Savings in Combinatorial Auctions through Transformation Relationships

Andrea Giovannucci[1], Jesús Cerquides[2], and Juan A. Rodríguez-Aguilar[1]

[1] Artificial Intelligence Research Institute, IIIA-CSIC, Spain
{andrea,jar}@iiia.csic.es
[2] Dept. de Matemàtica Aplicada i Anàlisi, Universitat de Barcelona, Barcelona, Spain
cerquide@maia.ub.es

Abstract. In a previous work we extended the notion of multi-unit combinatorial reverse auction (MUCRA) by adding a new dimension to the goods at auction. A buyer can express transformability relationships among goods: some goods can be transformed into others at a transformation cost. Through this new auction type, a buyer can find out what goods to buy, to whom, and what transformations to apply to the acquired goods in order to obtain the best savings. The main focus of the paper is to perform some preliminary experiments to quantitatively assess the potential savings that a buying agent may obtain in considering transformation relationships.

1 Introduction

Since many reverse (or direct) auctions involve the buying (or selling) of a variety of complementary assets, combinatorial auctions [5] (CA) have recently deserved much attention in the literature. In particular, a significant amount of work has been devoted to the problem of selecting the set of winning bids [13, 3]. Nonetheless, to the best of our knowledge, while the literature has considered the possibility to express relationships among goods on the bidder side —such as complementarity and substitutability (e.g. [6],[14])—, the impact of the production relationships among the different assets to sell/buy on the bid-taker side has been only addressed so far in [7].

Consider that a company devoted to the assembly and repairing of personal computers (PCs) requires to assembly new PCs in order to fulfil his demand. Figure 1 graphically represents the way a PC is assembled. Our graphical description largely borrows from the representation of Place/Transition Nets (PTN) [10], a particular type of Petri Net. Each circle (corresponding to a PTN *place*) represents a good. Horizontal bars connecting goods represent assembly/disassembly operations, likewise *transitions* in a PTN. Assembly and disassembly operations are labelled with an indexed *t*, and shall be referred to as *transformation relationships* (t-relationships henceforth). In particular, t_1 and t_2 stand for assembly operations. An arc connecting a good to a transformation indicates that the good is an *input* to the transformation, whereas an arc connecting a transformation to a good indicates that the good is an *output* from the transformation. In our example, a CPU, RAM, USB and Empty Board are *input goods* to t_2, whereas Motherboard is an *output good* of t_2. Thus, t_2 represents the way a motherboard is manufactured (assembled). The labels on the arcs connecting *input goods* to transitions, and

the labels on the arcs connecting *output goods* to transitions indicate the units required of each *input good* to perform a transformation and the units generated per *output good* respectively. In figure 1, the labels on the arcs connected to t_2 indicate that 1 motherboard is assembled from 1 CPU, 4 RAM units, 3 USBs and 1 empty motherboard. Each transformation has an associated cost every time it is carried out. In our example, assembling a motherboard via t_2 costs €7.

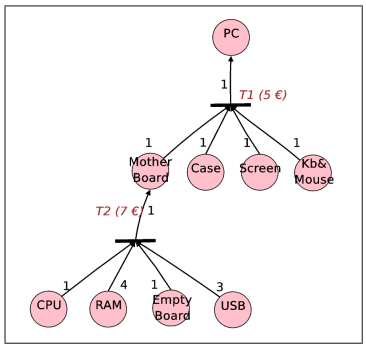


Fig. 1. Graphical representation of an RFQ with t-relationships.

Say that the company's warehouse contains most of the components composing each PC. However, there are no components to assemble motherboards. Therefore, the company would have to start an automated sourcing process to acquire such components. For this purpose, it may opt for running a combinatorial reverse auction [14] with qualified providers. But before that, a buying agent may realise that he faces a decision problem: shall he buy the required components to assemble them in house into motherboards, or buy already-assembled motherboards, or opt for a *mixed purchase* and buy some components to assemble them and some already-assembled motherboards? This concern is reasonable since the cost of components plus transformation (assembly) costs may eventually be higher than the cost of already-assembled motherboards. Hence, the buying agent requires a combinatorial reverse auction mechanism that provides: (a) a language to express required goods along with the relationships that hold among them; and (b) a winner determination solver that not only assesses what goods to buy and to whom, but also the transformations to apply to such goods in order to obtain the initially required ones. In the first part of this paper we summarize how we solved these issues in [7].

Firstly, since state-of-the-art multi-unit CA only allow buying agents to require a fixed number of units per good when expressing his requirements (henceforth referred to as *Request for Quotation* (RFQ)), we undo such constraint to allow for the introduction of t-relationships among goods. Thus, we introduce a formal definition of a *Transformability Network Structure* (TNS) that largely borrows from Place/Transition Nets [10], where transitions stand for t-relationships and places stand for goods. Secondly, we extend the formalisation of multi-unit combinatorial reverse auction (MUCRA),

departing from the model in [13], to introduce transformability by applying the expressiveness power of multi-set theory. Additionally, we provide a mapping of our formal model to integer programming that takes into account t-relationships to assess the winning set of bids along with the transformations to apply in order to obtain a buying agent's initial requirements. At this point, it is important to make clear that we focus on the following central problem: given a collection of bids on bundles and a collection of t-relationships, find a set of non-conflicting bids that minimises cost. Thus, it is beyond the scope of this paper to design any CA mechanisms that consider t-relationships from a game-theoretic perspective. Thirdly, we empirically analyse the benefits of introducing t-relationships with respect to a classical multi-unit combinatorial reverse auction. We explain how to generate artificial negotiation scenarios to compare in a fair way the two types of auctions. Then, we experimentally observe that the benefits of introducing t-relationships for a buyer increase when its production process is more efficient than the providers' one, that is, when its transformation costs are smaller.

