Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 36 (2009) 1090–1109
computers & operations research
www.elsevier.com/locate/cor

Exact algorithms for the matrix bid auction☆

D.R. Goossens*, F.C.R. Spieksma

ORSTAT, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium

Available online 30 January 2008

Abstract

In a combinatorial auction, multiple items are for sale simultaneously to a set of buyers. These buyers are allowed to place bids on subsets of the available items. A special kind of combinatorial auction is the so-called *matrix bid auction*, which was developed by Day [Expressing preferences with price-vector agents in combinatorial auctions. PhD thesis, University of Maryland; 2004]. The matrix bid auction imposes restrictions on what a bidder can bid for a subsets of the items. This paper focusses on the winner determination problem, i.e. deciding which bidders should get what items. We discuss the computational complexity of the winner determination problem for a special case of the matrix bid auction. We present two mathematical programming formulations for the general matrix bid auction winner determination problem. Based on one of these formulations, we develop two branch-and-price algorithms to solve the winner determination problem. Finally, we present computational results for these algorithms and compare them with results from a branch-and-cut approach based on Day and Raghavan [Matrix bidding in combinatorial auctions. Manuscript; 2006].

Keywords: Combinatorial auction; Matrix bids; Winner determination; Computational complexity; Branch-and-price

1. Introduction

In an auction where bidders are interested in multiple items, it is often the case that a bidder values a set of items higher or lower than the sum of the values this bidder attaches to the individual items. These so-called complementarity or substitution-effects, respectively, may be bidder-specific. A combinatorial auction is a way to make use of this synergy phenomenon, by allowing a bidder to express his preferences to a greater extent than for individual items only. Indeed, in such an auction a buyer is allowed to place bids on a subset of the items, sometimes called a *bundle*. The auction is concluded when the auctioneer decides to accept some of the bids and allocate the items accordingly to the bidders. For a thorough discussion on combinatorial auctions, integrating contributions on many interesting aspects from both theory and practice, we refer to the book edited by Cramton et al. [1].

In a combinatorial auction in its most general form, bidders can bid whatever amount they please on any subset of items. The problem of deciding which bidders should get what items in order to maximize the auctioneer's revenue is called the winner determination problem. Even if every bidder bids only on subsets of size 2 and all bids have a value equal to 1, this problem is *NP*-hard (Van Hoesel and Müller [2]). Moreover, the winner determination problem cannot

☆ This research was partially supported by FWO Grant No. G.0114.03; an extended abstract corresponding to this paper has been accepted for the 6th Workshop on Experimental Algorithms, Rome.
* Corresponding author.
E-mail address: Dries.Goossens@econ.kuleuven.be (D.R. Goossens).

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.12.009

be approximated to a ratio of $\max(K^{\varepsilon-1}, m^{\varepsilon-1/2})$ in polynomial time for any fixed $\varepsilon > 0$ (unless $P = ZPP$), where K is the number of bundles on which a bid has been made, and m is the number of items (Sandholm [3]).

Numerous attempts to cope with this computational complexity can be found in literature. One approach is to impose restrictions on what a bidder can bid for these sets. A common restriction on a bidder's preferences is that they should be non-decreasing, i.e. the valuation for a set S_1 cannot be higher than the valuation for a set S_2 if S_1 is a subset of S_2. Another restriction can be that the preferences should be supermodular. This means that the sum of valuations for two sets should not be higher than the sum of the valuation of the union of both sets and the valuation of the intersection of both sets. If there are only two bid functions a bidder can have, both of them non-decreasing, integer valued and supermodular, then the winner determination problem of this auction can be solved in polynomial time (de Vries and Vohra [4]). For other results based on restricting preferences, we refer to Nisan [5] and Tennenholtz [6].

The matrix bid auction, which was developed by Day [7], is also a combinatorial auction in which restrictions are imposed on what a bidder can bid. In the matrix bid auction, each bidder must submit a strict ordering (or ranking) of the items in which he[1] is interested. We assume that for each bidder, the extra value an item adds to a set is determined only by the *number* of higher ranked items in that set, according to the ranking of that bidder. Let G be the set of items, indexed by i and B the set of bidders, indexed by j. The ordering of the items is denoted by r_{ij}, which is item i's position in bidder j's ranking, for each $i \in G$ and $j \in B$. This ordering should be strict in the sense that for each bidder j, $r_{i_1 j} \neq r_{i_2 j}$ for any pair of distinct items i_1 and i_2. For instance, if $r_{ij} = 2$, item i is bidder j's second highest ranked item. Furthermore, each bidder j specifies values b_{ijk}, which correspond to the value the bidder is prepared to pay for item i given that it is the kth highest ranked item in the set that bidder j is awarded. The b_{ijk} values allow to determine the value bidder j attributes to any set $S \subseteq G$. Indeed, bidder j's bid on a set S is denoted as $b_j(S)$ and can be computed as

$$b_j(S) = \sum_{i \in S} b_{i,j,k(i,j,S)}, \tag{1}$$

where $k(i, j, S)$ is the ranking of item i amongst the items in the set S, according to bidder j's ranking. Notice that Eq. (1) assumes that no externalities are involved, i.e. a bidder's valuation depends only on the items he wins, and not for instance on the identity of the bidders to whom the other items are allocated. Furthermore, the matrix bid auction is a single-unit combinatorial auction. This means that for each item that is auctioned, only one unit of this item is available. The winner determination problem is, given the bids $b_j(S)$ for each set S and each bidder j, to determine which bidder is to receive which items, such that the total winning bid value is maximized. Notice that we assume that each bidder pays his bid for the subset he wins.

Observe that the value for index k of item i in bidder j's bid can never be higher than the rank r_{ij}. This allows us to arrange the values b_{ijk} as a lower triangular matrix for each bidder j, where the rows correspond to the items, ordered by decreasing rank and the columns correspond to values for k. Hence, the name matrix bid (with order). Notice also that bidder j's ranking r_{ij} does not necessarily reflect a preference order of the items. If an item is highly ranked, this merely means that its added value to a set depends on less items than the added value of a lower ranked item. Furthermore, we make no assumption regarding the b_{ijk} values. Indeed, these values may be negative, e.g. to reflect the disposal cost of an unwanted item. Specifying a sufficiently large negative value can also keep the bidder from winning this item in the first place. For a more elaborate discussion of the expressiveness of matrix bids and their relation to well-known micro-economic properties, we refer to Goossens [8].

