Computationally-efficient Winner Determination for Mixed Multi-Unit Combinatorial Auctions

Andrea Giovannucci,
Meritxell Vinyals,
J. A. Rodriguez-Aguilar
IIIA-CSIC. Bellaterra. Spain.
{andrea,meritxell,jar}@iiia.csic.es

Jesus Cerquides
Universitat de Barcelona, Barcelona, Spain.
cerquide@maia.ub.es

ABSTRACT

Mixed Multi-Unit Combinatorial Auctions offer a high potential to be employed for the automated assembly of supply chains of agents offering goods and services. Their winner determination problem is an NP-hard problem that can be mapped into an integer program. Nonetheless, the computational cost of the current solution hinders the application of mixed multi-unit combinatorial auctions to realistic scenarios. In this paper we propose a new integer program for mixed multi-unit combinatorial auctions that severely simplifies the problem by taking advantage of the topological characteristics of the winner determination problem. Furthermore, we provide empirical evidence showing that the new IP allows to cope with much larger supply chain formation scenarios.

1. INTRODUCTION

According to [8], "Supply Chain Formation (SCF) is the process of determining the participants in a supply chain, who will exchange what with whom, and the terms of the exchanges". Combinatorial Auctions (CAs) [3] are a negotiation mechanism well suited to deal with complementarities among the goods at trade. Since production technologies often have to deal with strong complementarities, SCF automation appears as a very promising application area for CAs. However, whilst in CAs the complementarities can be simply represented as relationships among goods, in SCF the complementarities involve not only goods, but also *transformations* (production relationships) along several levels of the supply chain.

The first attempt to deal with the SCF problem by means of Combinatorial Auctions (CA) was done by Walsh et al. in [8]. In order to automate SCF, they introduce the notion of task dependency network (TDN) as a way of capturing complementarities among production processes. Although very significant, this work does not allow bidders to express their preferences over bundles of production processes; it does not define a bidding language; and the structure of the supply chain has to fulfil strict criteria (e.g. acyclicity, processes can only produce one output good, etc). In order to overcome these drawbacks, Cerquides et al. introduce in [1] the so-called mixed multi-unit combinatorial auctions (MMUCAs), a generalisation of the standard model of CAs. Rather than negotiating over goods, in MMUCAs the auctioneer and the bidders can negotiate over *transformations*, each one characterized by a set of input goods and a set of output goods. A bidder offering a transformation is willing to produce its output goods after having received its input goods along with the payment specified in the bid. While in standard combinatorial auctions, a solution to the winner determination problem (WDP) is a set of atomic bids to accept, in MMUCAs, the *order* in which the auctioneer "uses" the accepted transformations matters. Thus, a *solution* to the WDP is a *sequence of transformations*. For instance, if bidder *Joe* offers to make dough if provided with butter and eggs, and bidder *Lou* offers to bake a cake if provided with enough dough, the auctioneer can accept both bids whenever he uses Joe's transformation before Lou's to obtain cakes.

Along the lines of [3], the WDP for MMUCAs can be solved by means of an Integer Program (IP), as shown in [1]. While this provides a first algorithmic solution to the WDP, in the IP proposed in [1] the number of variables grows quadratically with the overall number of transformations mentioned in the bids. Hence, such an IP hinders the application of MMUCAs.

Recent contributions on computationally efficient WDP solvers for different auction types (namely, [3] for CAs, [6] for MMUCAs, and [4] for multi-attribute double auctions) agree on and defend that a careful, formal analysis of the structure of WDPs can provide guidance for developing efficient winner determination solvers. Along the lines of these works, in this paper we propose an IP for MMUCAs that dramatically improves the computational efficiency of the IP reported in [1]. The search space reduction is achieved by enforcing MMUCA solutions to fulfil a template. The template reduces the possible orderings among transformations without losing solutions.

At this aim, we found our analysis on observing the structure of the WDP that results after establishing *dependence relationships* among transformations. For instance, in the example above, Lou's transformation clearly depends on Joe's: no dough, no cake! The analysis of the WDP based on dependency relationships helps design an IP that *a priori* establishes *when* to *use* each transformation.

Notice that the improvements that we propose in this paper are investigated both at the theoretical level (by showing a drastic reduction in the search space due to a different problem representation) and at the empirical level (by showing a very significant reduction of the solving time).

Cite as: Computationally-efficient Winner Determination for Mixed Multi-Unit Combinatorial Auctions, A. Giovannucci et. al, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16.,2008,Estoril,Portugal,pp.1071-1078.

Therefore, we argue that our main contribution is to make headway in the applicability of MMUCAs to SCF.

The paper is organised as follows. Section 2 summarises the work in [1] to provide an introduction to the WDP for MMUCAs along with a description of an IP solver. Section 3 introduces an improved version of the MMUCA WDP IP solver in [1], the Connected Component Integer Program (CCIP). Section 4 discusses the empirical results comparing the solvers in sections 2.1 and 3. Finally, section 5 draws some conclusions and outlines paths to future research.