The paper is organised as follows. In section 2 we provide some background knowledge on place/transition nets and multi-sets. In section 3 we present a formal model of multi-unit combinatorial reverse auctions with t-relationships among goods, along with its winner determination problem and its mapping to integer programming. Section 4 thoroughly describes the data set generator and some experimental results. Finally, section 5 draws some conclusions and outlines directions for future research.

2 Background

A *multi-set* is an extension to the notion of set, considering the possibility of *multiple appearances* of the same element. A *multi-set* $\mathcal{M}_X$ over a set X is a function $\mathcal{M}_X : X \rightarrow \mathbb{N}$ mapping X to the cardinal numbers. For any $x \in X$, $\mathcal{M}_X(x) \in \mathbb{N}$ is called the *multiplicity* of x. An element $x \in X$ *belongs* to the multi-set $\mathcal{M}_X$ if $\mathcal{M}_X(x) \neq 0$ and we write $x \in \mathcal{M}_X$. We denote the set of multi-sets over X by X_{MS}. Given the multi-sets $\mathcal{M}_S, \mathcal{M'}_S \in S_{MS}$, their union is defined as: $\mathcal{M}_S \cup \mathcal{M'}_S(x) = \mathcal{M}_S(x) + \mathcal{M'}_S(x)$.

Following [10], a *Place/Transition Net Structure* (PTNS) is a tuple $N = (G, T, A, E)$ such that: (1) G is a set of *places*; (2) T is a finite set of *transitions* such that $P \cap T = \emptyset$; (3) $A \subseteq (G \times T) \cup (T \times G)$ is a set of *arcs*; (4) $E : A \rightarrow \mathbb{N}^+$ is an *arc expression* function. A *marking* of a PTNS is a multi-set over G. A PTNS with a given initial marking $\mathcal{M}_0 \in G_{MS}$ is denoted by $PTN = (N, \mathcal{M}_0)$ and it is called a *Place/Transition Net* (PTN). The graphical representation of a PTNS is composed of the following graphical elements: places are represented as circles, transitions are represented as bars, arcs connect places to transitions or transitions to places, and E labels arcs with values (see figure 1).

A *step* is a non-empty and finite multi-set over T. A step $\mathcal{S} \in T_{MS}$ is *enabled* in a marking $\mathcal{M} \in G_{MS}$ if the following property is satisfied: $\forall g \in G \sum_{t \in \mathcal{S}} E(g,t)\mathcal{S}(t) \leq \mathcal{M}(g)$.

Let step $\mathcal{S}$ be enabled in a marking $\mathcal{M}_1$. Then, $\mathcal{S}$ may *occur*, changing the $\mathcal{M}_1$ marking to another $\mathcal{M}_2 \in G_{MS}$ marking. Setting $Z(g,t) = E(t,g) - E(g,t)$ $\mathcal{M}_2$ is expressed as: $\forall g \in G\ \mathcal{M}_2(g) = \mathcal{M}_1(g) + \sum_{t \in \mathcal{S}} Z(g,t)\mathcal{S}(t)$. Moreover, we say that marking $\mathcal{M}_2$ is *directly reachable* from marking $\mathcal{M}_1$ by the occurrence of step $\mathcal{S}$, and we denote it by $\mathcal{M}_1[\mathcal{S} > \mathcal{M}_2$.

A *finite occurrence sequence* is a finite sequence of steps and markings:$\mathcal{M}_1[\mathcal{S}_1 > \mathcal{M}_2 \ldots \mathcal{M}_n[\mathcal{S}_n > \mathcal{M}_{n+1}$ such that $n \in \mathbb{N}$ and $\mathcal{M}_i[\mathcal{S}_i > \mathcal{M}_{i+1}\ \forall i \in \{1, \ldots, n\}$. $\mathcal{M}_1$ is called the *start marking*, while $\mathcal{M}_{n+1}$ is called the *end marking*. The *firing count multi-set* $\mathcal{K} \in T_{MS}$ associated to a finite occurrence sequence is the union of all its steps: $\mathcal{K} = \bigcup_{i \in \{1,2,\ldots,n\}} \mathcal{S}_i$.

A marking $\mathcal{M}''$ is *reachable* from a marking $\mathcal{M}'$ iff there exists a finite occurrence sequence having $\mathcal{M}'$ as start marking and $\mathcal{M}''$ as end marking. We denote it as $\mathcal{M}'[\mathcal{S}_1\ \ldots \mathcal{S}_n > \mathcal{M}''$, where $n \in \mathbb{N}$. Furthermore the start and end markings are related by the following equation:

$$\forall g \in G \qquad \mathcal{M}''(g) = \mathcal{M}'(g) + \sum_{t \in \mathcal{K}} Z(g,t)\mathcal{K}(t). \tag{1}$$

The set of all possible markings reachable from a marking $\mathcal{M}'$ is called its *reachability set*, and is denoted as $R(N, \mathcal{M}')$.

In [12], Murata shows that in an *acyclic* Petri Net a marking $\mathcal{M}''$ is *reachable* from a marking $\mathcal{M}'$ iff there exists a multi-set $\mathcal{K} \in T_{MS}$ such that expression 1 holds (which is equivalent to say that the state equation associated to a PTN admits an integer solution). As a consequence, when a Petri Net is acyclic, the reachability set $R(N, \mathcal{M}')$ is represented as:

$$\begin{aligned} R(N, \mathcal{M}') = \{\mathcal{M}'' \mid \exists \mathcal{K} \in T_{MS} : \forall g \in G \\ \mathcal{M}''(g) = \mathcal{M}'(g) + \sum_{t \in \mathcal{K}} Z(g,t)\mathcal{K}(t)\}. \end{aligned} \tag{2}$$

3 MUCRA with t-Relationships

In this section we formalise the winner determination problem (WDP) for MUCRA with t-relationships (MUCRAtR) borrowing from our work in [7].

