, we can always use them before transformations t_5 and t_9 (since these depend on g_4) without losing solutions.

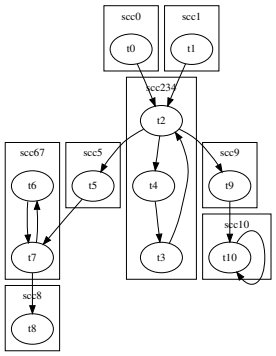
Assuming that the auctioneer does not care about the ordering of a solution sequence as long as enough goods are available for every transformation in the sequence, we can impose "a priori" constraints on the ordering of trans-



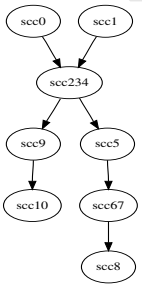
(a) A bid set



(b) TDG



(c) SCCs of the TDG



(d) The strict order

Fig. 3. An MMUCA bid set, the corresponding TDG, SCC, and Order Relation.

formations without losing solutions. The way of imposing such constraints is via a *solution template*, a pattern that any candidate sequence must fulfil to be considered as a solution. For instance, row 4 in table 1 shows a sample of solution template. A solution sequence *fulfilling* that template must have transformations t_0 in position 1 and t_1 in position 2, whereas it is free to assign positions 3, 4, or 5, to the transformations in $\{t_2, t_3, t_4\}$. For instance, row 3 of table 1 does not fulfil the template in row 4, whereas rows 1 and 2 do.

Notice that the constraints in the solution template derive from our analysis of the dependence relationships among transformations. Hence, in order to build a solution template, we must firstly analyse the dependence relationships among transformations to subsequently use them to constrain the positions at which a transformation can be used.

At this aim, we can follow the sequential process illustrated in Figure 3:

(1) Define the so-called transformation dependency graph (TDG), a graph where two transformations t and t' are connected by an edge if they

have a good that is both output of t and input to t' (direct dependence). Figure 3 (b) depicts the TDG for the bids represented in Figure 3(a).

(2) Assess the *strongly connected components* (SCC) of the TDG. Depending on the received bids, the TDG may or may not contain strongly connected components. In order to constrain the position of transformations, we transform the TDG in an acyclic graph where the nodes that form a strongly connected component are collapsed into a single node. The main idea is that the positions of transformations in a strongly connected component are drawn from the same set of positions, but we cannot impose any order regarding the position each transformation takes on. In Figure 3(c) we identify strongly connected components or SCCs in the graph. In Figure 3(d) we can see the graph resulting from transforming (collapsing) each SCC into a node.

In [29], we prove that if there is a strict order among transformations (e.g. like the one depicted in Figure 3(d)), we can always construct a solution template that restricts the positions that can be assigned to those transformations in a way that, if a solution sequence fulfils the solution template, the strict order is also fulfilled. For instance, consider the solution template in row 4 in table 1 that we construct considering the strict order in Figure 3(d). Since the solution sequences in rows 1 and 2 of table 1 fulfil the solution template in row 4, they both fulfil the strict order.

Now we are ready to characterise valid solutions to the MMUCA WDP. Looking back at the solution sequences in rows 1 and 2 of table 1, we realise that both are *partial sequences*. A partial sequence is a sequence with "holes", meaning that there can be some positions in the sequence that are *empty*. Formally, a *partial sequence* over a non-empty finite set M is a *partial* function $K : \{1, \ldots, n\} \rightarrow M$, with $n \in \mathbb{N}$. Therefore, a valid solution to the MMUCA WDP can be encoded as a partial sequence of transformations that *fulfils* some solution template.

Henceforth we employ S to denote a solution template. Intuitively, a solution template is a function that maps each position to a finite set of transformations that can take on such position in the solution sequence. For instance, in table 1 we see that $S(3) = [t_2]$ where $[t_2]$ stands for the strongly connected component to which t_2 belongs to (namely, the one containing $\{t_2, t_3, t_4\}$). We employ S^{-1} to indicate the inverse of a solution template S. $S^{-1}([t])$ indicates the set of positions that map to the strongly connected component containing t via S. For instance, in table 1 we see that $S^{-1}([t_2]) = \{3, 4, 5\}$.

3.3.2 Detailed IP Formulation

Hereafter, we restrict ourselves to bids in the XOR-language, which is known to be fully expressive for MMUCAs [4]. However, we can easily extend the results to other bidding languages, in particular languages including an OR-operator. Let B be the set of all atomic bids. Recall that an atomic bid $b = (\mathcal{D}_b, p_b, l_b)$ consists of a multiset of transformations, a price, and a label indicating the owner of the bid, i.e. $\mathcal{D}_b \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$, $p_b \in \mathbb{R}$, and $l_b \in L$ where L is a set of bidders and B_l is the set of all bids submitted by bidder $l \in L$.

For each bid b, let t_{bk} be a unique label for the kth transformation in the *underlying set of elements* in $\mathcal{D}_b$ (for some arbitrary but fixed ordering over such set) [2]. Let T_b be the set of all t_{bk} for bid b. Let T be the set of all t_{bk}, namely the set of all distinguishable transformations (no copies of the very same transformation) mentioned anywhere in the bids. Let $(\mathcal{I}_{bk}, \mathcal{O}_{bk})$ be the actual transformation labelled by t_{bk}.

The auctioneer has to decide which transformations to accept and in which order to implement them. At this aim, we represent each solution with a partial sequence $J : \{1, \ldots, |\mathcal{D}|\} \rightarrow T$. We employ the following decision variables : x^m_{bk} will take on value 1 only if transformation t_{bk} is selected at the m-th position within the solution sequence (i.e. $J(m) = t_{bk}$). Let S be the solution template resulting from the strict order generated by the TDG. For solver CCIP we only allow as solutions partial sequences fulfilling S, imposing that no transformation can hold positions out of the one specified by S, i.e. $x^m_{bk} = 0 \; \forall m \notin S^{-1}([t_{bk}])$.

Furthermore, we employ the following auxiliary decision variables: x_b is a binary variable that takes value 1 if bid b is accepted; and x_{bk} is an integer variable that represents the multiplicity of transformation t_{bk} (namely, the number of positions that transformation t_{bk} holds in the solution sequence).