2. MMUCA

In this section we firstly recall the notions of transformation and valuation over transformations, and the notion of a bidding language to transmit an agent's valuation in a MMUCA. Secondly, in subsection 2.1 we recall the definition and solution of the WDP for MMUCA.

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 $\mathcal{I}$. Bidders can offer any number of such transformations, including several copies of the same transformation. That is, agents can negotiate over *bundles of transformations* modelled as multisets $\mathcal{D} \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$. For example, $\{(\emptyset, \{eggs\}), (\{dough\}, \{cake\})\}$ means that the agent in question is able to deliver eggs (no input required) and that it is able to deliver a cake if provided with dough.

An *atomic bid* $Bid_b = (\mathcal{D}_b, p_b) = (\{(\mathcal{I}^1, \mathcal{O}^1), \ldots, (\mathcal{I}^n, \mathcal{O}^n)\}, p_b)$ specifies a finite multiset of finite transformations $\mathcal{D}_b$, a price p_b. Intuitively, Bid_b means that the agent is willing to make a payment of p_b in return for being allocated all the transformations in $\mathcal{D}_b$ (in case p_b is a negative number, this means that the agent will accept the deal if it *receives* an amount of $|p_b|$)[1]. For instance, $(\{(\{butter, eggs\}, \{dough\})\}, -20)$ means that the agent can produce dough for \$20 if given butter and eggs.

A suitable *bidding language* should allow a bidder to encode choices between alternative bids and the like [3]. Regarding an IP solution to the WDP, the work in [1] shows that the XOR-language is fully expressive for MMUCA, and then restricts the bids received to be expressed in this language. Relying on their results, we will assume the same bidding language. However, the results can be easily extended to other bidding languages, in particular languages including an OR operator.

2.1 A General IP for the WDP

The *input* to the 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 combinatorial auctions, a solution to the WDP is a set of atomic bids to accept. As to MMUCAs, 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 $Bid_1 = (\{a\}, \{b\}, 10)$ and $Bid_2 = (\{b\}, \{c\}, 20)$ is feasible involves realising that he has to use Bid_1 before Bid_2. Thus, a *solution* to the WDP will be a *sequence of transformations*. A *valid* solution has to meet two conditions:

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.

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.

Let B be the set of all atomic bids. An atomic bid $b = (\mathcal{D}_b, p_b)$ consists of a multiset of transformations and a price. $\mathcal{D}$ is the multi-set of all the submitted transformations. Then, the maximum length of the solution sequence is $\ell = |\mathcal{D}|$. L is the set of bidders. B_l is the set of all bids submitted by bidder $l \in L$. For each bid b, let T_b be the set of different transformations in $\mathcal{D}_b$ and let t_{bk} be a unique label for each transformation in T_b (for some arbitrary but fixed ordering of different transformations in T_b). Let $(\mathcal{I}_{bk}, \mathcal{O}_{bk})$ be the actual transformation labelled by t_{bk}. Finally, let $T = \bigcup_b T_b$ be the set of all different t_{bk}.

The auctioneer has to decide which transformations to accept and in which order to implement them. Thus, we define a decision variable $x^m_{bk} \in \{0, 1\}$, where b ranges in $\{1, \ldots, |B|\}$; for each b, k ranges in $\{1, \ldots, |T_b|\}$; and m ranges in $\{1, \ldots, \ell\}$. x^m_{bk} takes on value 1 if the transformation t_{bk} is selected at the mth position of the solution sequence, and 0 otherwise. We also introduce the following auxiliary decision variables: x_b is a binary variable that takes value one if bid b is accepted and x_{bk} is an integer variable that represents the number of times that transformation t_{bk} appears 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:

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

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

[1]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*, a bidder would always be prepared to accept more goods and give fewer goods away, without requiring a change in payment. 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.

variables as:

$$\mathcal{I}^m(g) = \sum_b \sum_k x^m_{bk} \cdot \mathcal{I}_{bk}(g) \qquad \forall g \in G \qquad (2)$$

$$\mathcal{O}^m(g) = \sum_b \sum_k x^m_{bk} \cdot \mathcal{O}_{bk}(g) \qquad \forall g \in G \qquad (3)$$

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

$$\mathcal{M}^m(g) = \mathcal{U}_{in}(g) + \sum_{i=1}^{m} \sum_b \sum_k x^i_{bk} \cdot (\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g))$$

Now, we are ready to express as linear equations all the constraints that a *valid solution sequence* must fulfil:

1. The variable x_{bk} contains the number of times that t_{bk} is selected in the solution sequence.

$$x_{bk} = \sum_m x^m_{bk} \quad (\forall b, k) \qquad (4)$$

2. At most one transformation is selected at each position of the solution sequence.

$$\sum_b \sum_k x^m_{bk} \leq 1 \quad (\forall m) \qquad (5)$$

3. Selecting at least one transformation within bid b implies selecting all the transformations within the same bid, each with its corresponding multiplicity.