3.1 Transformability Network Structures

A Transformability Network Structure describes the different ways in which goods can be transformed and at which cost. More formally, a *transformability network structure* (TNS) is a pair $S = (N, C_T)$, where $N = (G, T, A, E)$ is a Place-Transition Net Structure and $C_T : T \rightarrow \mathbb{R}^+$ is a cost function. The cost function associates a *transformation cost* to each *t-relationship*. In this context we associate: (1) the *places* in G to a set of goods to negotiate upon; (2) the *transitions* in T to a set of *t-relationships* among goods; (3) the *directed arcs* in A along with their weights E to the specification of the number of units of each good that are either consumed or produced by a transformation.

The values of C and the values of E label respectively transitions (between parenthesis) and arcs in figure 1.

Given a Place/Transition net $PTN = (N, \mathcal{M}_0)$, if we consider $\mathcal{M}_0$ as a good configuration, PTN defines the space of good configurations *reachable* by applying transformations to $\mathcal{M}_0$. The application of transformations is obtained by firing transitions on PTN. Hereafter, we consider the concepts of *transformation step*, *enabling*

of a transformation step, *occurrence of a transformation step* and *transformation sequence* as the counterparts to, respectively, *step*, *enabling of a step*, *occurrence of a step*, and *finite occurrence sequence* on a PTN.

We also need to define the concept of transformation cost, taking into account the cost of transforming good configuration $\mathcal{M}_0$ into another good configuration $\mathcal{M}_1 \in R(N, \mathcal{M}_0)$ by means of some transformation sequence $J = (\mathcal{S}_1, \ldots, \mathcal{S}_n)$. The $\mathcal{K}$ firing count multi-set associated to J accounts for the number of times a transition in the sequence is fired. Thus, the cost of transforming good configuration $\mathcal{M}_0$ into good configuration $\mathcal{M}_1$ amounts to adding the transformation cost of each transition in the firing count multi-set $\mathcal{K}$ associated to J. We assess the transformation cost associated to J as $C_{TS}(J) = \sum_{\mathcal{S} \in J} \sum_{t \in \mathcal{S}} C_T(t)\mathcal{S}(t) = \sum_{t \in \mathcal{K}} C_T(t)\mathcal{K}(t)$. Notice that the transformation cost of a transformation sequence only depends on its firing count multi-set.

3.2 WDP for MUCRA with t-Relationships

In a classic MUCRA scenario, a *Request for Quotation* (RFQ) can be expressed as a multi-set $\mathcal{U} \in G_{MS}$ whose multiplicity indicates the number of units required per good. In the example of figure 1, if $\mathcal{U}(motherboard) = 1$,$\mathcal{U}(CPU) = 1$,$\mathcal{U}(RAM) = 2$,$\mathcal{U}(EmptyBoard) = 1$,$\mathcal{U}(USB) = 2$, $\mathcal{U}$ would be representing a buying agent's need for 1 motherboard (M), 1 CPU (C), 1 empty board (E), 2 RAM units (R), and 2 USB (U) connectors. Nonetheless, since t-relationships hold among goods, the buyer may have different alternatives depending on the bids he receives. If we represent each bid as a multi-set $\mathcal{B} \in G_{MS}$, whose multiplicity indicates the number of units offered per good, the buyer might, for instance, have the following alternatives: (1) buy all items as requested, formally $\mathcal{M}_0 = \{M, C, R, R, E, U, U\}$; and (2) $\mathcal{M}_1 = \{C, R, R, R, R, E, U, U, U\} \bigcup \{C, R, R, E, U, U\}$, do not buy any motherboard, but buy its parts (1 CPU, 4 RAM units, 1 Empty Board, and 3 USB connectors) instead to manufacture it at cost $C_T(t_2) = €7$. The overall cost of the purchase results from the cost of the acquired units plus the additional transformation cost. Notice that both alternatives allow the buyer to obtain his initial requirement, though each one at a different cost. The goal of the WDP is to assess what alternative to select optimally.

We begin by defining the set of possible auction outcomes. Given a set of bids B, a possible auction outcome is a pair (W, J), where $W \subseteq B$, and $J = (\mathcal{S}_1, \ldots, \mathcal{S}_n)$ is a transformation sequence, such that the application of J to $PTN = (N, \cup_{\mathcal{B} \in W} \mathcal{B})$ allows a buyer to obtain a good configuration that fulfils his requirements in $\mathcal{U}$. More formally, the set of possible auction outcomes is defined (assuming free disposal) as:

$$\Omega = \{(W, J), W \subseteq B \mid \exists \mathcal{X} \in G_{MS}(\bigcup_{\mathcal{B} \in W} \mathcal{B})[J > \mathcal{X}, \mathcal{X} \supseteq \mathcal{U}\}. \tag{3}$$

To each auction outcome (W, J) we associate an *auction outcome cost* as follows:

$$C_O(W, J) = \sum_{\mathcal{B} \in W} C_B(\mathcal{B}) + C_{TS}(J) \tag{4}$$

where $C_B : B \rightarrow \mathbb{R}^+$ stands for the bid cost function.