As an example, we consider the following matrix bid, where an airline company expresses its preferences for the right to fly between London and Paris, Paris and Madrid, and London and Berlin. Let us assume this airline company has its home base in London:

London-Paris	2		
Paris-Madrid	−8	3	
London-Berlin	6	6	0

[1] He can be replaced by she (and his by her).

Consider now the value this airline company j attributes to the combination of the flights London–Paris and London–Berlin. Observe that for this choice of S, London–Paris is the highest ranked item (i.e. $k(\textit{London–Paris}, j, S) = 1$), and London–Berlin is the second highest ranked item (i.e. $k(\textit{London–Berlin}, j, S) = 2$). We find using (1):

$$\begin{aligned} b_j(S) &= b_{\text{London–Paris},j,k(\text{London–Paris},j,S)} + b_{\text{London–Berlin},j,k(\text{London–Berlin},j,S)} \\ &= b_{1,j,1} + b_{3,j,2} \\ &= 2 + 6 = 8. \end{aligned}$$

Notice that the airline company is not willing to bid more than 8 for any set of flights. The airline company also feels that the flight Paris–Madrid is only interesting if this flight can be connected to its home base. Indeed, no combination of flights without linking Paris–Madrid to London will result in a positive valuation, because the company charges a (disposal) cost of 8 if it gets Paris–Madrid without a connecting flight. Furthermore, the airline company can only handle two flights. Indeed, it will never pay an extra amount for a third flight.

1.1. Motivation

There are several reasons for investigating a combinatorial auction with matrix bids. First, bids in any practical combinatorial auction are likely to posses some structure. In literature, we find descriptions of both theoretical structures (see e.g. Leyton-Brown and Shoham [9], Nisan [5], and Rothkopf et al. [10]) and structures in practice (see e.g. Bleischwitz [11] and Goossens et al. [12]). Capturing and understanding this structure is important, not only since it allows to develop algorithms that can be more efficient than algorithms for a general combinatorial auction, but also because it improves our understanding of various properties of an auction.

Second, matrix bid auctions allow for a faster computation due to the restriction on the preferences that is assumed. Indeed, from their computational experiments, Day and Raghavan [13] conclude that the computation time for the general combinatorial auction is higher and grows much faster than for the matrix bid auction. Moreover, they manage to solve the winner determination problem for matrix bid auctions with 72 items, 75 bidders and over 10^{23} bids, whereas for the general combinatorial auction, the largest instances that can be solved have 16 items, 25 bidders, and less than 10^9 bids.

Finally, the matrix bid auction also offers a compact way of representing preferences. Indeed, each bidder only needs to communicate an ordered list of m items and $m(m+1)/2$ matrix bid entries, which is far less than bids for each of the 2^m possible sets of items in a general combinatorial auction. We do recognize that choosing a ranking of the items and filling the matrix bid with appropriate values might not be a trivial task for the bidder. However, we developed a procedure that recognizes whether a given collection of bids can be translated into a matrix bid, and—in case this is not possible—an algorithm that approximates this collection of bids by a matrix bid in a way that does not expose the bidder to paying more than he stated for any set of items (see Goossens [8]).

1.2. Our contribution

In Section 2, we strengthen a result by Day [7] that shows that the matrix bid auction winner determination problem is *NP*-hard, by studying a special case where all bidders have an identical ranking of the items. We show that this problem is *APX*-hard. However, given a fixed number of bidders, it can be solved in polynomial time. We discuss a mathematical programming formulation by Day [7] and compare it to a formulation based on the set packing problem. We find that both formulations are equally strong. Based on the set packing formulation, we develop two branch-and-price algorithms to solve the winner determination problem in Section 4. Finally, in Section 5, we present our computational results and compare them with results from the branch-and-cut approach by Day and Raghavan [13].

2. Computational complexity

The key assumption in the matrix bid auction is that for each bidder, the extra value an item adds to a set depends only on the number of higher ranked items in that set, according to the ranking of that bidder. Despite this restriction, the winner determination problem of the matrix bid auction remains *NP*-hard (Day [7]). Even if each bidder has the

same ranking of the items, the matrix bid auction winner determination problem remains *NP*-hard. Moreover, unless $P = NP$, there exists no polynomial-time approximation scheme (PTAS) for this problem.

Theorem 1. *There exists no PTAS for the winner determination problem for the matrix bid auction even when the ranking of the items is identical for each bidder, unless* $P = NP$.

Proof. We consider the winner determination problem for the matrix bid auction where all bidders have an identical ranking. We refer to this problem as MBI. The reduction is from the three-dimensional matching (3DM) problem. The 3DM problem is described as follows: given a set $M \subseteq X \times Y \times Z$ of triples, where each of the sets X, Y and Z has exactly q elements, find the largest matching in M. Kann [14] shows that it is *NP*-hard to decide whether there exists a matching of size q, or whether every matching has a size of at most $(1-\delta)q$ for some fixed $\delta > 0$ (see also Petrank [15]).

Every instance of 3DM can be reduced to an instance of MBI in polynomial time. Suppose that the $3q$ elements of the sets X, Y, and Z correspond to $3q$ items and that each 3-element subset in M corresponds to a bidder. We pick an arbitrary ordering of the items and let this be the ranking of the items for each bidder. Each bidder thus has a matrix bid with this ranking and with the following entries. The highest ranked item of the triple corresponding to the bidder gets a value of 1 in the first column, the second highest ranked item gets a value of 2 in the second column, and the third highest ranked item gets a value of 3 in the third column. All other entries get a value of zero.

If an instance of 3DM has a matching of size q, then the corresponding instance of MBI has a solution of value $6q$. Indeed, a solution of 3DM consists of q pairwise disjoint 3-element subsets, corresponding to q bidders in MBI. Each supplier has a bid of 6 for the 3 items represented by the 3-element subset. Accepting these bids leads to a sum of winning bids equal to $6q$. Since every element of $X \cup Y \cup Z$ occurs exactly once in the solution of 3DM, every item will also be auctioned exactly once in the MBI solution.