$$x_{bk} = x_b \cdot |\mathcal{D}_b|_{t_{bk}} \quad (\forall b, k) \qquad (6)$$

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

4. The atomic bids submitted by each bidder are mutually exclusive (XOR).

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

5. We must ensure that all transformations have enough input goods available at each position of the transformation sequence. This maps to the condition:

$$\mathcal{M}^{m-1}(g) \geq \mathcal{I}^m(g) \qquad \forall m, \forall g$$

This can be expressed by means of our decision variables as:

$$\mathcal{U}_{in}(g) + \sum_{i=1}^{m-1} \sum_b \sum_k x^i_{bk} \cdot (\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g)) \geq \sum_b \sum_k x^m_{bk} \cdot \mathcal{I}_{bk}(g) \qquad (\forall g, m) \qquad (8)$$

6. After having performed all the selected transformations, the set of goods held by the auctioneer must be a superset of the final goods.

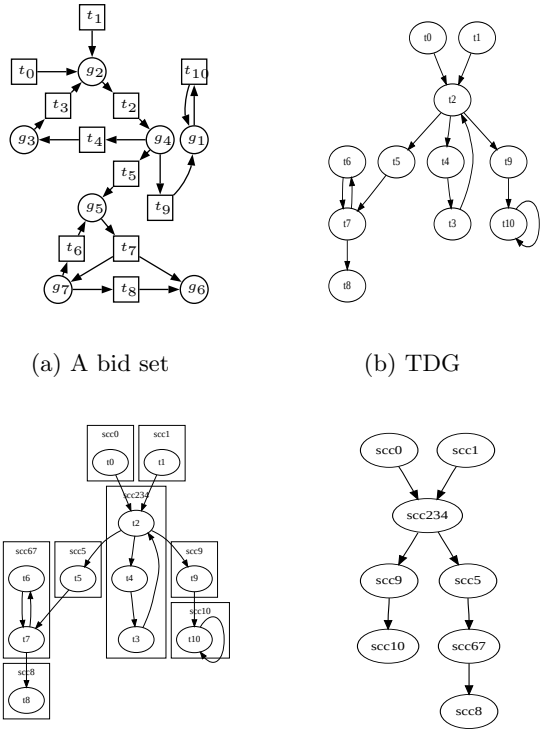
$$\mathcal{M}^{\ell}(g) \geq \mathcal{U}_{out}(g) \quad (\forall g \in G). \qquad (9)$$

In case of no free-disposal on the auctioneer's side simply substitute $\geq$ by $=$.

Therefore, solving the WDP for MMUCAs with XOR-bids amounts to maximising $\sum_{b \in B} x_b \cdot p_b$, while fulfilling constraints in equations (4)–(9).

Finally, an optimal solution sequence is obtained from the solution of the IP by making transformation t_{bk} the mth element of the solution sequence iff $x^m_{bk} = 1$. Henceforth, we shall refer to this solver as Direct IP solver (DIP for short).

The number of decision variables in the above integer program is of the order of $\ell \cdot |T|$ (corresponding to x^m_{bk}). This represents a serious computational cost. Thus, in what follows, we try to significantly reduce the number of variables (and thus the search space) required to solve the problem by analysing the topology of the WDP that results when putting together bids that yield dependency relationships among transformations.



(a) A bid set (b) TDG

(c) SCCs of the TDG (d) The strict order

Figure 1: An MMUCA bid set, the corresponding TDG, SCC, and Order Relation.

3. IMPROVING THE SOLVER: TRANSFORMATION DEPENDENCY ANALYSIS

In this section we introduce CCIP, a mapping of the MMUCA WDP into a new IP that substantially reduces the number of variables and constraints used by DIP. Next we outline the intuitions underlying the improvement we propose, whereas in the remaining subsections we develop a rigorous description of such intuitions.

The WDP for standard CAs can be mapped into IP using a linear number of variables. However, as to MMUCA, we need a quadratic number of variables, following the DIP mapping. Since decision variable x^m_{bk} means "use the k-th transformation of bid b at position m of the sequence" it is reasonable to think that if we can reduce the number of

Position	1	2	3	4	5	6	7	8	9	10	11
Seqn. 1	t_0	t_2			t_1		t_4				
Seqn. 2	t_0	t_1	t_2	t_4							
Seqn. 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.

positions at which each transformation can be used, we will be able to reduce the number of decision variables. In what follows we provide the rationale to achieve such reduction and to found CCIP.

Consider that after receiving a bunch of bids, we draw the relationships among goods and transformations, as shown in figure 1 (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 as obtained by DIP. 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 very same transformations, because t_2 lacks of enough input goods (g_2) to be used.

Firstly, when looking for solutions, we wonder whether we can avoid considering re-orderings of the solution sequence such as the one in row 3. That would largely reduce our search space. Secondly, since solutions at rows 1 and 2 are equivalent to the auctioneer, he would be happy with any of the two. Therefore, it is reasonable to pose whether we can constrain our search so that the number of re-orderings of row 2 (and hence of solutions equivalent to the one at row 2) that we consider is reduced.