Let $(\mathcal{I}^m, \mathcal{O}^m)$ be the mth transformation in the solution sequence, i.e. the t_{bk} such that $x^m_{bk} = 1$. Say that we represent with the multiset of goods $\mathcal{M}^m$ the quantity of resources available to the auctioneer after performing m transformations. Since $\mathcal{U}_{in}$ represents the auctioneer's stock, we have that $\mathcal{M}^0 = \mathcal{U}_{in}$. For the remaining positions, the following relationship holds:

$$\mathcal{M}^m(g) = \mathcal{M}^{m-1}(g) + \mathcal{O}^m(g) - \mathcal{I}^m(g) \qquad \forall g \in G \qquad (1)$$

[2] Within set theory, a multiset can be formally defined as a pair (A, m) where A is some set and $m : A \rightarrow N$ is a function from A to the set $N = \{1, 2, 3, \cdots\}$ of (positive) natural numbers. The set A is called the *underlying set of elements*. For each a in A the multiplicity (that is, number of occurrences) of a is the number $m(a)$. In our case, the underlying set of in $\mathcal{D}_b$ will be a finite set of transformations without copies of the very same transformation.

because enacting transformation $(\mathcal{I}^m, \mathcal{O}^m)$ consumes the goods in $\mathcal{I}^m$ and produces the goods in $\mathcal{O}^m$. For instance, say that the auctioneer begins with $\mathcal{U}_{in} = \{a, a, d, d\}$. If we apply the first transformation $(\mathcal{I}^1, \mathcal{O}^1) = (\{a, a\}, \{c\})$ (from two units of a produce one unit of c), the auctioneer ends up with $\mathcal{M}^1 = \{c, d, d\}$.

Note that $\mathcal{I}^m$ and $\mathcal{O}^m$ can be assessed from our decision variables as:

$$\mathcal{I}^m(g) = \sum_{b \in B} \sum_{k=1}^{|T_b|} x_{bk}^m \cdot \mathcal{I}_{bk}(g) \qquad \forall g \in G \tag{2}$$

$$\mathcal{O}^m(g) = \sum_{b \in B} \sum_{k=1}^{|T_b|} x_{bk}^m \cdot \mathcal{O}_{bk}(g) \qquad \forall g \in G \tag{3}$$

Hence, equation (1) can be unfolded into the equation:

$$\mathcal{M}^m(g) = \mathcal{U}_{in}(g) + \sum_{i=1}^{m} \sum_{b \in B} \sum_{k=1}^{|T_b|} x_{bk}^i \cdot (\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g))$$

Now, we are ready to explicitly state the constraints that a valid solution has to fulfil in solver CCIP.

(1) Variable x_{bk} contains the number of times that t_{bk} is selected in the solution sequence. x_{bk} is obtained by summing up x_{bk}^m over the positions m assigned to $[t_{bk}]$ ($m \in S^{-1}([t_{bk}])$):

$$x_{bk} = \sum_{m \in S^{-1}([t_{bk}])} x_{bk}^m \qquad \forall b \in B \ \forall 1 \leq k \leq |T_b| \tag{4}$$

(2) We impose that at most one transformation can hold each position:

$$\sum_{t_{bk} \in S(m)} x_{bk}^m \leq 1 \quad \forall m \tag{5}$$

Notice that the sum is only over the transformations that belong to the same strongly connected component. The constraints in equation 5 enforce that the solution is a partial sequence, and hence no more than one transformation can be assigned to the same position of the sequence.

(3) We impose the cardinality semantics of a combinatorial bid b: selecting at least one transformation within a bid implies selecting all the transformations within the same bid with the corresponding multiplicity. Formally:

$$x_{bk} = x_b \cdot |\mathcal{D}_b|_{t_{bk}} \qquad \forall b \in B \ \forall 1 \leq k \leq |T_b| \tag{6}$$

where $|D_b|_{t_{bk}}$ is the multiplicity of transformation t_{bk} in $\mathcal{D}_b$.

(4) The atomic bids submitted by each bidder are mutually exclusive (XOR). Thus, we impose the XOR semantics of each bid as follows:

$$\sum_{b \in B_l} x_b \leq 1 \qquad \forall l \in L \tag{7}$$

(5) We enforce that enough goods are available to use the corresponding transformations at each position of the solution sequence. This constraint, represented by equation 8 below, must be added only if the transformations assigned to position m belong to a simple cycle. We say that a position m belongs to L_F iff the solution template S maps m to transformations belonging to a simple cycle. Now, we can impose:

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{bk} \in S(l)} x_{bk}^{l} \cdot [\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g)] \geq \sum_{t_{bk} \in S(m)} x_{bk}^{m} \cdot \mathcal{I}_{bk}(g) \qquad \forall g, \forall m \in L_F \tag{8}$$

(6) After having performed all the selected transformations, we enforce that the goods held by the auctioneer must be a superset of the final goods, namely at least $\mathcal{U}_{out}$:

$$\mathcal{U}_0(g) + \sum_{m} \sum_{t_{bk} \in S(m)} x_{bk}^{m} \cdot [\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g)] \geq \mathcal{U}_{out}(g) \qquad \forall g \tag{9}$$

Hence, solving the MMUCA WDP amounts to optimising the following objective function:

$$\max \sum_{b \in B} x_b \cdot p_b \tag{10}$$

subject to inequations 4 to 9.

4 Bid Generator Requirements

In order to test and compare MMUCA WD algorithms, researchers must be provided with algorithms or test suites to generate artificial data that are representative of the auction scenarios a WD algorithm is likely to encounter. Hence, WD algorithms can be accurately tested, compared, and improved. Unfortunately, we cannot benefit from any previous results in the literature since they do not take into account the notion of transformation introduced in

[4,30]. In this section we make explicit the requirements for a bid generation technique considering that in MMUCA agents trade transformations instead of goods.

A naive approach to artificial bid generation would be to create bids at random. It is easy to see that this approach would generate unrealistic bids, from which we cannot get useful conclusions when solving MMUCAs on them. Let us consider a random bid $b = \{\{(\mathcal{I}, \mathcal{O})\}, p, l\}$. If goods appearing in sets $\mathcal{I}$ and $\mathcal{O}$ are selected randomly, there is little chance that they will represent a realistic transformation. Also, if p is chosen randomly, it will not be related with the actual values of the goods in the sets $\mathcal{I}$ and $\mathcal{O}$ and consequently the transformation would be either too profitable or too expensive for the auctioneer, unrealistically easing the problem.

If individual bids randomly generated may be unrealistic, sets of random bids also present similar drawbacks. Let us consider two bids b_1 and b_2 by different bidders. In a real MMUCA, there is a high chance that two bids involve the same goods (or the same type of goods; both are offering close transformations with small variations in price). However, if b_1 and b_2 are generated at random, there is little chance that they will contain similar goods or that their prices are related.

These two simple examples clearly show that random bid generation creates unrealistic scenarios. Testing WD algorithms on these scenarios is basically useless, because any extracted conclusion cannot be used in real settings. The bid generator has to satisfy a number of requirements to make the artificial bids close to the bids that are likely to appear in a real-world auction.