If our instance of 3DM has a matching of size at most $(1-\delta)q$, at most $(1-\delta)q$ entries with value 3 in the matrix bids can be used, resulting in a MBI solution value of $(1-\delta)6q$. Notice that for a maximal solution value, we need to use a maximal number of entries with value 3. The number of items remaining is $3q - 3(1-\delta)q = 3\delta q$. Each pair of these items adds at most 3 to the solution value, resulting in a maximal solution value for MBI of

$$(1-\delta)6q + \frac{9\delta q}{2} = \left(6 - \frac{3}{2}\delta\right) q.$$

Consequently, a PTAS for MBI would imply that we could distinguish between instances of 3DM with a matching of size q and instances where every matching has a size of at most $(1-\delta)q$, which is an *NP*-hard problem (Kann [14]). □

Theorem 1 implies that the winner determination problem for the matrix bid auction where bidders have an identical ranking of the items is *NP*-hard, which strengthens the complexity result by Day [7]. In Theorem 1, the number of bidders is part of the input. In the case that the number of bidders is fixed (and we still assume identical rankings), the winner determination problem can be solved in polynomial time.

Theorem 2. *The winner determination problem for a matrix bid auction with a fixed number of bidders, all having an identical ordering of the items, can be solved in polynomial time.*

Proof. We will show that the winner determination problem for a matrix bid auction with a fixed number of bidders, say n, all having an identical ranking r of the items, say $1, 2, \ldots, m$, can be solved by solving a longest path problem on an acyclic graph involving $O(m^{n+2})$ nodes and $O(nm^{n+2})$ arcs.

This graph contains nodes indexed by $(i, s_1, s_2, \ldots, s_n, k)$, a source, and a sink. The index i refers to item i and ranges from 1 to m. The indices s_j, with $j \in \{1, 2, \ldots, n\}$, and k range from 0 to r_i, with $\sum_j s_j + k = r_i$. There are arcs from each node $(i, s_1, s_2, \ldots, s_n, k)$ to $(i+1, s'_1, s'_2, \ldots, s'_n, k')$, provided that $s'_j \geqslant s_j$ for all $j \in \{1, 2, \ldots, n\}$, and that $k' \geqslant k$. Furthermore, there is an arc from the source to each node $(1, s_1, s_2, \ldots, s_n, k)$, and an arc from each node $(m, s_1, s_2, \ldots, s_n, k)$ to the sink. The arc from node $(i, s_1, s_2, \ldots, s_n, k)$ to $(i+1, s'_1, s'_2, \ldots, s'_n, k')$ has a cost of b_{i+1,j,s'_j} where j is the index for which $s'_j = s_j + 1$, if $k' = k$. If $k' = k+1$, then this arc has a cost of zero. All arcs

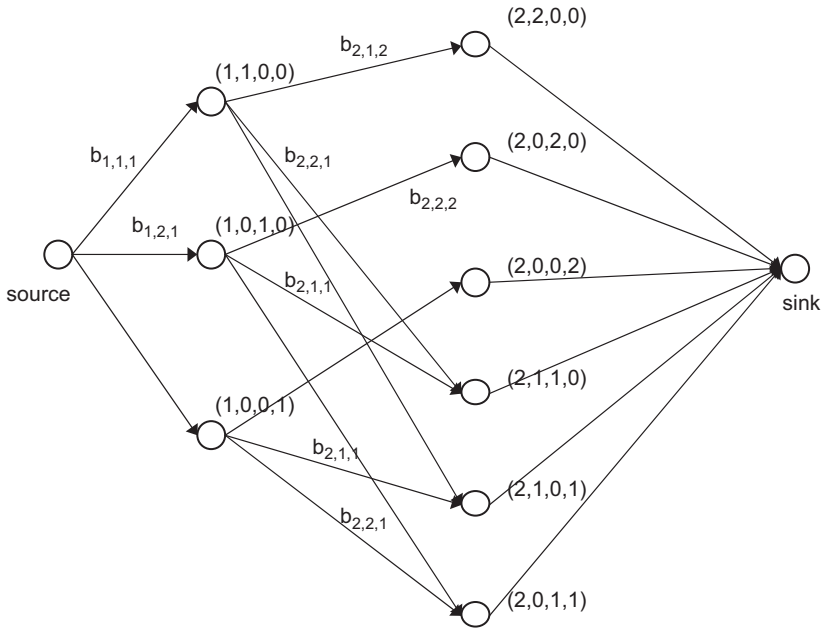

Fig. 1. Illustration of the graph for 2 items and 2 bidders.

to the sink also have a cost of zero. The graph is depicted in Fig. 1 for a setting with 2 items and 2 bidders. All arcs without indication of the corresponding cost have a cost equal to zero.

The graph described above should be interpreted as follows. Each node $(i, s_1, s_2, \ldots, s_n, k)$ corresponds to a state where a decision has been made on the allocation of item i and all items ranked higher than i, with each bidder j receiving s_j items and k items remaining with the auctioneer. Selecting an arc from $(i, s_1, s_2, \ldots, s_n, k)$ to $(i+1, s'_1, s'_2, \ldots, s'_n, k')$ therefore corresponds to allocating item $i+1$ to that bidder j for which $s'_j = s_j + 1$. If there is no such bidder, then item $i+1$ remains with the auctioneer (and $k' = k+1$). In this way, each path from source to sink determines how the items are to be allocated, and there is a path from the source to the sink for each possible allocation.

We now sketch the equivalence between the length of a path in the graph and the value of an allocation of the items. We know that in a matrix bid, the value of adding an item i to a set is determined only by the number of higher ranked items. Since the graph contains only arcs from higher ranked items to lower ranked items, the effect of adding an item i to a set on the bid for this set can be determined, regardless of whatever items are added to the set further down the path. The cost of an arc is nothing else but the appropriate entry from the matrix bid of the bidder receiving the item. This means that the length of any path from source to sink corresponds to $\sum_j b_j(S_j)$, where S_j is the set of items allocated to bidder j, according to that path. Therefore, the winner determination problem for a matrix bid auction with a fixed number of bidders, all having an identical ordering of the items, can be solved by solving a longest path problem. This can be done in polynomial time, since the underlying graph is acyclic. □

3. Mathematical formulations

In this section, we present two mathematical formulations for the matrix bid auction winner determination problem. The first formulation (see also Day [7]) is inspired by the assignment problem, the second by the set packing problem. We show that the LP-relaxations of both formulations are equally strong.