Back to the example in figure 1 (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 establish an order among them because they form a cycle and then they can feed with input goods one another (they depend on one another). However, any permutation of the three could be valid for the 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 transformations without losing solutions. The way of imposing such constraints is via a *solution template*, a pattern that any solution sequence must fulfil to be considered. 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 $\{t_2, t_3, t_4\}$. 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.

Imposing constraints on the ordering of transformations drastically reduces the number of decisions. Thus, while DIP must decide for every single transformation whether it occupies or not every possible position in the solution sequence , a solution template constraints the positions each transformation can occupy. For instance, according to row 4 in table 1, transformation t_0 can only occupy position 1 and transformation t_2 can occupy either positions 3,4, or 5, but no other. DIP would check whether t_0 and t_2 can occupy any of the eleven possible positions in a solution sequence. For the example of figure 1, DIP would require $\ell \cdot |T| = 11 * 11$=121 decision variables, our new strategy (based on the analysis of the structure of the search space) would require 19 decision variables. Therefore, the search space is reduced from 2^{121} to 2^{19} alternatives.

In what follows, we formally analyse how we can extend the intuitions above to the general case in order to yield a new IP, the so called CCIP, by relying on the notion of dependence among transformations, and using it to constrain the positions at which a transformation can be used.

3.1 Transformation Dependencies and solution template

In this section, first we formally introduce the concept of *dependency* among transformations. Next, we introduce a function that constrains transformations to hold a limited number of positions within a solution sequence, that is, the solution *template*.

3.1.1 Transformation Dependencies

When solving the WDP, the auctioneer has to decide which transformations to buy and in which order he should use them. If he can a priori constrain the positions of transformations, he will reduce the search space. In order to constrain the positions of transformations, we have to formalise the concept of dependency among the transformations sent by bidders. Our aim is to define a dependency relationship among transformations such that, given two transformations t and t', provides the following possibilities:

- If no dependency holds among t and t' we can safely enforce any ordering among t and t' without losing solutions
- If t' depends on t and t does not depend on t' we can enforce t to appear before t' without losing solutions.
- If t' depends on t and t depends on t' we cannot enforce any ordering among them.

In the following, we describe how this can be done analysing the input and output goods of each transformation. We first define the 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). After that, we say that a dependency relationship between two transformations exists if there is a path that connects them in the TDG. That is, a transformation t' depends on another transformation t if t' has an input good that is also an output good of t (direct dependence) or if there is another transformation t'' such that t' depends on t'' and t'' depends on t (indirect dependence).

DEFINITION 3.1. *Given a set of bids in the XOR bidding language, the associated Transformation Dependency Graph (TDG) is a graph $TDG = (V, E)$ such that:*

- *Each transformation is a vertex: $V = T$,*
- *A directed arc connects two transformations t and t' iff there exists a good that is both output of t and input to t'. More formally,*

$$(t, t') \in E \iff \mathcal{O}_t \cap \mathcal{I}_{t'} \neq \emptyset$$

Figure 1(a) depicts an example of auction representing goods at trade as circles, transformations as squares, a transformation input goods as incoming arrows and its output goods as outgoing arrows. Figure 1 (b) depicts the TDG for the bids represented in figure 1(a).

Depending on the received bids, the TDG may or may not contain cycles. However, we have to assume that the graph is cyclic in the general case. In order to constrain the position of transformations, we will transform the cyclic TDG in an acyclic graph where the nodes that form a cycle are collapsed. The main idea is that the transformations contained in a cycle have to be considered *equivalent* ($\sim$). In Fig. 1(b) we can see a TDG with cycles. In Fig. 1(c) we identify the cycles (formally strongly connected components or SCCs[2]) in the graph. In Fig. 1(d) we can see the graph resulting from transforming (collapsing) each SCC into a node.

A (cyclic) graph defines a preorder $\lesssim$ on T. We denote this preorder as a pair $(T, \lesssim)$. The semantics of the preorder is that $t \lesssim t'$ iff a path exists between t and t'. Note that a preorder allows the existence of pairs t, t' such that $t \lesssim t'$ and $t' \lesssim t$.Given a set T equipped with a preorder $\lesssim$, we can define an equivalence relation $\sim$ on T as follows:

$$t \sim t' \iff t \lesssim t' \wedge t' \lesssim t \tag{10}$$

In our case $t \sim t'$ means that t depends on t' and t' depends on t, This means that the TDG has a cycle that contains t and t'. Hence, we will not be able to constrain the order among them.

It is possible to define a strict partial order over the quotient set $(T/_\sim, \prec)$ such that:

$$[t] \prec [t'] \iff t \lesssim t' \wedge t \not\gtrsim t' \tag{11}$$

Then, we say that $t \prec t'$ if $[t] \prec [t']$. This means that t' depends on t but not vice versa. Hence, we can enforce that t is used before t'.