Since MMUCAs generalize CAs, as discussed in [4], our approach is to depart from artificial data sets generators for CAs, keeping the requirements summarized in [31], namely:

(1) there is a finite set of goods;
(2) certain goods are more likely to appear together than others;
(3) the number of goods in a bundle is often related to which goods compose the bundle;
(4) valuations are related to which goods appear in the bundle;
(5) valuations can be configured to be subadditive, additive or superadditive in the number of goods requested; and
(6) sets of XOR'ed bids are constructed on a per-bidder basis.

Notice though that the requirements above must be reformulated and extended in terms of transformations, since a bidder in a MMUCA bids over a bundle of transformations whereas a bidder in a CA bids over a bundle of goods. Hence, in what follows we discuss the CA requirements listed above reformulated for MMUCA:

1. There is a finite set of transformations. A CA generator bundles goods from a given set of goods to construct bids. Hence, the respective question reformulated for MMUCA is: What is the set of transformations from which a MMUCA generator constructs bids? Within a given market we expect several producers to offer the very same or similar services (transformations) at different prices, as well as several consumers to require the very same or similar services (transformations) valued at different prices. In other words, within a given market we can identify a collection of common services that companies request and offer. For instance, in the example in Figure 1 (a), several providers may offer to assemble a cylinder through the very same transformation: $t = (\{6 \cdot screws, 1 \cdot cylinder\_line, 1 \cdot cylinder\_ring, 1 \cdot cylinder\_head\}, \{cylinder\})$. Eventually, a provider may either offer to perform such transformation several times (e.g. as many times as cylinders are required), or to bundle it with other transformations, or both.

2. Certain transformations are more likely to appear together than others. In any market, services and goods are related to each other. For example, the production process for a good can also generate some other products that can be sold with it or used in another industrial process. Moreover, some services or products are usually bought together by the final customer.

3. The number of transformations in a bundle is often related to which transformations compose the bundle. Since bids are composed as combinations of market transformations, we must introduce the notion of *transformation multiplicity* as the counterpart of good multiplicity (the number of units of a given good within an offer or a request). Say that in a CA a bidder submits a bid for the goods in multi-set $\{engine, engine, piston\}$. It is clear that the multiplicity in this bundle of good *engine* is two, whereas the multiplicity of good *piston* is one. But things become more complicated when we consider transformations because the multiplicity of a given transformation must be defined in terms of another transformation, which in turn is composed of multiple input and output goods. Intuitively, we say that a transformation is a multiple of another one if both share the same input and output goods and the former has more input and output goods than the latter but keeping the same ratio between input and output goods. For instance, given transformations $t = (\{6 \cdot screws, 1 \cdot cylinder\_line, 1 \cdot cylinder\_ring, 1 \cdot cylinder\_head\}, \{cylinder\})$ and $t' = (\{12 \cdot screws, 2 \cdot cylinder\_line, 2 \cdot cylinder\_ring, 2 \cdot cylinder\_head\}, \{2 \cdot cylinder\})$ we way that t' has multiplicity two with respect to t.

4. Valuations are related to which transformations appear in the bundle; furthermore, transformation valuations keep consistency with respect to bidder valuations for goods involved in each transformation. A further issue has to do with the way bidders value transformations and bundles of transformations. Notice that performing a transformation to assemble the engine in Figure 1 (a) results in a new product that has more market value than its parts. Therefore, a car maker values the transformation according to his expected benefits, namely the difference between the expected market value of the engine and the cost of its parts. Hence, if the parts cost \$850 and the expected market value of the engine is \$1000, the car maker should be willing to offer to pay less than \$150 for the transformation. On the other hand, any provider is expected to request less than \$150 in order to perform the transformation. In general, buyers and providers in a MMUCA should value a transformation on the basis of the difference between the expected market value of its output goods and the cost of its input goods. Notice though that we are not assuming here that such a difference must be always positive. Likewise, the bidder should value bundles of transformations considering the prices of transformations included in it.

5. Appropriate valuations can be configured to be subadditive, additive or superadditive in the number of transformations requested. This requirement tries to capture the multiplicity-based (volume-based) discounts policies that are applied in the real world. Significant discounts are applied in real markets when goods and services are traded at certain numbers of units. For example, we observe that screws are usually traded in higher quantities than full engines. Thus, not surprisingly the same (percentage) discount may apply to an offer for 100 screws than to an offer for 5 engines. Hence, an offer to produce more than 5 engines, though more unlikely, should reflect higher discounts.

6. Sets of XOR'ed bids are constructed on per-bidder basis. When a bidder submits different bids in an XOR bid he declares they are mutually exclusive offers, expressing substitutability. For example, the following offer $\text{BID}_1(\{(\{engine\}\{\})\}, 100, l)$ XOR $\text{BID}_2(\{(\{2 \cdot engine\}\{\})\}, 190, l)$ stands for a bidder that offers to buy two engines or one engine but not in any case three engines. On the other hand when a bidder expresses complementarity he translates the OR bids as XOR bids. For example if a bidder wants to buy one engine or one cylinder he submits the following XOR-bid: $\text{BID}_1(\{(\{engine\}\{\})\}, 100, l)$ XOR $\text{BID}_2(\{(\{cylinder\}\{\})\}, 30, l)$ XOR $\text{BID}_3(\{(\{cylinder, engine\}\{\})\}, 140, l)$. In both cases bids submitted in the same XOR bid are likely to have similarities and, consequently, combining bids into XOR bids randomly does not capture this property.

7. Unrequested goods by the auctioneer may become involved in the auction. This requirement stems from the fact that, unlike auctioneers in CAs, not all goods involved in a MMUCA must be requested by the auctioneer. In our example, a car maker in need of engines as depicted in Figure 1 (a), it can run a MMUCA requesting only engines. Thereafter, bidders may offer already-assembled engines, or other goods (e.g. parts like crankcases, crankshafts, or screws) that jointly with transformations over such goods help produce the requested goods.

8. Goods and bidders are organized into levels. Unlike CAs, goods in MMUCAs are organised according to the structure of the supply chain they belong to. Thus, each good must be situated in a level of the supply chain. Structuring the supply chain into levels also affects bidders, who are usually expected to buy goods from one level and sell goods in the next one. In this sense, we can also talk of assigning levels to bidders.

5 An Algorithm for Artificial Data Set Generation

We describe a bid generation algorithm that automates the generation of artificial data sets for MMUCA while capturing the above requirements.

The algorithm's purpose is to generate MMUCA WDPs. We recall that an instance of this problem has three main components:

- a multiset of goods in stock (namely $\mathcal{U}_{in}$);
- a multiset of goods required (namely $\mathcal{U}_{out}$); and
- a set of bids.

Figure 4 shows the architecture of the generator. First, the *good generation* process creates the set of goods over which the auction will be run. Second, the *market transformation generation* process defines the transformations which are commonly traded in that market. Third, the *requested and stocked good generation* process determines the goods (and quantities) previously available ($\mathcal{U}_{in}$) and the goods (and quantities) requested by the auctioneer ($\mathcal{U}_{out}$). Fourth, the *bid generation* process creates a set of bids by customizing and composing market transformations. Finally, the *pricing* process assesses the price for each atomic bid.