We define the binary variable x_{ijk} to be 1 if bidder j receives item i as the kth best item, and 0 otherwise. This leads to the formulation below, to which we refer as the assignment formulation:

maximize

$$\sum_{i\in G}\sum_{j\in B}\sum_{k=1}^{r_{ij}} b_{ijk}x_{ijk} \tag{2}$$

subject to

$$\sum_{j\in B}\sum_{k=1}^{r_{ij}} x_{ijk} \leqslant 1 \quad \forall i \in G, \tag{3}$$

$$\sum_{i\in G: r_{ij} \geqslant k} x_{ijk} \leqslant 1 \quad \forall j \in B, \ \forall k \in \{1, \ldots, r_{ij}\}, \tag{4}$$

$$\sum_{l\in G: k \leqslant r_{lj} \leqslant r_{ij}} x_{ljk} \leqslant \sum_{l\in G: k-1 \leqslant r_{lj} < r_{ij}} x_{ljk-1} \quad \forall i \in G, \ \forall j \in B, \ \forall k \in \{2, \ldots, r_{ij}\}, \tag{5}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in G, \ \forall j \in B, \ \forall k \in \{1, \ldots, r_{ij}\}. \tag{6}$$

Constraints (3) enforce that each item can be assigned to at most one bidder, while constraints (4) make sure that for each bidder, at most one item is the kth best item in the set this bidder gets. Finally, constraints (5) impose that a bidder cannot get an item as the kth best item in a set, unless a higher ranked item was assigned to this bidder as his $(k-1)$th best item in this set. Constraints (6) are the integrality constraints.

Notice that the formulation (2)–(6) is not the minimal correct formulation for the matrix bid winner determination problem. Indeed, constraints (4) for $k \in \{2, \ldots, r_{ij}\}$ are redundant in (2)–(6), since they are already enforced by constraints (4) for $k = 1$ and constraints (5). Also, replacing constraints (5) with the following (weaker) constraints still results in a correct formulation:

$$x_{ijk} \leqslant \sum_{l\in G: k-1 \leqslant r_{lj} < r_{ij}} x_{ljk-1} \quad \forall i \in G, \ \forall j \in B, \ \forall k \in \{2, \ldots, r_{ij}\}.$$

However, with this formulation, all constraints (4) remain necessary.

The set packing formulation below makes use of binary variables $y(S, j)$, which equals 1 if bidder j wins set S, and 0 otherwise. The first set of constraints (8) enforces that each item is awarded to at most one bidder. The second set of constraints (9) guarantees that no bidder receives more than one set. The integrality constraints are (10).

maximize

$$\sum_{j\in B}\sum_{S\subseteq G} b_j(S) y(S, j) \tag{7}$$

subject to

$$\sum_{S\supseteq\{i\}}\sum_{j\in B} y(S, j) \leqslant 1 \quad \forall i \in G, \tag{8}$$

$$\sum_{S\subseteq G} y(S, j) \leqslant 1 \quad \forall j \in B, \tag{9}$$

$$y(S, j) \in \{0, 1\} \quad \forall S \subseteq G, \ \forall j \in B. \tag{10}$$

Notice that this set packing formulation can also be used for the winner determination problem of a general combinatorial auction. Indeed, the matrix bid auction only differs from a general combinatorial auction in the way $b_j(S)$ is computed. Notice also that the assignment formulation is polynomially sized in the number of bidders and the number of items. This is not the case for the set packing formulation. In the following theorem, we prove that the LP-relaxation of the set packing formulation and the LP-relaxation of the assignment formulation are equally strong.

Theorem 3. *The LP-relaxation of the assignment formulation and the LP-relaxation of the set packing formulation are equally strong. Moreover, if the assignment formulation has an integral solution that is optimal with respect to the LP-relaxation, this is also the case for the assignment formulation, and vice versa.*

Proof. In order to prove the first part of this theorem, we need to show that the LP-relaxation of the set packing formulation is at least as strong as the LP-relaxation of the assignment formulation and vice versa. In order to prove

the first relation, we need to show that any solution $\hat{y}$ of the LP-relaxation of the set packing formulation can be transformed to a solution $\hat{x}$ of the LP-relaxation of the assignment formulation with the same objective function value. This is accomplished by the following procedure. For the remainder of this proof, if we mention a formulation, we mean in fact its LP-relaxation.

First, we initialize all variables $\hat{x}_{ijk}$ to 0, for all $i \in G$, $j \in B$, and $k \in \{1, \ldots, r_{ij}\}$. We consider each variable $\hat{y}(S, j)$, with $S \subseteq G$ and $j \in B$ once, and set for each item i in S

$$\hat{x}_{i,j,k(i,j,S)} \leftarrow \hat{x}_{i,j,k(i,j,S)} + \hat{y}(S, j). \tag{11}$$

Thus, in this procedure, the value of each variable $\hat{y}(S, j)$ is added to $|S|\hat{x}_{ijk}$ variables, namely those with item $i \in S$, and $k = k(i, j, S)$. It follows that the following equality is valid:

$$\sum_{k=1}^{r_{ij}} \hat{x}_{ijk} = \sum_{k=1}^{r_{ij}} \sum_{S:i\in S\wedge k=k(i,j,S)} \hat{y}(S, j) = \sum_{S\supseteq\{i\}} \hat{y}(S, j) \quad \forall i \in G, \quad j \in B. \tag{12}$$

Using this equality, we verify that (3) holds for $\hat{x}$:

$$\sum_{j\in B} \sum_{k=1}^{r_{ij}} \hat{x}_{ijk} = \sum_{j\in B} \sum_{S\supseteq\{i\}} \hat{y}(S, j) \leqslant 1. \tag{13}$$

Notice that the last inequality follows from the feasibility of $\hat{y}$ (see (8)). We also establish for $j \in B$, and $k \in \{1, \ldots, r_{ij}\}$:

$$\begin{aligned} \sum_{i\in G:r_{ij}\geqslant k} \hat{x}_{ijk} &= \sum_{i\in G} \sum_{S:i\in S\wedge k(i,j,S)=k} \hat{y}(S, j) \\ &= \sum_{S:|S|\geqslant k} \hat{y}(S, j) \\ &\leqslant \sum_{S\subseteq G} \hat{y}(S, j) \leqslant 1, \end{aligned} \tag{14}$$

which shows that $\hat{x}$ satisfies (4). Finally, we have that for each $i \in G$, $j \in B$, and $k = 1, \ldots, r_{ij}$:

$$\sum_{l\in G:k\leqslant r_{lj}\leqslant r_{ij}} \hat{x}_{ljk} = \sum_{l\in G:r_{lj}\leqslant r_{ij}} \sum_{S:l\in S\wedge k(l,j,S)=k} \hat{y}(S, j). \tag{15}$$