We are now ready to formally define the concept of *dependence*. We recall that two transformations t, t' can be such that: (1) t depends on t' or t' depends on t but not both, or (2) t and t' are mutually dependent; or (3) t and t' do not depend on one another. More formally, we can differentiate the following three cases:

$t \prec t'$: t *depends* on t'. A one-way directed path between t and t' exists in the TDG. Then, all the transformations along the path connecting t to t' can contribute to increase the goods present in at least one of the inputs of t'. Hence, t' *depends* on their execution. For instance, in figure 1(a) we have that t_5 depends on t_2. In this case we must enforce that t comes before t' within the solution sequence if we do not want to lose valid solutions.

$t' \sim t$: t and t' are *mutually dependent*. There exist both a simple path between t and t' *and* one between t' and t. Therefore, they are part of a simple cycle of the TDG. For instance, in figure 1(a), we have that $t_2 \sim t_4$. Obviously, we cannot order them since the circularity of the relationship implies that they depend on each other. In order to prevent the loss of valid solutions, we cannot reject any of the two cases, t_4 before t_2 and t_2 before t_4 within a solution sequence.

$t \not\lesssim t'$ and $t' \not\lesssim t$: no path exists among t and t'. The relative positions of t and t' within the solution sequence do not affect the validity of the solution in any case. Then, it does not matter how t and t' are planned in the solution. Thus, we can randomly select the order between them.

3.1.2 Sequences with order

In what follows we assume that T is a non-empty finite set equipped with a preorder $(T, \lesssim)$. Our aim is to assess the positions to a-priori assign to transformations in such a way that the order established by the TDG is not violated. We explained in section 3.1.1 that we have to make sure that if a transformation t' depends on a transformation t (that is $t \prec t'$), they must be assigned positions such that t comes before t' in the sequence. Thus, the first step is knowing the valid solutions that respect the strict order imposed by $(T/_\sim, \prec)$. First, we illustrate the concept of *partial sequence*. A partial sequence is a sequence with "holes", meaning that there could be some positions of the sequence that are *empty*. A solution to the MMUCA WDP can be encoded as a partial sequence of transformations. After that, we define when a partial sequence fulfils an order relationship. Then, we define an order enforcing function as a template that, if fulfilled, guarantees the fulfilment of the order. Finally, we show that we can construct an order enforcing function for every strict order $(T/_\sim, \prec)$. These results pave the way for the construction of the new IP formulation in section 3.2.

DEFINITION 3.2. *A Partial Sequence over a non-empty finite set T is a **partial** function $K : \{1, \ldots, n\} \to T$, with $n \in \mathbb{N}$.*

Examples of partial sequences are in rows 1, 2, and 3 of table 1.

The fact the function is partial implies that some integers may not have an image. Such integers are the holes in the sequence that we previously mentioned. Now, we can define whether a partial sequence fulfils a strict order relationship.

DEFINITION 3.3. *We say that a partial sequence K over T fulfils the order relation $(T/_\sim, \prec)$ if:*

$$\forall i, j \in dom(K) \qquad [K(i)] \prec [K(j)] \Rightarrow i < j \tag{12}$$

This definition formally states that a partial sequence K fulfils the order relationship $\prec$ only if the relative order among

[2]Notice that [2] shows that the SCCs of a graph can be computed in time $\Theta(V + E)$.

transformations within K does not violate $\prec$. For instance, row 3 of table 1 does not fulfil the order relation defined in figure 1(d), whereas row 2 does.

We mentioned at the beginning of this section that our aim is to build a *template* that allows us to a-priori limit the set of positions that each transformation can hold within a solution sequence in such a way that no solution is lost. This is formally captured by the concept of $\mathcal{T}$*-bounded Order Enforcing Function*

DEFINITION 3.4. *Given a strict order $(T/_\sim, \prec)$ and a a multi-set $\mathcal{T} \in \mathbb{N}^T$, a $\mathcal{T}$-bounded Order Enforcing Function $S : \{1, \ldots, |\mathcal{T}|\} \rightarrow T/_\sim$ is a sequence of equivalence classes satisfying the following constraints:*

$$S(i) \prec S(j) \Rightarrow i < j \tag{13}$$

$$|S|_{[t]} = \sum_{t' \in [t]} |\mathcal{T}|_{t'} \quad \forall [t] \in T/_\sim \tag{14}$$

where $|S|_{[t]}$ is the number of times the equivalence class $[t]$ appears in the sequence S. Henceforth, S will denote a $\mathcal{T}$-bounded order enforcing function for $(T/_\sim, \prec)$.

A $\mathcal{T}$-bounded order enforcing function S limits the possible positions that the elements of each equivalence class can hold. Those positions are such that the strict order $(T/_\sim, \prec)$ is fulfilled. Analogously, S assigns to each equivalence class a set of allowed positions within a solution sequence. For instance, row 3 of table 1 does not fulfil the template in row 4, whereas row 2 does.

Equation 13 guarantees that any partial sequence that fulfils S fulfils the order relationship $(T/_\sim, \prec)$. Equation 14 ensures that enough positions are available to an equivalence class $[t]$ (for instance, if three units of transformation t_0 are offered, three positions must be allowed to t_0). Notice that there is no overlapping among the positions assigned to different equivalence classes in virtue of equation 13.