In the following, we detail each of the processes mentioned above.

5.1 Good Generation

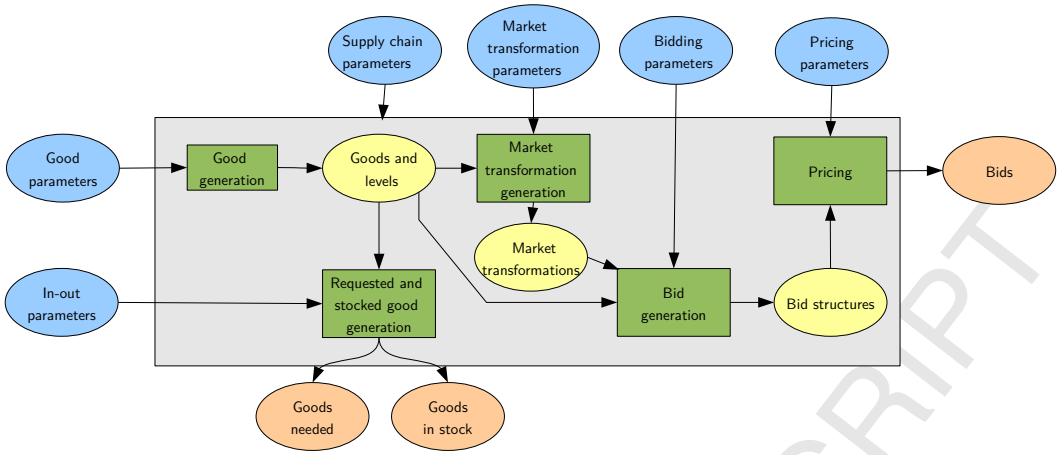


Fig. 4. Architecture of the MMUCA WDP generator

Algorithm 1 GenGoods(n_g, $\mathcal{M}_g$,ℓ_g)

1: $G \leftarrow \{\}$;
2: **for** $i = 1$ to n_g **do**
3: $\quad g \leftarrow newGood()$;
4: $\quad g.level \leftarrow Sample(\ell_g)$;
5: $\quad g.m \leftarrow Sample(\mathcal{M}_g)$;
6: $\quad G \leftarrow G \cup \{g\}$;
7: **end for**
8: return G

For each of the n_g goods at auction, this process (detailed in algorithm 1) defines two characteristics: its level and its multiplicity distribution. Following requirement 8, goods are structured into different levels. Each good is assigned to one of these levels by sampling a probability distribution ℓ_g, which is a parameter of the process (line 4). Following requirement 3 in [31], we model the fact that different goods are usually traded in different quantities (i.e. at the customer level, cars are usually traded in single units whereas wheels in 4 units). However, we do not want to impose a fixed number of units (packaging) at which each good is traded, but represent the fact that it is common to ask for 1, 2, 3 or 4 wheels in a request. The good multiplicity distribution models this by assigning a probability to each of the possible quantities at which a good is traded. In our generator, for each good we obtain its multiplicity distribution by sampling the good multiplicity law ($\mathcal{M}_g$) that is received as parameter.

5.2 Market Transformations Generation

This process (detailed in algorithm 2), following requirement 1, generates a fi-

Algorithm 2 GenMarketTransformations($G, \mathcal{M}_t, d_{IO}, \ell_t, \mathscr{n}_{in}^{MT}, \mathscr{n}_{out}^{MT}$)

for $level = 1$ to $nLevels$ **do**
  $MT(level) \leftarrow \{\,\}$
end for
for $g \in G$ **do**
  $MT(g.level) \leftarrow MT(g.level) \cup \{GenOTransf(g, \mathcal{M}_t)\}$
  $MT(g.level + 1) \leftarrow MT(g.level + 1) \cup \{GenITransf(g, \mathcal{M}_t)\}$
end for
for $i = 1$ to $d_{IO} \cdot n_g$ **do**
  $level \leftarrow Sample(\ell_t)$
  $MT(level) \leftarrow MT(level) \cup \{GenIOTransf(level, \mathcal{M}_t, \mathscr{n}_{in}^{MT}, \mathscr{n}_{out}^{MT})\}$;
end for
return MT

Algorithm 3 GenITransf($g, \mathcal{M}_t$)

$quantity \leftarrow Sample(g.\mathscr{m})$;
$mt.in \leftarrow \{quantity$ units of $g\}$; $mt.out \leftarrow \{\,\}$;
$mt.\mathscr{m} \leftarrow Sample(\mathcal{M}_t)$;
return mt;

Algorithm 4 GenOTransf($g, \mathcal{M}_t$)

$quantity \leftarrow Sample(g.\mathscr{m})$;
$mt.in \leftarrow \{\,\}$; $mt.out \leftarrow \{quantity$ units of $g\}$;
$mt.\mathscr{m} \leftarrow Sample(\mathcal{M}_t)$;
return mt;

Algorithm 5 GenIOTransf($level, \mathcal{M}_t, \mathscr{n}_{in}^{MT}, \mathscr{n}_{out}^{MT}, p_b, p_f$)

$mt.in \leftarrow SelectGoods(level - 1, \mathscr{n}_{in}^{MT}, p_b, p_f))$;
$mt.out \leftarrow SelectGoods(level, \mathscr{n}_{out}^{MT}, p_b, p_f)$;
$mt.\mathscr{m} \leftarrow Sample(\mathcal{M}_t)$;
return mt;

nite set of transformations that are usually offered and requested in the market that we refer to as *market transformations*. Therefore, market transformations represent the *goods* providers and buyers can request and bid for. Hence, bids for MMUCAs shall be composed as combinations of market transformations. For each market transformation, the process determines: its level in the supply chain, its multiplicity distribution and which are its input and output goods and their quantities. As previously argued for goods, and following requirement 3 listed in section 4, each transformation is asigned a multiplicity distribution, sampled from the transformation multiplicity law ($\mathcal{M}_t$) that the algorithm receives as a parameter.

We assume that it is always possible in the market to sell and buy each good at auction by itself. To represent that, for each good we construct two market

Algorithm 6 SelectGoods($level, \mathscr{n}, p_b, p_f$)

1: $goodset \leftarrow \{\}$;
2: $nGoods \leftarrow Sample(\mathscr{n})$;
3: **for** $i = 1$ to $nGoods$ **do**
4: $\quad gLevel \leftarrow$SampleLevel($level - 1, p_b, p_f$));
5: $\quad g \leftarrow SelectGood(gLevel)$;
6: $\quad quantity \leftarrow Sample(g.\mathscr{m})$;
7: $\quad goodset \leftarrow mt.in \cup \{quantity$ units of $g\}$;
8: **end for**
9: **return** $goodset$;

transformations: one (O-transformation) that offers a number of units of that good and one (I-transformation) that asks for them.