Given a set of bids B, an RFQ $\mathcal{U} \in G_{MS}$, and a transformability network structure $S = (N, C_T)$, the winner determination problem for an MUCRA with t-relationships

amounts to assessing the auction outcome $(W^{opt}, J^{opt}) \in \Omega$ that minimises the auction outcome cost function C_O. Formally,

$$(W^{opt}, J^{opt}) = \arg \min_{(W,J)\in\Omega} C_O(W, J) \tag{5}$$

3.3 Mapping to Integer Programming

In section 2, we defined the reachability set according to equation 2 for the case of acyclic Petri nets. Thus, if we restrict to the case of acyclic TNS, a finite occurrence sequence J is completely specified by its firing count vector $\mathcal{K}$. Then, we can rewrite expressions 3 and 4 respectively as follows:

$$\Omega = \{(W, \mathcal{K}), W \subseteq B, \mathcal{K} \in T_{MS} \mid \exists \mathcal{X} \in G_{MS}(\bigcup_{\mathcal{B}\in W} \mathcal{B})[\mathcal{K} > \mathcal{X}, \mathcal{X} \supseteq \mathcal{U}\}. \tag{6}$$

$$C_O(W, \mathcal{K}) = \sum_{\mathcal{B}\in W} C_B(\mathcal{B}) + C_{TS}(\mathcal{K}) \tag{7}$$

where $C_{TS}(\mathcal{K}) = \sum_{t\in\mathcal{K}} C_T(t)\mathcal{K}(t)$. Hence, the WDP when considering acyclic TNSs can be restated, from equation 5, to assess:

$$(W^{opt}, \mathcal{K}^{opt}) = \arg \min_{(W,\mathcal{K})\in\Omega} C_O(W, \mathcal{K}) \tag{8}$$

We can model the problem of assessing $(W^{opt}, \mathcal{K}^{opt})$ as an Integer Programming problem. For this purpose, we need to associate integer variables to the elements in: (1) a generic subset of bids ($W \subseteq B$); and (2) a generic firing count multi-set ($\mathcal{K}$).

In order to represent W we assign a binary decision variable $x_{\mathcal{B}}$ to each bid $\mathcal{B} \in B$, standing for whether $\mathcal{B}$ is selected ($x_{\mathcal{B}} = 1$) or not ($x_{\mathcal{B}} = 0$) in W. A multi-set is uniquely determined by its mapping function $\mathcal{K} : T \to \mathbb{N}$. Hence, we represent a multi-set $\mathcal{K} \in T_{MS}$ by considering an integer decision variable q_t for each $t \in T$. Each q_t represents the multiplicity of element t in the $\mathcal{K}$ multi-set.

Then, consider $G = \{g_1, \ldots, g_n\}, n \in \mathbb{N}$, is a finite set of goods, $T = \{t_1, \ldots, t_r\}, r \in \mathbb{N}$, is a finite set of transitions, $U = \{u_1, \ldots, u_n\}$ is a finite set of requirements, where $u_i = \mathcal{U}(g_i)$ stands for the number of units requested of good g_i, and $B = \{\mathcal{B}_1, \ldots, \mathcal{B}_m\}, m \in \mathbb{N}$, is a finite set of bids. Furthermore, for each bid $\mathcal{B}_j \in B$ we construct a pair $\langle p_j, [a_j^1, \ldots, a_j^n]\rangle$ where $p_j = p(\mathcal{B}_j)$ stands for the bid price and $a_j^i = \mathcal{M}_{\mathcal{B}_j}(g_i)$ stands for the units offered of good g_i by bid $\mathcal{B}_j$. Therefore, the problem represented by expressions 6, 7 and 8 can be translated into the following integer program:

$$min[\sum_{j=1}^{m} x_j p_j + \sum_{k=1}^{r} q_k c(t_k)] \tag{9}$$

$$s.t.\ \forall 1 \leq i \leq n\ \sum_{j=1}^{m} a_j^i x_j + \sum_{k=1}^{r} q_k m_k^i \geq u_i \tag{10}$$

where $x_j \in \{0, 1\}\ \forall 1 \leq j \leq m$ stands for whether bid $\mathcal{B}_j$ is selected or not, and $m_k^i = Z(g_i, t_k)$. Hence, notice that each element m_k^i is in fact obtained from the *incidence*

matrix[12] of the place-transition net within a TNS[3]. Notice that leaving the $q_t(t \in T)$ decision variables unbounded is utterly unrealistic because it is equivalent to say that the buyer has got the capability of applying as many transformations as required to fulfil $\mathcal{U}$.Therefore it is realistic to assume that the number of in-house transformations that he can apply are constrained. Hence, we add the following constraints: $\forall t \in T\ q_t \in \{0, 1, \ldots, max_t\}$, where $max_t \in \mathbb{N}$.

The new integer program defined by expressions 9 and 10 can be readily implemented with the aid of an optimisation library. Notice too that our integer program can be clearly regarded as an extension of the integer program we must solve for an MUCRA as formalised in [15]. Thus, the second component of expression 9 changes the overall cost as transformations are applied, whereas the second component of expression 10 makes sure that the units of the selected bids fulfil with a buyer's requirements taking into account the units consumed and produced by transformations.

4 Empirical Evaluation

The main purpose of our experiments is to empirically evaluate the benefits of introducing t-relationships in a multi-unit combinatorial reverse auction. Our experiments artificially generate different data sets, each one composed of a a TNS, a buyer's requirements, and a collection of combinatorial bids. Each data set stands for a multi-unit combinatorial reverse auction problem. We solve the WDP for each auction problem regarding and disregarding t-relationships to quantitatively assess the potential savings that a buyer/auctioneer may obtain thanks to t-relationships. In order to solve the WDP for an MUCRA, as formalised in [15], we exploit the equivalence to the multi-dimensional knapsack problem pointed out in [9]. In order to solve the WDP for an MUCRA with t-relationships we implement the integer program represented by expressions 9 and 10.