We employ S^{-1} to indicate the inverse of an enforcing function S. $S^{-1}([t])$ indicates the set of integers that map to the equivalence class $[t]$ via S. More formally:

$$S^{-1}([t]) = \{m \in \{1, \ldots, |\mathcal{T}|\} \mid S(m) = [t]\}$$

The following lemma guarantees that, for any order relationship arising from a set of bids, we can construct an order enforcing function.

LEMMA 3.1. *Given a strict order $(T/_\sim, \prec)$ and a multi-set $\mathcal{T} \in \mathbb{N}^T$ such that $\forall t \; |\mathcal{T}|_t \geq 1$, at least a $\mathcal{T}$-bounded order enforcing function S exists.*

Its proof can be found in [5]. This lemma means that if we have a strict order among transformations, we can always construct an order enforcing function that restricts the positions that can be assigned to those transformations in a way that, if the order enforcing function is fulfilled, so will be the strict order. In the next section we will use it to construct a function that constrains the positions where transformations can be used, imposing the strict ordering $\prec$ that was defined in the section 3.1.1.

3.2 Connected component IP solver

The aim of this section is to introduce a new IP that improves solver DIP. We call the improved solver, described in the remaining of this section, solver CCIP. We simplify DIP in two ways: (1) we get rid of a set of IP constraints, following a method similar to the one proposed in [6]; and (2) we reduce the number of decision variables (the associated search space) and simplify the constraints by considering as possible solutions only partial sequences fulfilling a $\mathcal{D}$-bounded order enforcing function S and exclude all other solutions. With this aim, we employ the order enforcing function resulting from applying lemma 3.1 to the order generated by the TDG. In section 3.2.1 we detail how to remove or simplify some constraints, and finally, in section 3.2.2, we introduce the IP formulation of solver CCIP.

3.2.1 Reducing the number of constraints

In DIP equation 8 is applied at each position m of the solution sequence to check that enough input goods are present to perform the transformation assigned to position m. Analogously, recall that equation 9 states that at the end of the sequence at least $\mathcal{U}_{out}$ goods are available to the auctioneer.

The combination of those two constraints plus restricting the positions transformations can hold makes some of those constraints redundant. In particular, we can get rid of constraint 8 at each position m where none of the transformations assigned to position m belong to any cycle of the graph. This can be seen as a refinement of the technique employed in [6].

Intuitively, equation 9 is a *global* condition enforcing that at the end of the sequence the global input-output balance at each good of the net in figure 1(a) is positive. On the other hand, equation 8 is *local* to each position, and enforces that enough input goods are present at each position. If the transformation assigned to position m does not belong to a cycle, the local condition is implied by the global one.

Notice that, by definition, each time an equivalence class contains $n > 1$ transformations, each transformation in the equivalence class belongs to a simple cycle of length n. However, when the equivalence class contains a single transformation, the constraint in equation 9 will only be included if the transformation depends on itself (self-loop).

3.2.2 Detailed IP formulation

In this section we employ the same notation as in section 2.1.We represent each solution with a partial sequence $J : \{1, \ldots, |\mathcal{D}|\} \rightarrow T$. We employ decision variables similar to the ones employed for solver DIP : 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 order enforcing function resulting from applying lemma 3.1 to the order generated by the TDG. In CCIP solver, 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. $x^m_{bk} = 0 \; \forall m \notin S^{-1}([t_{bk}])$. By means of this operation we manage to drastically reduce the number of decision variables and the complexity of the constraints.

Next, analogously to section 2.1, we employ the following auxiliary decision variables. First, x_b is a binary variable that takes value one if bid Bid_b is accepted. Furthermore, x_{bk} is an integer variable that represents the number of positions that transformation t_{bk} holds in the solution sequence.

In what follows we explicitly state the constraints that a valid solution has to fulfil in solver CCIP. Those constraints correspond to equations (4) to (9).

1. x_{bk} is obtained by summing up x^m_{bk} over the positions

m assigned to $[t_{bk}]$ ($S^{-1}([t_{bk}])$):

$$x_{bk} = \sum_{m \in S^{-1}([t_{bk}])} x^m_{bk} \qquad \forall b \ \forall k \tag{15}$$

because we can remove from equation (4) decision variables x^m_{bk} for all $m \notin S^{-1}([t_{bk}])$.

2. We are interested in that at most one transformation can hold each position:

$$\sum_{t_{bk} \in S(m)} x^m_{bk} \leq 1 \quad \forall m \tag{16}$$

Notice that the sum is only over the transformations of a single equivalence class[3].