The remaining market transformations (the IO transformations), are randomly generated following algorithm 5. The number of IO market transformations is determined as a product of the number of goods in the market and the market transformation density parameter d_{IO} (algorithm 2, line 8). The number of goods in and out of a transformation are determined by sampling the probability distributions $\mathscr{n}_{in}^{MT}$and $\mathscr{n}_{out}^{MT}$. In order to maintain the supply chain levels, each market transformation is assigned a level by sampling the probability distribution for transformation levels ℓ_t (algorithm 2, line 9). We select the input and output goods of a market transformation by taking into account its level (see algorithm 6). We assume there is a natural flow for transformations in the supply chain. Usually, a market transformation at level k has input goods at level $k-1$ and outputs goods at level k. However, in most real scenarios, the levels of the supply chain are not strictly defined. Hence, we allow our market transformations to occasionally break these restrictions. In order to do that we have introduced a Markov chain model (an example is depicted in Figure 5), controlled by two parameters p_f and p_b. The parameter p_b is the probability of changing the level against the natural flow of the supply chain. The parameter p_f is the probability of changing the level following the natural flow of the supply chain. For each good to be added to a market transformation, first we determine the level of the good using the Markov chain and then we randomly select the good from that level.

5.3 Requested and Stocked Goods Generation

This process (detailed in algorithm 7) assesses the number of units of each good that the auctioneer requests ($\mathcal{U}_{out}$) and the number of units of each good that the auctioneer has available when the auction starts ($\mathcal{U}_{in}$). The number of goods to be included in $\mathcal{U}_{out}$ (resp. $\mathcal{U}_{in}$) is determined by sampling the probability distribution $\mathscr{n}_{out}$(resp. $\mathscr{n}_{in}$). Once the number of goods is determined,

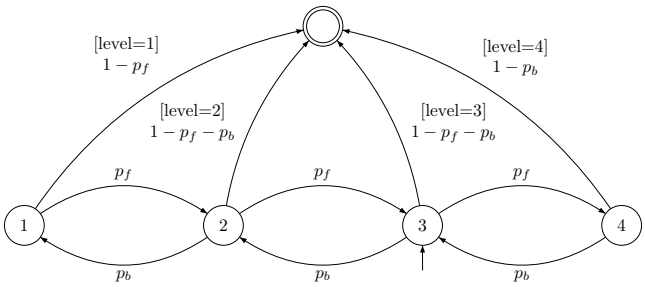


Fig. 5. Example of Markov chain for level selection in a 4 level supply chain for a bidder selling a good at level 3.

Algorithm 7 RequestedAndStocked($a, \aleph_{in}, \aleph_{out}, p_b, p_f$)

1: $\mathcal{U}_{in} \leftarrow SelectGoods(a-1, \aleph_{in}, p_b, p_f))$;
2: $\mathcal{U}_{out} \leftarrow SelectGoods(a, \aleph_{out}, p_b, p_f)$;
3: **return** $\mathcal{U}_{in}, \mathcal{U}_{out}$;

the level of the auctioneer in the supply chain (a) is used to determine the goods using the Markov chain model explained in the previous section.

Notice that by selecting a subset of the goods we fulfil requirement 7 listed in section 4, namely *unrequested goods by the auctioneer may be involved in the auction.*

5.4 Bid Generation

The bid generation algorithm (algorithm 8) generates the bids involved in the auction. Since the XOR language has been proven to be fully expressive [4] each bidder is assumed to submit a single XOR bid.

Algorithm 8 BidGeneration($n_t, \ell_b, \aleph_{and}, \aleph_{xor}$)

1: $Bids \leftarrow \{\}$
2: **while** NumTransformations($Bids$) $< n_t$ **do**
3: $XORBid \leftarrow \{\}$
4: $nXORClauses \leftarrow Sample(\aleph_{xor})$
5: $level \leftarrow Sample(\ell_b)$
6: **for** $x = 1$ to $nXORClauses$ **do**
7: $XORBid \leftarrow XORBid \cup GenerateAtomicBid(level, \aleph_{and})$;
8: **end for**
9: $Bids \leftarrow Bids \cup \{XORBid\}$
10: **end while**
11: **return** Bids;

The algorithm adds new bidders until a certain number of transformations given by parameter n_t is reached (line 2). For each bidder, the algorithm determines its level by sampling ℓ_b (line 5). After that, it computes the atomic bids composing it. We sample the number of atomic bids from the probability distribution $\mathscr{n}_{xor}$(line 4). Each atomic bid is composed of a set of transformations that we compute by sampling the probability distribution $\mathscr{n}_{and}$(algorithm 9). The concrete transformations are randomly obtained out of the market transformations available at the level of the supply chain where the bidder is located. The quantity of each transformation is computed by sampling the transformation multiplicity distribution. Notice that by assessing the number of units to include in a bundle using a probabilistic distribution that depends on each transformation we fulfil requirement 3: *the number of transformations in a bundle is often related to which transformations compose the bundle*. Also note that by selecting the market transformations from the level of the bidder we fulfil requirement 2 in section 4 stating *certain transformations (for which some complementarities exist) will be more likely to appear together than others*. Moreover, since commonly XOR bids would be composed of atomic bids created from transformations from the same level, bids submitted in a XOR form are more likely to have similarities fulfilling, in that way, requirement 6 namely *sets of XOR'ed bids are constructed on per-bidder basis*.

5.5 Pricing bids

The last stage of the algorithm (detailed in algorithm 10) determines a price for each of the atomic bids. To fulfil the requirements concerning valuations listed in section 4, a pricing policy must provide the means to price a good, a transformation, multiple units of the very same transformation, and a bundle of transformations in a realistic manner. As to pricing goods, in order to vary prices among bidders, our algorithm receives as a parameter a probability distribution $\not{p}$ over matrixes. Once the number of goods (n_g) and of bidders (n_b) is known, the algorithm samples $\not{p}$ to get a matrix $n_b \times n_g$ where the (i, j) entry is the price bidder i assigns to good j.