4.1 Data Set Generation

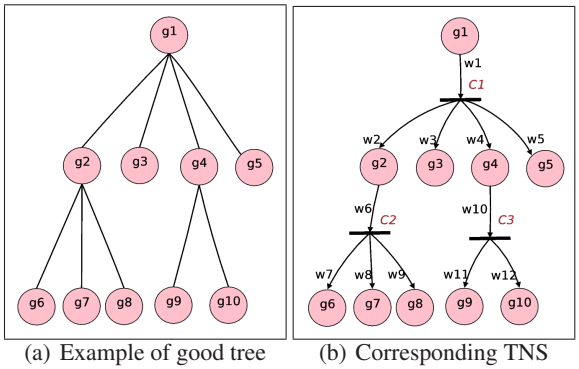
As outlined above, each data set shall be composed of: (1) a TNS; (2) an RFQ; and (3) a set of combinatorial bids. The WDP for an MUCRA will consider the last two components of the data set, whereas the WDP for an MUCRAtR will consider them all. When we create a TNS associated to an auction, we should enforce a certain coherence among the items prices, and thus among bid prices. This concern is taken into account at bid generation time, and it will be explained in section 4.1. Thus, in the following we detail the generation of TNSs, RFQs and bids.

First we detail the TNS creation. Recall from section 3 that if we restrict to the case of an acyclic TNS, then the WDP for an MUCRAtR can be formulated as an integer program. Thus, we shall focus on generating acyclic TNSs for our data sets. For this purpose, we create TNSs fulfilling the following requirements: (a) each transition receives a single input arc; (b) each place has got no more than one input and one output

[3] Given a TNS (N, C_T) where $N = (G, T, A, E)$ is a place-transition net with r transitions and n places, its incidence matrix $M = [m_k^i]$ is an $r \times n$ matrix of integers such that $m_k^i = E(t_k, g_i) - E(g_i, t_k)$ represents the difference of tokens of place g_i produced and consumed by transition t_k.

arc; and (c) there exists a place, called *root place*, that has got only output arcs. Figure 2(b) depicts an example of a TNS that satisfies such requirements.

We have designed an algorithm to construct acyclic TNSs that is composed of two sequential sub-algorithms. Firstly, algorithm 1 creates a tree structure from which a second algorithm constructs a TNS by creating transformations with costs and attaching weights to the edges connecting places with transformations. Figure 2 illustrates the extension of a tree to an acyclic TNS. A distinguishing feature of our algorithm is that, since we aim at empirically assessing the potential savings when considering t-relationships independently of TNSs' shapes, it is capable of constructing acyclic TNSs that may largely differ in their shapes (e.g. with different widths, depths, either symmetric or asymmetric,...etc).



(a) Example of good tree (b) Corresponding TNS

Fig. 2. Extension of a tree to a TNS.

Algorithm 1 constructs a tree of n nodes (goods) and r branching points (i.e. nodes with children). It represents the tree as a vector of n components, named $Tree$. The value of each vector component is a pointer to the index of the father good. For instance, if the i-th component of $Tree$ is set to j, it means that good g_j is the father of good g_i. Given this representation, it is easy to build a random tree. The rough idea is: (1) build a null vector $Tree$ of n components; (2) set to 0 the first component of $Tree$; (3) set each element of the vector $Tree[j]$ to a random number chosen in $[1, j-1]$. This constructive process first builds the root (g_1), then assigns a child (g_2) to the root (g_1), then assigns a child (g_3) to a random node within $\{g_1, g_2\}$,...etc. More in detail. While (line 2) the exact required number (r) of father goods are not generated, the algorithm proceeds as follows. At each iteration of the loop in line (5), the algorithm assesses the father of each good g_j. Firstly, it stores the indexes of the goods that have at least one child into $Fathers$ through the EXTRACT-NONZERO-ENTRIES function applied over $Tree$. If the father goods that have been generated are less than r (line 8), the father of g_j is randomly assessed (line 9), otherwise its father is selected out of the goods that have already children (line 11). Finally, the algorithm returns a tree (line 18).

From the tree generated by algorithm 1, a second algorithm extends it to generate a TNS taking as inputs the minimum and maximum arc weights (w_{min} and w_{max} respectively) and the minimum and maximum transformation costs (c_{min} and c_{max} respec-

Algorithm 1 CREATE-TREE(n, r)

```
1: k ← 0
2: while k ≠ r do
3:    Tree ← Empty-Vector(n)
4:    k ← 0
5:    for j ← 2 to n do
6:       Fathers ← EXTRACT-NONZERO-ENTRIES(Tree)
7:       k ← length[Fathers]
8:       if k < r then
9:          father ← EXTRACT-RANDOM-NUMBER(1, j − 1)
10:      else
11:         father ← EXTRACT-RANDOM-ELEMENT(Fathers)
12:      end if
13:      Tree[j] ← father
14:   end for
15:   Fathers ← EXTRACT-NONZERO-ENTRIES(Tree)
16:   k ← length[Fathers]
17: end while
18: return Tree
```

tively). The algorithm assigns to each branching node of the input tree a *t-relationship* having as input good the branching node, and as output goods the children of the very same branching node. Then it attaches to each created *t-relationship* a transformation cost randomly chosen in $[c_{min}, c_{max}]$. Finally, the algorithm assigns to each arc in the created TNS an integer random weight in $[w_{min}, w_{max}]$. The outputs of the algorithm are thus the incidence matrix M of the associated TNS and a transformation cost vector C. Notice that M and C are enough to characterise a TNS and to build expressions 9 and 10 of the Integer Program.