Thus we can write for each $i \in G$, $j \in B$, and $k = 2, \ldots, r_{ij}$:

$$\begin{aligned} &\sum_{l\in G:k\leqslant r_{lj}\leqslant r_{ij}} \hat{x}_{ljk} - \sum_{l\in G:k-1\leqslant r_{lj}<r_{ij}} \hat{x}_{ljk-1} \\ &= \sum_{l\in G:r_{lj}\leqslant r_{ij}} \sum_{S:l\in S\wedge k(l,j,S)=k} \hat{y}(S, j) - \sum_{l\in G:r_{lj}\leqslant r_{ij}} \sum_{S:l\in S\wedge k(l,j,S)=k-1} \hat{y}(S, j). \end{aligned} \tag{16}$$

Consider some $\hat{y}(S, j)$ occurring in the first term. The corresponding set S has at the kth position ($k \geqslant 2$) some item l, $r_{lj} \leqslant r_{ij}$. It follows that there must be some other item, say l' with $r_{l'j} \leqslant r_{lj}$ at position $k - 1$. Hence, this $\hat{y}(S, j)$ also occurs in the second term. It follows that the expression (16) cannot have a positive value, and hence (5) is satisfied. Notice also that the transformation procedure (11) does not affect the objective function value. Moreover, it transforms any integral solution $\hat{y}$ to an integral solution $\hat{x}$.

Hence, we have shown that the set packing formulation is at least as strong as the assignment formulation and if the set packing formulation has an integral solution that is optimal with respect to the LP-relaxation, this is also the case for the assignment formulation. In the remainder of this proof, we show that the assignment formulation is at least strong as the set packing formulation. In order to prove this second relation, we show that any solution $\hat{x}$ of the LP-relaxation

of the assignment formulation can be transformed to a solution $\hat{y}$ of the LP-relaxation of the set packing formulation with the same objective function value. This is accomplished by the following procedure, CONVERT($\hat{x}$).

Algorithm 1 CONVERT($\hat{x}$)

for ($j \in B$) **do**
  Initialize $\hat{y}(S, j) \leftarrow 0$ for all $S \subseteq G$;
  Step 1:
  for ($i \in G$) **do**
    $\hat{y}(\{i\}, j) \leftarrow \hat{x}_{ij1}$
  end for
  Step 2:
  for ($k = 2$ to m) **do**
    for ($i \in G$: $r_{ij} = k$ to m) **do**
      Step 2a:
      $T = \{S \subseteq \{i' : r_{i'j} < r_{ij}\} : |S| = k - 1\}$;
      while ($\hat{x}_{ijk} > 0$) **do**
        Pick a set S from T and remove S from T;
        if ($\hat{x}_{ijk} > \hat{y}(S, j)$) **then**
          $\hat{y}(S \cup \{i\}, j) \leftarrow \hat{y}(S, j)$;
          $\hat{x}_{ijk} \leftarrow \hat{x}_{ijk} - \hat{y}(S, j)$;
          $\hat{y}(S, j) \leftarrow 0$;
        else
          $\hat{y}(S \cup \{i\}, j) \leftarrow \hat{x}_{ijk}$;
          $\hat{y}(S, j) \leftarrow \hat{y}(S, j) - \hat{x}_{ijk}$;
          $\hat{x}_{ijk} \leftarrow 0$;
        end if
      end while
    end for
  end for
end for

The CONVERT procedure translates any solution for the assignment formulation to a solution for the set packing formulation. First, we argue that the CONVERT algorithm terminates.

The crucial step in the CONVERT algorithm is step 2a, which has to be performed for each bidder j, for each k from 2 to m, and for each $i \in G$ with $r_{ij} \geqslant k$. Let us consider now a bidder j, item i, and rank k, for which step 2a is to be performed, and let $\tilde{y}(S, j)$ be the solution as it is constructed by the CONVERT algorithm so far. In order to guarantee that the while loop in step 2a terminates, we need

$$\hat{x}_{ijk} \leqslant \sum_{S: S \subseteq \{i': r_{i'j} < r_{ij}\} \wedge |S| = k-1} \tilde{y}(S, j). \tag{17}$$

Notice that in CONVERT, so far, each variable $\tilde{y}(S, j)$, with $|S| = k - 1$ and l being the lowest ranked item in S, has been increased at most once, namely with (a fraction of) $\hat{x}_{l,j,k-1}$. Furthermore, the total value of $\hat{x}_{l,j,k-1}$ has been added exclusively over variables $\tilde{y}(S, j)$ with $|S| = k - 1$ and l the lowest ranked item in S. Therefore, we have that the total fraction that has been added to variables $\tilde{y}(S, j)$ with S containing $k - 1$ items ranked higher than i equals

$$\sum_{i': k \leqslant r_{i'j} < r_{ij}} \hat{x}_{i',j,k-1}. \tag{18}$$

Notice that the value of each variable $\tilde{y}(S, j)$ may also have been decreased in CONVERT. Indeed, variables $\tilde{y}(S, j)$ with S containing $k - 1$ items and the one with the lowest rank being l can be decreased only with (a fraction of) variables $\hat{x}_{i',j,k}$ with i' ranked higher than l, and lower than i (since step 2a has not yet been performed for rank k and item i or items ranked lower than i).

Furthermore, the total value of $\hat{x}_{l,j,k}$ has been subtracted only from variables $\widetilde{y}(S, j)$, with S containing $k-1$ items, all ranked higher than l. Therefore, we have that the total fraction that has been subtracted from variables $\widetilde{y}(S, j)$ with S containing $k-1$ items ranked higher than i equals

$$\sum_{i':k \leqslant r_{i'j} < r_{ij}} \hat{x}_{i',j,k}. \tag{19}$$

Thus,

$$\sum_{S:S \subseteq \{i':r_{i'j} < r_{ij}\} \wedge |S|=k-1} \widetilde{y}(S, j) = \sum_{i':k \leqslant r_{i'j} < r_{ij}} \hat{x}_{i',j,k-1} - \sum_{i':k \leqslant r_{i'j} < r_{ij}} \hat{x}_{i',j,k}. \tag{20}$$

Further, it follows from (5) that

$$\hat{x}_{ijk} \leqslant \sum_{i':k \leqslant r_{i'j} < r_{ij}} \hat{x}_{i',j,k-1} - \sum_{i':k \leqslant r_{i'j} < r_{ij}} \hat{x}_{i',j,k} \tag{21}$$

for each bidder j, for each k from 2 to m, and for each $i \in G$ with $r_{ij} \geqslant k$. From (20) and (21) we conclude that (17) is true and hence the CONVERT algorithm terminates.