Algorithm 9 GenerateAtomicBid(level,$\mathscr{n}_{and}$)

1: $Transformations \leftarrow \{\}$;
2: $nTransformations \leftarrow Sample(\mathscr{n}_{and})$;
3: **for** $b = 1$ to $nTransformations$ **do**
4: $\quad t \leftarrow SampleMarketTransformation(level)$
5: $\quad quantity \leftarrow Sample(t.\mathscr{m})$
6: $\quad Transformations \leftarrow Transformations \cup \{quantity \text{ copies of } t\}$
7: **end for**
8: **return** $\{Transformations\}$;

Algorithm 10 Pricing(Bids,$\not{s}$,δ_m,δ_{and})

1: $prices \leftarrow Sample(\not{s}, |Bids|, |G|)$;
2: **for** $b_{xor} \in Bids$ **do**
3: **for** $b_a \in AtomicBids(b_{xor})$ **do**
4: $b_a.price \leftarrow 0$;
5: **for** $t \in Transformations(b_a)$ **do**
6: $rawValue \leftarrow \sum_{g \in t.outputs} prices[b_{xor}, g] - \sum_{g \in t.inputs} prices[b_{xor}, g]$;
7: $b_a.price \leftarrow b_a.price + rawValue \cdot t.multiplicity \cdot \delta_m(t.multiplicity)$;
8: **end for**
9: $b_a.price \leftarrow b_a.price \cdot \delta_{and}(|Transformations(b_a)|)$;
10: **end for**
11: **end for**
12: **return** Bids;

Thereafter, a transformation's price for bidder b is assessed in terms of the difference from his valuation of its output goods with respect to his valuation of its input goods. Accordingly *transformation valuations keep consistency with respect to bidder valuations for goods involved in each transformation* as stated by requirement 5 in section 4.

Finally, each atomic bid valuation is obtained by adding the prices of its transformations. Hence *valuations are related to which transformations compose the bundle* as stated by requirement 4 although varying among different bidders.

Furthermore we propose to introduce superadditivity by applying multiplicity-based discounts to transformations addressing the requirement that *valuations can be configured to be subadditive, additive o superadditive in the number of transformations requested*. In order to do that, a discount δ_m is applied to the valuation of a tranformation depending on the number of times the transformation appears in the atomic bid (line 7). Also, a discount δ_{and} is applied to the atomic bid valuation depending on the number of different transformations in the atomic bid (line 9).

6 Experimental results

In this section we use the MMUCA problem generator to analyse the performance of the IP solver explained in section 3.3.2. We first provide details about the experimental design and then summarize the empirical results.

Group	Parameter	Description
Supply chain parameters	l	Number of levels
	a	Auctioneer level
	p_b	Probability of changing level against the natural order
	p_f	Probability of changing level favouring the natural order
Good parameters	n_g	Number of goods
	$\mathcal{M}_g$	Good multiplicity law
	ℓ_g	Distribution over levels for goods
Market transformation parameters	$\mathcal{M}_t$	Transformation Multiplicity law
	d_{IO}	Density of IO transformations
	ℓ_{MT}	Distribution over levels for market transformations
	$\mathscr{n}_{in}^{MT}$	Distribution of the number of input goods in market transformations
	$\mathscr{n}_{out}^{MT}$	Distribution of the number of output goods in market transformations
In-out parameters	$\mathscr{n}_{in}$	Distribution over the number of goods in $\mathcal{U}_{in}$
	$\mathscr{n}_{out}$	Distribution over the number of goods in $\mathcal{U}_{out}$
Bidding parameters	n_t	Number of transformations
	$\mathscr{n}_{and}$	AND arity distribution
	$\mathscr{n}_{xor}$	XOR arity distribution
	ℓ_b	Distribution of bids per level
Pricing parameters	$\not{p}$	Distribution over price profiles
	δ_m	Transformation multiplicity discount function
	δ_{and}	Bid discount function

Table 2
Artificial data set generator parameters.

6.1 Experimental design

We have performed two experiments. In the first one, we compared a set of factors that affect the algorithm time performance, to get an idea of the relative sensitivity among each of them. In the second one, we try to ascertain the relationship between the syntactical complexity of the bidders expressions and the time complexity of the solver.

In order to apply the MMUCA problem generator, we need to specify the parameters summarized in table 2. In both experiments we fixed some of the parameters to make the study computationally feasible. Notice that our experiments and conclusions do not pretend being exhaustive but just providing some first ideas on the plausible scenarios where MMUCA could be applied and showing the flexibility of the generator to analyse and simulate different scenarios. We present now the parameters that have been kept fixed in both experiments.

6.1.1 Supply chain parameters

In our experiments the number of supply chain levels l has been fixed to 5. The auctioneer is fixed to be in the middle of the supply chain, so its level (a) is fixed to 3. Whenever p_b and p_f are modified, we keep fixed the probability of changing level (either forward or backward) to 30%, that is, $p_f + p_b = 0.3$.

6.1.2 Good parameters

A sampling of the good multiplicity law ($\mathcal{M}_g$) returns the probability distribution for the multiplicity of a good. In our experiments, when we sample $\mathcal{M}_g$ to determine the multiplicity distribution for a good, we sample a parameter m_g from a uniform distribution in [0.01,1]. The multiplicity distribution is a geometric distribution with success probability m_g. The probability distribution that distributes goods among levels (ℓ_g) is fixed to a uniform distribution.

6.1.3 Market transformation parameters

The transformation multiplicity law ($\mathcal{M}_t$) is built as follows. First, parameter m_t is sampled from a uniform distribution in [0.8,1]; then, m_t is used as the parameter of a geometric distribution. Note that making this choice means that we are assuming that multiplicities for transformations are expected to be smaller than multiplicities for goods. The number of IO market transformations generated per good (d_{IO}) is fixed to 2. This means that the number of IO transformations equal the sum of input (I) and output (O) transformations. The probability distribution that determines the level from which to draw a market transformation (ℓ_t) is determined by assuming that each level we move away from the auctioneer level, we decrease the probability of receiving a bid from that level by 50%. This means that bids are mostly concentrated closer to the auctioneer level. The probability distributions for the number of goods in and out of a transformation ($\nu_{in}^{MT}$and ν_{out}^{MT}) are both fixed to a geometric distribution with parameter 0.7.

6.1.4 Other parameters

- In-out parameters. The distribution over the number of goods in stock (ν_{in}) is a geometric distribution with success probability 0.4. The distribution over the number of goods requested by the auctioneer (ν_{out}) is a geometric distribution with success probability 0.3. Both distributions are bounded so that the number of goods in stock or requested can never be larger than half the total number of goods.
- Bidding parameters. We fix the probability distribution that determines the level from which to draw a bid (ℓ_b) in the same way.

Parameter	n_g	p_b	n_t	$\varkappa_{and}$	$\varkappa_{xor}$
Values	$\{20, 50\}$	$\{0, 0.1\}$	$\{50, 100, 150, 200, 250\}$	$\{1, 2\}$	$\{1, 2\}$

Table 3
Generator parameter settings for the factor analysis

- Pricing parameters. The discount due to the multiplicity of the transformations is $\delta_m(n) = 0.1 * (1 - e^{-0.05n})$ and the discount due to the bid length is $\delta_{and}(n) = 0.1 * (1 - e^{-0.5n})$ In both cases the maximum discount is 10%. he transformation multiplicity discount grows more slowly than the bid discount.