3. We impose the cardinality semantics of a combinatorial bid b:

$$x_{bk} = x_b \cdot |\mathcal{D}_b|_{t_{bk}} \qquad \forall b \in B \ \forall k \in \{1, \ldots, |D_b|\} \tag{17}$$

4. We impose that the XOR semantics of a bid:

$$\sum_{b \in B_l} x_b \leq 1 \qquad \forall l \in L \tag{18}$$

5. We enforce that enough goods are available to use the corresponding transformations at each position of the solution sequence. As argued in section 3.2.1, equation 19 must be added only if the transformations assigned to positions m belong to a simple cycle. We define the set L_F of positions m where the equation must be added as $m \in L_F$ iff the transformations in the equivalence class $S(m)$ belong to a simple cycle. Now we can impose:

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{bk} \in S(l)} x^l_{bk} \cdot [\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g)] \geq \sum_{t_{bk} \in S(m)} x^m_{bk} \cdot \mathcal{I}_{bk}(g) \qquad \forall g, \forall m \in L_F \tag{19}$$

6. We enforce that the goods available to the auctioneer at the end of the solution sequence is at least $\mathcal{U}_{out}$:

$$\mathcal{U}_0(g) + \sum_m \sum_{t_{bk} \in S(m)} x^m_{bk} \cdot [\mathcal{O}_{bk}(g) - \mathcal{I}_{bk}(g)] \geq \mathcal{U}_{out}(g) \ \forall g \tag{20}$$

Hence, solving the MMUCA WDP is equivalent to optimise the objective function:

$$\max \sum_b x_b \cdot p_b \tag{21}$$

subject to constraints 15 to 20.

The number of decision variables in the above integer program is $VCCIP = \sum_{q \in T/\sim} |q||\mathcal{D}|_q$ where $|q|$ is the number of different transformations in the equivalence class q and $|\mathcal{D}|_q$ is the number of different copies in $\mathcal{D}$ of transformations that belong to q. Hence, whereas in DIP the number of variables is given by the number of transformations (being equal to $\ell|T|$) in CCIP they depend on the topology of transformations in the specific problem we try to solve. However, it is easy to see that $\ell \leq VCCIP \leq \ell|T|$.

[3]The constraints in equation 16 enforce that the solution is a partial sequence, that is no more than one transformation can be assigned to the same position of the sequence.

3.3 The Proof of Equivalence

We argued at the beginning of section 3 that if we assume ordering not to be relevant as long as the sequence is enabled, we can say that DIP generates redundant equivalent solutions. Next, in section 3.2 we introduced a solver that finds a subset of those solutions: the solutions fulfilling the order enforcing function provided by the TDG. Thus, in order to prove that no solution is missed by CCIP we have to show that any solution that solver DIP can find can be reordered into a solution of solver CCIP.

Furthermore, we have to check that the other way around is also true. That is, we have to make sure that solver CCIP does not introduce new solutions that are not solution to solver DIP. More formally, we have to prove that:

1. COROLLARY 3.1. *Any solution found by solver DIP can be reordered into a solution to solver CCIP.*
2. COROLLARY 3.2. *Any solution found by solver CCIP is a solution to solver DIP.*

The proof of these two corollaries is very simple if we have previously demonstrated that:

1. THEOREM 3.1. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils all the constraints of solver CCIP.*
2. THEOREM 3.2. *Given a partial sequence J, solution to solver CCIP, it fulfils all the constraints of DIP.*

The proof of theorems 3.1 and 3.2 and corollaries 3.1 and 3.2 can be found in [5].

4. EMPIRICAL COMPARISON

As shown in section 3.2.2, CCIP provides a more concise IP formulation than DIP in terms of both number of variables and constraints. In this section, we empirically analyse whether the achieved reduction also translates into a reduction in computational cost.

Firstly, since the number of decision variables of both DIP and CCIP depends on the number of transformations within the submitted bids. Therefore, to compare their computational performance and to analyse their scalability, we have chosen to observe their solving times as the number of transformations increases. As detailed in [7], the generator creates three types of transformations: I-transformations (no output goods), O-transformations (no input goods) and IO-transformations (input and output goods).

In order to run an empirical comparison we have employed randomly generated MMUCA WDPs using the (publicly available) artificial data set generator to evaluate MMUCA WD algorithms proposed in [7]. We have set the artificial data set generator's parameters for this experiment as listed in Table 2.

We ran our experiments as follows. We generated MMUCA WDP instances with transformations within the range $[0, 300]$. In fact, we sampled the interval to generate 50 WDP instances every 20 transformations. Both solvers, DIP and CCIP were fed with the very same WDP instances. We solved each WDP instance using IP implementations of both solvers on CPLEX 10.1, recording both the solutions and solving times. Moreover, we set a maximum time limit to 4800 seconds for each solver to find a solution for each WDP instance. Whenever any of the solvers exceeded the time limit, we stopped it to subsequently mark the WDP as

n_{goods}	20
$n_{IO\_market\_transformations}$	$n\_Transformations/3$
max_price	100
σ_{prices}	0.05
$p_{good\_requested}$	0.3
$\mu_{add\_new\_transformation}$	1.0
$\sigma_{add\_new\_transformation}$	0
$\mu_{add\_new\_XOR\_clause}$	1.0
$\sigma_{add\_new\_XOR\_clause}$	0
$p_{good\_in\_input}$	0.2
$p_{good\_in\_output}$	0.1
α	0.1
p_ITransformations	0.6
p_OTransformations	0.1
allow_cycles	1

Table 2: Artificial generator parameter values.