We now argue that solution $\hat{y}$ is indeed feasible with respect to constraints (8), (9), and the relaxation of (10).

For each bidder j and each item i, it is clear that after step 1, $\sum_{S \supseteq \{i\}} \hat{y}(S, j) = \hat{x}_{ij1}$. In step 2, each value $\hat{x}_{ijk}$ is spread over one or more variables $\hat{y}(S, j)$ with S containing item i. Also, for each variable $\hat{y}(S, j)$ that is increased, a variable $\hat{y}(S \backslash \{i\}, j)$ is decreased with the same value. Therefore, after step 2, $\sum_{S \supseteq \{i\}} \hat{y}(S, j) \leqslant \sum_{k=1}^{r_{ij}} \hat{x}_{ijk}$. Summing over the bidders gives

$$\sum_{j \in B} \sum_{S \supseteq \{i\}} \hat{y}(S, j) \leqslant \sum_{j \in B} \sum_{k=1}^{r_{ij}} \hat{x}_{ijk}.$$

Given (3), this implies that constraints (8) are satisfied.

For each bidder j, it is clear that after step 1 of CONVERT, $\sum_{S \subseteq G} \hat{y}(S, j) = \sum_{i \in G} \hat{x}_{ij1}$. In step 2, for every variable $\hat{y}(S, j)$ whose value is increased, there is some other variable $\hat{y}(S', j)$ whose value is reduced by the same amount. Given (4), this implies that constraints (9) are satisfied.

Each variable $\hat{y}(S, j)$ is increased by at most one variable $\hat{x}_{ijk}$. Therefore, it follows from the relaxation of constraints (6) that $\hat{y}(S, j) \leqslant 1$ for all $S \subseteq G$ and each bidder j. By construction of the algorithm, no variable $\hat{y}(S, j)$ will have a value less than zero. Thus, $\hat{y}$ satisfies the relaxation of constraints (10).

Further, the objective function value of both solutions $\hat{x}$ and $\hat{y}$ is the same. Consider any bidder j. After step 1, the objective function of solution $\hat{y}$ has a value equal to $\sum_{i \in G} b_{ij1} \hat{x}_{ij1}$, since $b(\{i\}, j) = b_{ij1}$. Every time step 2a is performed, the objective function value is increased by $(b(S \cup \{i\}, j) - b(S, j))\hat{x}_{ijk}$. Since set S contains only items ranked higher than item i, we have $b(S \cup \{i\}, j) - b(S, j) = b_{ijk}$, where k is the number of items in S plus one. Therefore, after step 2 the objective function equals $\sum_{i \in G} \sum_{k=1}^{r_{ij}} b_{ijk} \hat{x}_{ijk}$. Summing over all bidders j shows that the CONVERT($\hat{x}$) procedure produces a solution $\hat{y}$ with the same objective function value as $\hat{x}$.

Finally, it is easy to see that if the CONVERT procedure is confronted with an integral solution $\hat{x}$, it will produce an integral solution $\hat{y}$. Thus, we can conclude that the assignment formulation and the set packing formulation are equally strong, and that if one formulation has an integral optimal solution, this is also the case for the other formulation. □

4. Branch-and-price algorithms for solving the matrix bid auction

Theorem 3 shows that the set packing formulation (7)–(10) is equally strong as the assignment formulation (2)–(6). Here we outline an algorithm based on the set packing formulation. Solving the LP-relaxation of the set packing formulation is, however, not trivial, given the huge amount of variables ($n2^m$). Considering that only a small percentage of these variables are non-zero in an optimal solution, column generation suggests itself as an efficient solution technique. Column generation was proposed by Dantzig and Wolfe [16] and starts by solving the LP-relaxation considering only

a subset of the variables. This problem is also called the restricted master problem. Notice that this problem can be restricted to $m + n$ variables, whereas the assignment formulation requires $nm(m + 1)/2$ variables, which may still be large. The next step is to verify whether any of the variables that were not considered could improve the current solution. In other words, we search for a variable with a non-negative reduced cost. This problem is called the pricing problem. If we find such a variable, we add it to the restricted master problem and solve it again. This re-optimizing and pricing is to be repeated until the pricing problem fails to produce new variables, indicating that the LP-relaxation has been solved to optimality.

Notice that the column generation procedure does not guarantee to find an integral solution. In case of a fractional solution, a branching decision needs to be made, partitioning the solution space in order to create a number of smaller subproblems. With branch-and-price, this results in a search tree where column generation has to be applied in every node (we refer to Vanderbeck and Wolsey [17] for a more elaborate description of the branch-and-price technique). The key to an efficient branch-and-price algorithm is an easy-to-solve pricing problem. The branching rule should therefore not destroy the structure of the pricing problem or increase its complexity when moving deeper down the search tree.

4.1. Column generation for the matrix bid auction

In this section, we show how the LP-relaxation of the set packing formulation of the matrix bid winner determination problem can be solved using column generation. We also prove that the pricing problem can be solved in polynomial time, since it can be solved by solving a shortest path problem.