Secondly, we detail the RFQ creation. Considering the notion of requirements as described in section 3.3, an RFQ is represented as a set $U = \{u_1, \ldots, u_n\}$ where $u_i = \mathcal{U}(g_i)$ stands for the number of units requested of good g_i. We generate each number of required units $u_i \in U$ from a uniform discrete distribution $U[u_{min}, u_{max}]$, where u_{min} and u_{max} are two parameters standing for the minimum and maximum number of units required per item respectively. Notice that this differs from Leyton-Browns's approach [11] since we have not included any constraint ensuring that each data set involves the same total number of required units.

Finally, we focus on generating bids. In order to create a data set, the most delicate task is concerned with the generation of a collection of combinatorial bids. To the best of our knowledge, and as already pointed out in [3], no real-world benchmarks of combinatorial bids do exist. Thus, with the purpose of comparing winner determination algorithms, we find two approaches in the CA literature to generate artificial data sets: (1) design a specific generator for the MUCA/MUCRA domain, as in [11]; or (2) given the equivalence of the WDP for an MUCA/MUCRA to the multi-dimensional knapsack problem [9], employ the very same data sets used for evaluating MDKP solvers (e.g. [2],[4] and [1]). Unfortunately, we cannot benefit from any previous results in the literature since they do not take into account the novel notion of t-relationship, and thus the generated data set never reflects the relationships among goods. For instance, consider the t_2 transformation in figure 1. It is clear that it is unrealistic that a bid offers a motherboard cheaper than some of its components. This fact motivated the need for substantially changing the approach in [11] to coherently introduce t-relationships.

In order to generate a *plausible* set of combinatorial bids, we assume that all providing agents produce goods in a *similar* manner. In other words, they share similar production structures (similar TNSs). This means that given a particular good, two providing agents will roughly use the same raw materials (components), but acquired at different prices and transformed at different costs. We believe that this assumption is utterly realistic. Under this assumption, we can artificially generate bids that can be used for both MUCRA and MUCRAtR. Nonetheless, one might be tempted to intuitively think that bids that take into account t-relationships are not valid whatsoever for an MUCRA since that would lead to an unfair comparison with MUCRAtR. Not at all. Upon reception of an RFQ, providing agents do compose bids taking into account their *own* t-relationships, and thus their *own* production costs. Thereafter, in an MUCRA scenario, the winner determination algorithm shall solely focus on finding an optimal allocation for the required goods, whereas in an MUCRAtR scenario, the winner determination algorithm shall assess whether an optimal allocation that considers the buying agent's t-relationships can be obtained. Therefore, the difference is that an MUCRAtR winner determination algorithm does consider and exploit both the buying agent's t-relationships along with the implicit transformation cost within each bid, while an MUCRA winner determination algorithm does not.

Recall from section 3.3 that from each $\mathcal{B}_j \in B$ we can construct a pair $\langle p_j, [a_j^1, \ldots, a_j^n] \rangle$ where p_j stands for the bid price and $[a_j^1, \ldots, a_j^n]$ for the units offered per good.

First of all, we describe how to generate the units to offer for each bid based on algorithm 2. This algorithm receives several input parameters, namely: n (number of goods); m (number of bids to generate); $p_{bid\_density}$ (parametrizes a geometric distribution [8] used for obtaining the number of goods that jointly appear in a bid); and $p_{offered\_units}$ (parameter of a geometric distribution used for obtaining the number of units offered per good.).

Algorithm 2 GENERATE-OFFER-VECTOR$[n, m, p_{bid_density}, p_{offered_units}]$

```
1: for j ← 1 to m do
2:    [a_j^1, ..., a_j^n] ← Empty-Vector[n]
3:    k ← SAMPLE-GEOMETRIC-DISTRIBUTION[p_bid_density]
4:    for l ← 1 to k do
5:       i ← randomly choose in [1, .., n] such that a_j^i = 0
6:       u ← 0
7:       while u ≠ 0 do
8:          u ← SAMPLE-GEOMETRIC-DISTRIBUTION[p_offered_units]
9:          a_j^i ← u
10:      end while
11:   end for
12: end for
```

Algorithm 2 proceeds as follows. For each bid, it firstly obtains the number of goods to jointly bid for from a geometric distribution (line 3). It subsequently obtains the number of units to offer per good (line 8) from another geometric distribution. We employed geometric distributions since they provide large variances.

Once generated the units to offer per good for all bids, we must assess all bid prices. This process is rather delicate when considering t-relationships if we want to guarantee

the production of a set of plausible bids. As outlined above, we make the assumption that all providing agents in the market share similar production structures, which in turn are similar to the buying agent's one. In practice, our providing agents use the same TNS as the buying agent, though each one has his own transformation costs, which in turn are assessed as a variation of the buying agent's ones. With this assumption in mind, next we describe how to produce a unitary price for each good offered in a given bid. For this purpose, we depart from the value of a parameter, p_{root}, standing for the average price of the root good of a TNS 4.1 (e.g. the root good in figure 2(b) is g_1).

The first step of our pricing policy calculates the unitary price of the root good for each bid under the assumption that all providing agents have similar values for such good. Thus, for each bid $\mathcal{B}_j \in B$, its unitary price for the root good is assessed as $P_{root,j} = p_{root} \cdot |\lambda|$, where λ is sampled from a distribution $N(1, \sigma_{root\_price})$[4].