6.1.5 Computational details

Considering the above-described experimental scenarios, we have run our experiments as follows. Firstly, we have generated 50 solvable WDP instances for each parameter setting using a MATLAB implementation of the artificial data set generator detailed in section 5 whose source code is publicly available at `http://zeus.maia.ub.es/~cerquide/mmucaGenerator` . We have solved each WDP with an IP implementation of MMUCA in CPLEX 10.1 and recorded the resulting solving times. Notice though that we have set to 4800 seconds the time deadline to solve each instance. We have only considered feasible WDP instances to calculate solving times since the time required by CPLEX to prove infeasibility is (usually) significantly lower than time required to find an optimal solution. Our tests have been run on a Dell Precision 490 with double processor Dual-Core Xeon 5060 running at 3.2 GHz with 2GB RAM on Linux 2.6.

6.2 Factor analysis

In our first experiment we compare among multiple parameters to get a first idea of how each of them influences the complexity. Table 3 shows the range of parameter values explored. For $\varkappa_{and}$ and $\varkappa_{xor}$, in order to simplify our analysis, we fixed the probability distribution to a single value (no randomness was involved in $\varkappa_{and}$ and $\varkappa_{xor}$ in our experiment).

Figure 6 compares the growth in solving time as the number of transformations increases for different values of p_b and n_g. Each line shows the medians of the computing times for one combination of values p_b and n_g as the number of transformations increases. The first conclusion arising from this figure is that the main driver for complexity is p_b, that is, the probability of breaking the natural flow of the supply chain. This is reasonable because the solver efficiency depends on the size of the strongly connected components of the

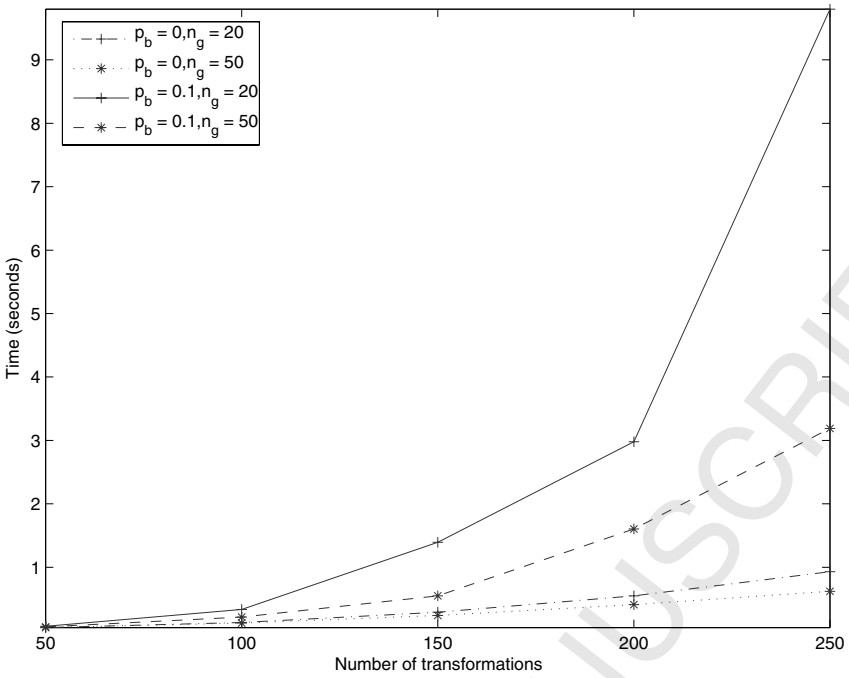


Fig. 6. Factors affecting the time complexity of MMUCA solver

problem, and if the natural flow is frequently broken, the size of the strongly connected components is expected to increase.

As expected, figure 6 also shows that the algorithm complexity increases as n_t (the number of transformations) increases. However, the ratio of increase has a strong dependency of the value of p_b. For $p_b = 0$, the algorithm scales nicely as the number of transformations increases.

The number of goods n_g is the third factor affecting the computational performance of the solver. We observe that, curiously, the setting with a largest number of goods has a smaller time complexity. This is explained by the way we generate bids. Thus, given a number of transformations and a value for p_b, the larger the number of goods the less likely to have large strongly connected components.

Regarding the remaining parameters, there was no clear pattern arising from the analysis of $\mathscr{m}_{and}$ and $\mathscr{m}_{xor}$ influence. With the objective to clarify this relationship, we run the following experiment.

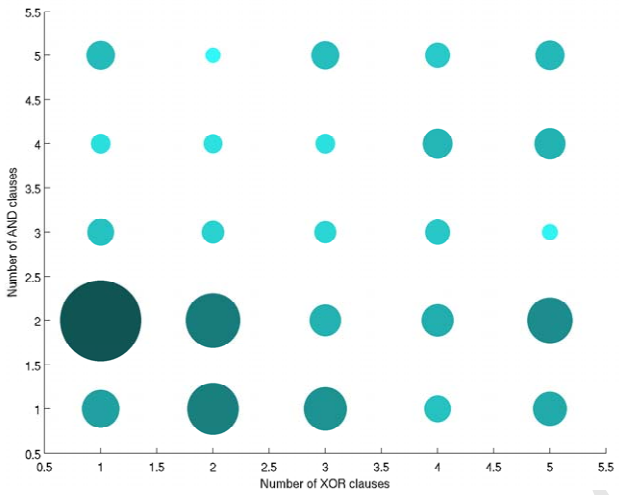


Fig. 7. Influence of the syntactical complexity of the bids in the time complexity of the MMUCA solver. The area of the circles represents the mean solving time.

6.3 Bidding language analysis

In this experiment we analysed specifically the influence of the syntactical complexity of the bids in the solver perfomance. We varied $\mathscr{n}_{xor}$ and $\mathscr{n}_{and}$ as described in table 4.

Parameter	n_g	p_b	n_t	$\mathscr{n}_{and}$	$\mathscr{n}_{xor}$
Values	20	0.05	200	$\{1,2,3,4,5\}$	$\{1,2,3,4,5\}$

Table 4
Generator parameter settings for the bidding language analysis

Figure 7 shows the median solving time as the area of a circle, for each combination of values of $\mathscr{n}_{xor}$ and $\mathscr{n}_{and}$. This figure confirms that there is no clear dependency pattern between the syntactical complexity of the bids and the time complexity of the solver. Furthermore, we would like to remark that with the settings provided, almost all the problems finished before reaching the time limit. We had 10 problems over the time limit, all of them for $\mathscr{n}_{and}$=1 (2 of them when $\mathscr{n}_{xor}$= 2, 5 when $\mathscr{n}_{xor}$=3, 1 when $\mathscr{n}_{xor}$=4, and 2 when $\mathscr{n}_{xor}$=5). This points out that even if there are no patterns arising from the analysis of a concentration measure such as the median, maybe there is a pattern regarding how the hardness of the problems spreads. Further experimental work needs to be performed to confirm this and to understand better the relationship between the bid complexity and the solving time complexity.