If we define u_i for each item $i \in G$ as the dual price associated with the corresponding constraint of (8), and v_j for each bidder $j \in B$ as the dual price associated with the corresponding constraint of (9), we can write the dual of the set packing formulation (7)–(10) as follows:

minimize

$$\sum_{i \in G} u_i + \sum_{j \in B} v_j \tag{22}$$

subject to

$$\sum_{i \in S} u_i + v_j \geqslant b_j(S) \quad \forall S \subseteq G, \ \forall j \in B, \tag{23}$$

$$u_i \geqslant 0, \ \ v_j \geqslant 0 \quad \forall i \in G, \ \forall j \in B. \tag{24}$$

We start by finding an optimal solution for the restricted master problem, i.e. the LP-relaxation of (7)–(10) considering only a limited number of variables $y(S, j)$. This solution is also an optimal solution for the (unrestricted) LP-relaxation of (7)–(10) if its corresponding dual variables form a feasible solution for (22)–(24), which has a constraint for every variable $y(S, j)$. Consequently, we need to add a new column or variable to the restricted master problem if a constraint of (23) is violated. The pricing problem thus boils down to determining the existence of a set S of items and a bidder j such that

$$\sum_{i \in S} u_i < b_j(S) - v_j. \tag{25}$$

Theorem 4. *The pricing problem, i.e. finding a set S of items and a bidder j such that a constraint of* (23) *is violated, can be solved by solving a shortest path problem.*

Proof. We construct a graph with a source and a sink, and a subgraph for each bidder j. Such a subgraph contains r_{ij} nodes for each item i, called item nodes. We will refer to an item node as (i, j, k), where i stands for the item and k ranges from 1 to r_{ij}. There are arcs from each node (i, j, k) to each node $(i', j, k + 1)$ where item i' is ranked lower than item i (i.e. $r_{i'j} > r_{ij}$). These arcs have a cost equal to $u_{i'} - b_{i',j,k+1}$. Notice that there are no arcs between nodes corresponding to different subgraphs. Furthermore, for each subgraph, there are arcs from the source node to node $(i, j, 1)$ for each item i with a cost equal to $u_i - b_{ij1}$ and there are arcs from each item node (i, j, k) to the sink with

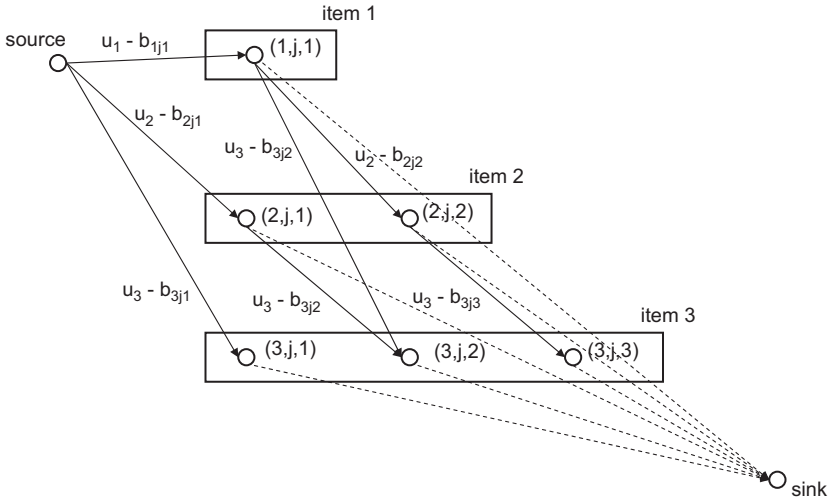

Fig. 2. The pricing problem as a shortest path problem.

cost v_j. A schematic representation of this graph is given in Fig. 2 for a setting with a single bidder j and three items. In order not to overload this figure, we did not put labels on the dashed arcs, which all have a cost v_j.

From the structure of this graph, it follows that all nodes of a path from the source to the sink correspond to the same bidder and each path contains at most one node per item. Moreover, exactly one arc with cost v_j is included in the path. Therefore, the length of a path containing nodes (i, j, k) of the items $i \in S$ of bidder j in this graph equals

$$\sum_{i \in S} (u_i - b_{ijk}) + v_j. \tag{26}$$

Furthermore, the graph ensures that an item i is in the path using its kth node only if a higher ranked item is in the path through its $(k-1)$th node. We can therefore state that $\sum_{i \in S} b_{ijk} = b_j(S)$ and it follows that the existence of a path with negative length corresponds to a violated constraint in the dual. Consequently, we need to solve a shortest path problem on a directed acyclic graph in order to solve the pricing problem. □

Thus, if the shortest path has a negative length, we can add a column for the corresponding bidder j containing the items in set S determined by the item nodes traversed in the path. Naturally, bidder j's bid for this set S is $b_j(S)$. Notice that since the pricing problem is solvable in polynomial time, the LP-relaxation of the set packing formulation for the matrix bid auction can also be solved in polynomial time.

Corollary 1. *The LP-relaxation of the set packing formulation* (7)–(10) *for the matrix bid auction winner determination problem can be solved in polynomial time.*

4.2. Branching on an item–bidder pair

The solution of the LP-relaxation of the matrix bid winner determination problem found by column generation may not be integral. If this is the case, we need to partition the solution space to eliminate this fractional solution. In this approach, we partition the solution space by the branching decision whether or not to assign an item to a bidder. We first prove that in a fractional solution, there always exists an item that has been fractionally assigned to one or more bidders.

Lemma 1. *For any fractional solution to the relaxation of* (7)–(10),

$$\exists i \in G, \quad j \in B : 0 < \sum_{S : S \supseteq \{i\}} y(S, j) < 1. \tag{27}$$

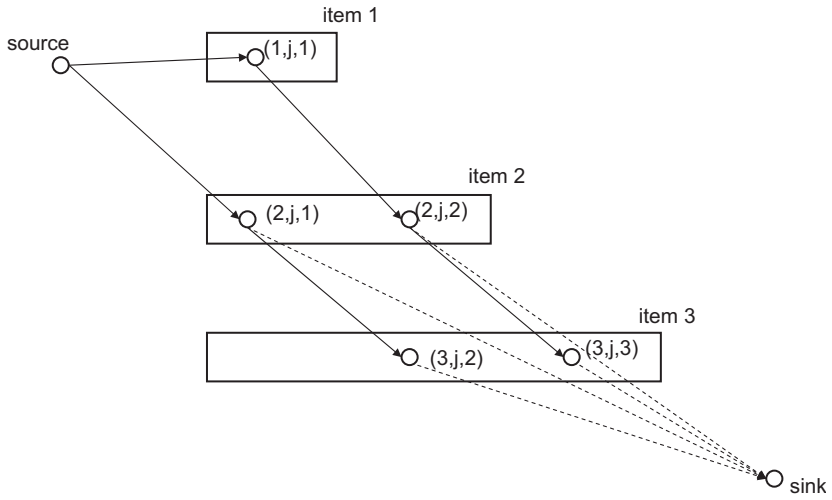

Fig. 3. The pricing problem where the bidder must get item 2.