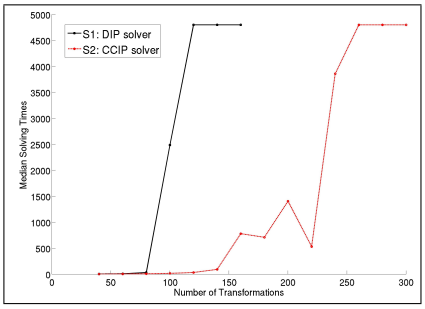


Figure 2: Comparison between DIP and CCIP.

time exceeded. After that, we set its solving time to the time limit to subsequently record it. Notice that we only considered feasible WDP instances to calculate solving times since the time required by CPLEX to prove unfeasibility is (usually) significantly lower than the time required to find an optimal solution. Finally, we ran all tests on a Dell Precision 490 with double processor Dual-Core Xeon 5060 running at 3.2 GHz with 2Gb RAM on Linux 2.6.

Figure 2 shows the results of our experiments. It depicts the median of the solving times obtained when varying the number of transformations. Observe that the MMUCA WDP computational cost increases exponentially as the number of transformations grows for both solvers. Nonetheless, given a time limit, CCIP was able to solve problems with more than twice the number of transformations that DIP did solve. Indeed, whereas 120 represents the empirical limit on the number of transformations for DIP, CCIP starts reaching the time limit when solving WDP instances containing more than 250 transformations. Furthermore, for WDPs with close to 100 transformations, DIP is in median about 70 times slower than CCIP. This ratio rapidly increases as the number of transformations gets close to 120, namely to the limit on the number of transformations.

5. CONCLUSIONS AND FUTURE WORK

MMUCAs offer a high potential to be employed for the automated assembly of supply chains of agents. However, in order for MMUCAs to be effectively applied to SCF, we must ensure computational tractability while preserving optimality. In the IP proposed in [1] the number of variables grows quadratically with the number of transformations mentioned in the bids, thus limiting the application of MMUCAs to SCF. In this paper we have proposed an IP, CCIP, for MMUCAs that dramatically improves the computational efficiency of the IP reported in [1] by taking advantage of the topological characteristics of the WDP. At this aim, we have founded our analysis on observing the structure of the WDP that results after establishing *dependence relationships* among transformations. At the theoretical level, we have proved that CCIP brings a drastic reduction of the search space to be explored (decision variables) to solve the WDP. At the empirical level, we have observed that CCIP: (i) can deal with WDPs with more than twice as many transformations as DIP; and (ii) can significantly reduce the computation time (by a factor larger than 70). Therefore, we argue that our main contribution has been to make headway in the applicability of MMUCAs to SCF.

As to future work, in order to outperform CCIP we plan to explore the design of a local algorithm. Although solutions may be sub-optimal with a local approach, the number of transformations that can be dealt with is expected to be larger, and hence the size of the supply chain scenarios we could tackle.

This work has been partially supported by the Spanish Ministry of Education and Science (grants 2006-5-0I-099, TIN-2006-15662-C02-01, CONSOLIDER CSD2007-0022, INGENIO 2010), and by the Generalitat de Catalunya (grant 2005-SGR-00093).

6. REFERENCES

[1] J. Cerquides, U. Endriss, A. Giovannucci, and J. A. Rodriguez-Aguilar. Bidding languages and winner determination for mixed multi-unit combinatorial auctions. In *Proc. of the 20th Intl. Joint Conferences on Artif. Intelligence (IJCAI)*, pages 1221–1226, 2007.

[2] T. Cormen. *Introduction to Algorithms*. MIT Press, 2001.

[3] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2006.

[4] Y. Engel, M. P. Wellman, and K. M. Lochner. Bid expressiveness and clearing algorithms in multiattribute double auctions. In *Proceedings of the 7th ACM conference on Electronic commerce*, pages 110–119, Ann Harbor, USA, 2006.

[5] A. Giovannucci, J. Cerquides, and J. A. Rodriguez-Aguilar. Proving the correctness of the CCIP solver for MMUCA. Technical report, IIIA-CSIC, 2007.

[6] A. Giovannucci, J. Rodríguez-Aguilar, J. Cerquides, and U. Endriss. Winner determination for mixed multi-unit combinatorial auctions via petri nets. In *20th Intl. Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2007)*, Hawaai, USA, May 2007.

[7] M. Vinyals, A. Giovannucci, J. Cerquides, P. Meseguer, and J. A. Rodriguez-Aguilar. Towards a realistic bid generator for mixed multi-unit combinatorial auctions. *14th Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion RCRA*, 2007.

[8] W. Walsh and M. Wellman. Decentralized supply chain formation: A market protocol and competitive equilibrium analysis. *Journal of Artificial Intelligence Research*, 19:513–567, 2003.