7 Conclusions and future work

In this work, we have attempted at making headway in the practical application of MMUCAs along two directions. Firstly, we have provided an algorithm to generate artificial data sets for MMUCA that tries to mimic supply chain scenarios. Our algorithm reformulates and extends in terms of transformations the requirements for an artificial data set generator for CAs. Secondly, we provide empirical tests for MMUCAs by assessing the performance of a CPLEX IP implementation. These tests indicate that the scalability of an IP implementation of MMUCA is affected by the size of the largest connected components in the transformation dependency graph. We have identified that when there is a strong natural flow in the supply chain the solver scales reasonably both with respect to number of transformations and to the number of goods. No clear relationship has been identified between the syntactical complexity of bids and the computational complexity of the solver.

Acknowledgements

This work has been partially funded by the Spanish projects TIN2006-15662-C02-01 and "Agreement Technologies" (CONSOLIDER CSD2007-0022, INGENIO 2010). The work of Meritxell Vinyals is supported by the Ministry of Education of Spain (FPU grant AP2006-04636).

References

[1] P. Cramton, Y. Shoham, R. Steinberg (Eds.), Combinatorial Auctions, MIT Press, 2006.

[2] M. H. Rothkopf, A. Pekec, R. M. Harstad, Computationally manageable combinational auctions, Management Science 44 (8) (1998) 1131–1147.

[3] N. Nisan, Bidding languages for combinatorial auctions, in: P. Cramton, et al. (Eds.), Combinatorial Auctions, MIT Press, 2006.

[4] J. Cerquides, U. Endriss, A. Giovannucci, J. A. Rodriguez-Aguilar, Bidding languages and winner determination for mixed multi-unit combinatorial auctions, in: Proc. of the 20th Int. Joint Conferences on Artif. Intelligence (IJCAI), Hyderabad, India, 2007, pp. 1221–1226.

[5] T. Sandholm, Algorithm for optimal winner determination in combinatorial auctions, Artificial Intelligence 135 (1-2) (2002) 1–54.

[6] A. Giovannucci, M. Vinyals, J. Cerquides, J. A. Rodriguez-Aguilar, Computationally-efficient winner determination for mixed multi-unit combinatorial auctions, in: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, 2008, to appear.

[7] T. Sandholm, S. Suri, A. Gilpin, D. Levine, Winner determination in combinatorial auction generalizations, in: AAMAS '02: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, Bologna, Italy, 2002, pp. 69–76.

[8] C. Caplice, Y. Sheffi, Combinatorial Auctions for Truckload Transportation, MIT Press, 2006, Ch. 21. Combinatorial Auctions.

[9] E. Cantillon, M. Pesendorfer, Auctioning Bus Routes: The London Experience, MIT Press, 2006, Ch. 22. Combinatorial Auctions.

[10] G. H. Martin Bichler, Andrew Davenport, J. Kalagnanam, Industrial Procurement Auctions, MIT Press, 2006, Ch. 23. Combinatorial Auctions.

[11] G. L. D. Michael O. Ball, K. Hoffman, Auctions for the Safe, Efficient, and Equitable Allocation of Airspace System Resources, MIT Press, 2006, Ch. 20. Combinatorial Auctions.

[12] A. Pekec, M. H. Rothkopf, Combinatorial auction design, Manage. Sci. 49 (11) (2003) 1485–1503.

[13] W. E. Walsh, M. P. Wellman, F. Ygge, Combinatorial auctions for supply chain formation, in: Proc. of the 2nd ACM Conference on Electronic Commerce, Minneapolis, Minnesota, 2000, pp. 260–269.

[14] V. Krishna, Auction Theory, Academic Press, 2002.

[15] P. Milgrom, Putting Auction Theory to Work, Cambridge University Press, 2004.

[16] L. M. Ausubel, P. Milgrom, The Lovely but Lonely Vickrey Auction, MIT Press, 2006, Ch. 1. Combinatorial Auctions.

[17] D. C. Parkes, Iterative Combinatorial Auctions, MIT Press, 2006, Ch. 2. Combinatorial Auctions.

[18] L. M. Ausubel, P. Milgrom, Ascending Proxy Auctions, MIT Press, 2006, Ch. 3. Combinatorial Auctions.

[19] P. Cramton, Simultaneous Ascending Auctions, MIT Press, 2006, Ch. 4. Combinatorial Auctions.

[20] P. C. Lawrence M. Ausubel, P. Milgrom, The Clock-Proxy Auction: A Practical Combinatorial Auction Design, MIT Press, 2006, Ch. 5. Combinatorial Auctions.

[21] S. P. Ailsa Land, R. Steinberg, PAUSE: A Computationally Tractable Combinatorial Auction, MIT Press, 2006, Ch. 6. Combinatorial Auctions.

[22] T. Sandholm, S. Suri, A. Gilpin, D. Levine, Winner determination in combinatorial auction generalizations, in: Proc. 1st Intl. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), ACM Press, 2002.

[23] A. Andersson, M. Tenhunen, F. Ygge, Integer programming for combinatorial auction winner determination, in: Fourth International Conference on Multiagent Systems (ICMAS 2000), Boston, MA, 2000, pp. 39–46.

[24] Y. Fujishima, K. Leyton-Brown, Y. Shoham, Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches., in: Proceeding of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), 1999, pp. 548–553.

[25] K. Leyton-Brown, Y. Shoham, M. Tennenholtz, An algorithm for multi-unit combinatorial auctions, in: Proceedings of the American Association for Artificial Intelligence Conference (AAAI), 2000, pp. 56–61.

[26] D. Lehmann, R. Mueller, T. M. Sandholm, The winner determination problem, MIT Press, 2006, Ch. 12. Combinatorial Auctions.

[27] R. Muller, Tractable Cases of the Winner Determination Problem, MIT Press, 2006, Ch. 13. Combinatorial Auctions.

[28] T. Sandholm, Optimal Winner Determination Algorithms, MIT Press, 2006, Ch. 14. Combinatorial Auctions.

[29] A. Giovannucci, J. Cerquides, J. A. Rodriguez-Aguilar, Proving the correctness of the CCIP solver for MMUCA, Tech. rep., IIIA-CSIC (2007).

[30] A. Giovannucci, J. A. Rodríguez-Aguilar, J. Cerquides, U. Endriss, On the winner determination problem in mixed multi-unit combinatorial auctions, in: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, Honolulu, Hawaii, USA, 2007, ACM Press.

[31] K. Leyton-Brown, Y. Shoham, Combinatorial Auctions, MIT Press, 2006, Ch. 18. A Test Suite for Combinatorial Auctions.