After that, our pricing policy proceeds as follows. Given a particular bid and a good whose unitary price is known, this is propagated down the TNS through the transition it is linked to towards its output goods. In fact, the value to propagate results from weighting the unitary price (considering the arc connecting the input good to the transition) and adding the transformation cost of the transition. The resulting value is unevenly distributed among the output goods according to a *share factor* randomly assigned to each output good. For instance, consider the TNS in figure 2(b) and a bid $\mathcal{B}_j$ such that: its unitary cost for g_1 is $P_{g_1,j} = €50$, *its* transformation cost (different from the buying agent's one) for t_1 is €10, and $w_1 = 2$. In such a case, the value to split down through t_1 towards g_2, g_3, g_4, and g_5 would be $50 * 2 + 10 = €110$. Say that g_2 is assigned 0.2 as share factor. Thus, $110 * 0.2 = €22$ would be allocated to g_2. Finally, that amount should be split further to obtain g_2 unitary price if $w_2 > 1$. For instance, if $w_2 = 2$, then the final unitary price for g_2 would be €11.

Generalising the example above, in what follows we provide a general way of calculating the unitary price for any good in a given bid. Let $\mathcal{B}_j$ be a bid represented by $\langle p_j, [a_j^1, \ldots, a_j^n] \rangle$ and g a good such that $a_j^g \neq 0$. Let t be a transition such that g is one of its output goods, and $father(g)$ is its single input good[5]. Besides, we note as G' the set of output goods of t. Then, we obtain $P_{g,j}$, the unitary price for good g of bid $\mathcal{B}_j$ as follows:

$$P_{g,j} = \frac{P_{father(g),k} \cdot |M[father(g), t]| + c(t) \cdot |\nu|}{M[g, t]} \, \omega_g \tag{11}$$

where $P_{father(g),k}$ is the unitary price for good $father(g)$ in a bid $\mathcal{B}_k \neq \mathcal{B}_j$; $|M[father(g), t]|$ indicates the units of good $father(g)$ that are input to transition t; ν is a value obtained from a distribution $N(\mu_{production\_cost}, \sigma_{production\_cost})$ that weighs transformation cost $c(t)$; $M[g, t]$ indicates the number of units of good g that are output by transition t; and ω_g is the share factor for good g.

Several remarks apply to equation 11. Firstly, the share factors for output goods must satisfy $\sum_{g' \in G'} \omega_{g'} = 1$. Secondly, it may surprise the reader to realise that the

[4] We consider the absolute value of the sample since the large tails of the normal distributions could occasionally bring about negative values.

[5] Recall that our method to construct acyclic TNSs ensures that there is a single input good per transition.

value to propagate down the TNS ($P_{father(g),k}$) is collected from a different bid. We enforce this crossover operation among bids to avoid undesirable *cascading effects* that occur when we start out calculating unitary prices departing from either high or low unitary root prices. In this way we avoid to produce non-competitive and extremely competitive bids respectively that could be in some sense regarded as noise that could eventually lead to diverting results. Notice that after applying our pricing policy we obtain P, an $n \times m$ matrix storing all unitary prices.

Finally, from equation 11 we can readily obtain the bid price for a each bid $\mathcal{B}_j \in B$ as $p_j = \sum_{i=1}^{n} a_j^i \cdot P_{i,j}$. To summarise, the parameters that is possible to set when creating an MUCRAtR scenario are listed along the first column of table 1.

Table 1. Parameters characterising our experimental scenario.

Parameter	***Explanation***	***Value***
n	The number of items	20
r	The number of transitions	8
u_{min}, u_{max}	The minimum/maximum number of units required per item	10/10
w_{min}, w_{max}	Minimum/Maximum arc weight	1/5
c_{min}, c_{max}	Minimum/Maximum Transformation cost	10/10
m	The number of bids to generate	1000
$p_{offered\_goods}$	Statistically sets the number of items simultaneously present in a bid	$\{0.2, 0.3, 0.4, 0.5\}$
$p_{offered\_units}$	Statistically sets the number of unit offered per item	$\{0.2, 0.3, 0.4, 0.5\}$
p_{root}	Average price of the $root$ good	1000
μ_{root_price} σ_{root_price}	Parameters of a Gaussian distribution weighting the $root$ price p_{root}	1 0.01
$\mu_{production\_cost}$ $\sigma_{production\_cost}$	Parameters of a Gaussian distribution setting the production costs difference between buyer and providers	0.8:0.1:1.8 0.1

4.2 Experimental Settings and Results

In order to measure the benefits provided by the introduction of t-relationships among goods we compute the cost of the optimal outcome, that is, the cost of the winning bid set for MUCRA (OC^{MUCRA}) and the cost of the winning bid set plus transformations for MUCRAtR ($OC^{MUCARtR}$). We define the savings increment (SI) as: $SI = 100 * \frac{OC^{MUCRA} - OC^{MUCRAtR}}{OC^{MUCRA}}$ The larger the index, the higher the benefits that a buyer can expect to obtain by using an MUCRAtR instead of an MUCRA.

The third column of table 1 shows the parameter configuration used in our experiments. In this preliminary experiment we consider the outcomes produced when varying three parameters, namely $\mu_{production\_cost}$, $p_{offered\_units}$ and $p_{bid\_density}$. In particular $\mu_{production\_cost}$ takes on values in $[0.8, 1.8]$, and we set $p_{offered\_units} = p_{bid\_density} \in \{0.2, 0.3, 0.4, 0.5\}$. Our experimental hypothesis are: (1) the SI index increases as the buyer's transformation costs decrease (larger $\mu_{production\_cost}$) with respect to the providers' ones; and (2) SI will increase when increasing the bid density (low $p_{bid\_density}$) and the number of offered units (low $p_{offered\_units}$). This hypothesis is motivated by the following reasons. Firstly, increasing the $\mu_{production\_cost}$ parameter models the fact that in-house transformations are cheaper, therefore more likely to

be employed. An MUCRAtR improves savings with respect to an MUCRA as more in-house transformations are employed. In such a case the sets of winning bids of MUCRA and MUCRAtR largely differ among them. Secondly, when increasing the bid density and the number of offered units, it is very difficult for an MUCRA to allocate offers so that they perfectly fit the requirements. An MUCRAtR, instead, can employ the *free-disposal* goods obtained by *imperfect allocations* as inputs to transformations, so as to obtain other required goods. Therefore, we will analyse how the difference in production costs between the buyer and providers affects SI. The difference among the buyer's transformation costs and the average transformation costs of providers is set by the $\mu_{production\_cost}$ parameter. We expect that, as the average transformation costs of providers increases with respect to the buyer's ones, so do the benefits of using an MUCRAtR instead of an MUCRA. In fact, the experimental results do strongly agree with our hypothesis. Figure 3 depicts the results when varying $\mu_{production\_cost}$ from 0.8 to 1.8. The x axis represents the values of the $\mu_{production\_cost}$ parameter. The y axis represents the corresponding values of SI. Each point is obtained by averaging 30 runs of the experiment. The legend lists the value of the $p_{offered\_goods} = (p_{offered\_units})$ parameter[6]. As $\mu_{production\_cost}$ increases, so do savings. As the bids' density and the number of offered units jointly increase, so does savings.

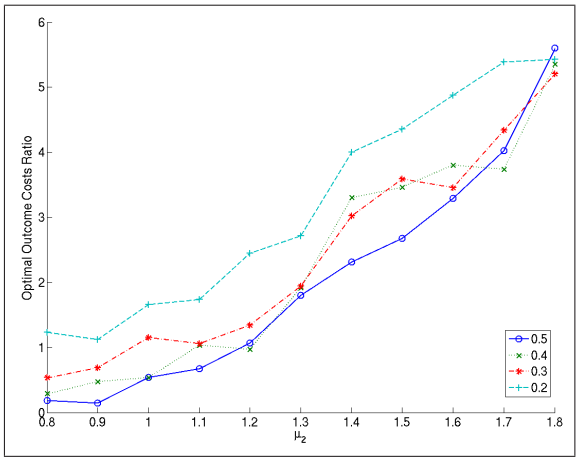


Fig. 3. Varying the $\mu_{production\_cost}$ parameter

5 Conclusions and Future Work

We have performed a set of preliminary experiments to quantitatively assess the potential savings of employing an MUCRAtR instead of an MUCRA in different scenarios.

[6] Notice that an increment in $p_{offered\_units} = p_{bid\_density}$ stands for a decrement in the number of offered units and in the bids' density, since they are parameters of a geometric distribution.

The main conclusion that stems from the experiments is that the cheaper the in-house t-relationships available to the buyer, the more he can benefit from applying them. The second conclusion is that when the number of offered units and the bid density increase simultaneously, so does the savings (SI).

The novel idea presented in the paper opens several paths to future development. The first being a further development of the empirical experiments and to carry out a sound sensitivity analysis. That would allows us to fully characterize the auction scenarios where exploiting t-relationships is expected to bring larger savings than an MUCRA. On the other hand, we aim at comparing MUCRAtR and MUCRA in terms of computational complexity, performing some scalability experiments, too.

Finally, notice that it was beyond the scope of this paper any mechanism design analysis. That is left out for future work.

Acknowledgements – This work was partially funded by the Spanish Science and Technology Ministry as part of the Web-i-2 project (TIC-2003-08763-C02-00). Giovannucci Andrea enjoys the BEC.09.01.04/05-164 CSIC scholarship.

References

1. http://elib.zib.de/pub/packages/mp-testdata/ip/sac94-suite/.
2. http://www.cs.ualberta.ca/∼holte/combinatorialauctions/.
3. A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *ICMAS 2000*, pages 39–46, Boston, MA, 2000.
4. P. Chu and J. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
5. S. de Vries and R. Vohra. Combinatorial auctions: A survey. *INFORMS Journal of Computing*, 15(3):284–309, 2003.
6. Y. Fujishima, K. Leyton-Brown, and 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)*, pages 548–553, August.
7. A. Giovannucci, J. A. Rodrguez-Aguilar, and J. Cerquides. Multi-unit combinatorial reverse auctions with transformability relationships among goods. In *First International Workshop on Internet and Network Economics (WINE 2005)*, volume 3828 of *LNCS*, Hong Kong, China, December 2005. www.iiia.csic.es/∼andrea/papers/WINE318a.pdf.
8. R. V. Hogg and J. Ledolter. *Engineering Statistics*. MacMillan Publishing Company, 1987.
9. R. C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Lecture Notes in Computer Science*, volume 2056. Springer-Verlag, Heidelberg.
10. K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, volume 1, chapter 2, pages 78–80. Springer, 1997.
11. K. Leyton-Brown, Y. Shoham, and M. Tennenholtz. An algorithm for multi-unit combinatorial auctions. In *American Association for Artificial Intelligence (AAAI)*, 2000.
12. T. Murata. Petri nets: Properties, analysis and applications. In *IEEE*, volume 77, pages 541–580, 1989.
13. M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinational auctions. *Management Science*, 44(8):1131–1147, 1998.
14. T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.
15. T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In *AAMAS 2002*, pages 69–76, Bologna, Italy, 2002.