Proof. We will prove this theorem by showing that a solution must be integral if it does not satisfy (27). Consider a solution for which property (27) is not valid. This means that each item has been assigned fully or not at all to each bidder. In this case, no items are split over multiple bidders. An item p for which $\sum_{S:S\supseteq\{p\}} y(S, j) = 1$ could, however, still be split over multiple sets of the same bidder j. It is easy to see that if bidder j is awarded a set S containing next to p any other item q, that this item then should occur in each set containing p in order to have the sum of the fractions of sets containing p equal 1. In other words, the sets of bidder j are identical, and we have, in fact, an integral solution. □

The branch-and-price algorithm can, however, only be valid if in every node of the search tree, all generated columns satisfy the previously made branching decisions. Prohibiting that an item is awarded to a certain bidder in the pricing problem can be done by simply removing the vertices corresponding to that item for that bidder from the graph. Enforcing that an item is awarded to a certain bidder in the pricing problem is less obvious. For that bidder, the arcs from the source to any lower ranked item need to be removed. Also the arcs from any higher ranked item to any item ranked lower than that item need to be deleted. Finally, the arcs from the higher ranked items to the sink must be removed as well. Clearly, all nodes that can no longer be reached as a consequence of these removals can now also be deleted, as are the arcs leaving those nodes, and so on. For all other bidders, we need to remove the vertices of that item from the graph. Fig. 3 shows the pricing problem where item 2 is forced to be awarded to the bidder whose item nodes are depicted. In this graph, we made sure that every path from the source to the sink of that bidder must include a node corresponding to item 2.

Notice that this branching rule does not destroy the structure of the pricing problem: in all branches, the pricing problem remains a shortest path problem. It is easy to see that this shortest path problem can be adjusted to produce columns that comply with a series of branching decisions. Moreover, when moving deeper down the tree, more and more arcs and nodes will be removed. Thus, we have described a valid branching rule where the pricing problem remains solvable as a shortest path problem throughout the search tree.

4.3. Branching on a pair of successive items

Ryan and Foster [18] suggest a branching rule for the set partitioning problems where two constraints are covered together or not at all by the variables in one branch, whereas in the other branch, each variable can cover at most one of these constraints. This rule can easily be generalized to set packing problems and can be translated to a combinatorial auction context as two items needing to go to the same bidder in one branch and to different bidders in the other branch. However, forcing two arbitrary items to go to the same bidder, but also forbidding that these items go to the same bidder, is not straightforward to achieve in the shortest path problem described in Section 4.1. Therefore, we modify this branching rule, such that it takes into account the ranking of the items specified in the bidder's matrix bid.

We partition the solution space by branching on a pair of items p and q. In one branch, we enforce that if item p is present in a bidder's set, then item q must be directly successive to p in this set, when the set is sorted according to this bidder's ranking of the items. In the other branch, no bidder can have items p and q as direct successors in a set, according to his ranking. We first prove that there always exists a pair of items such that the sets in which these items occur as direct successors according to the corresponding bidder's ranking have been fractionally assigned to one or more bidders. We introduce the notation $p \rightarrow_j q$ to denote that item p is directly succeeded by item q in a set, according to the ranking of bidder j.

Lemma 2. *For any optimal, extreme fractional solution to the relaxation of* (7)–(10),

$$\exists p, q \in G : 0 < \sum_{j \in B} \sum_{S: S \supseteq \{p,q\} \wedge p \rightarrow_j q} y(S, j) < 1. \tag{28}$$

Proof. Assume that we have an optimal, extreme fractional solution for which (28) is not satisfied. This means that for each pair of items, each bid on a set in which these items are direct successors according to ranking of the bidder that made the bid, has been assigned to that bidder for a total fraction of 0 or 1. Thus, for any items p and q for which $\sum_{j \in B} \sum_{S: S \supseteq \{p,q\} \wedge p \rightarrow_j q} y(S, j) = 1$, we can conclude that if item p is present in a set, that then also item q is present in this set. Therefore, each pair of sets to which a positive fraction has been assigned is disjoint or identical. Since there is a single variable $y(S, j)$ representing identical sets of the same bidder j, we conclude that identical sets must be split over multiple bidders. This leaves us with the problem of assigning a number of disjoint sets among one or more bidders, where each assignment of a set to a bidder has its profit, namely the bid of this bidder for this set. This problem is a maximum weighted assignment problem on a bipartite graph, where each node on one side of the partition represents a set, and each node on the other side of the partition represents a bidder. It follows that each optimal, extreme solution is integral. Consequently, for any optimal, extreme fractional solution to the relaxation of (7)–(10), property (28) is true. □

The above lemma shows that it is always possible to find a pair of items p and q on which to branch. However, we still need to enforce that the pricing problem will generate columns that satisfy the constraint imposed by the branching decision. In the branch where we impose $p \rightarrow_j q$, for each bidder j, we need to remove all arcs from nodes corresponding to p to any node not corresponding to q. Notice that for a bidder that ranks q higher than p, this comes down to removing all nodes related to p from the graph. This leaves us with a graph where if one arrives in a node related to p, the only option is to take an arc to a node related to q. In the branch where p should not be directly succeeded by q, it suffices, for each bidder, to remove the arcs going from a p-node to a q-node, if they exist.

Notice that this branching rule does not destroy the structure of the pricing problem either, even when we consider a sequence of branching decisions. Indeed, it is not hard to verify that when going deeper into the search tree, the pricing problem can still be solved as a shortest path problem on an increasingly smaller graph.

4.4. Implementation issues

Both branch-and-price algorithms were implemented using Visual C++ 6.0. The set packing problems were solved using Ilog Cplex 8.1. The LEDA libraries (version 5.0.1) allowed us to solve the shortest path problems in linear time, since the underlying graph is directed and acyclic. In the remainder of this section, some of the most important implementation issues are discussed.

4.4.1. Solving the root node

A first issue that needs to be solved is determining which columns will be used in the very first restricted master problem. Using many columns obviously increases the computation time needed to solve the restricted master problem. On the other hand, this may result in a solution that is closer to the optimal solution, such that less iterations for solving the pricing problem and re-optimizing are needed. In our case, after experimenting with a number of settings, it turned out that including a rather large number of variables to start the column generation process pays off. We constructed a set for every strictly positive entry in the matrix bid by taking the item corresponding to this entry and completing the set with the k highest ranked items, where k is the entry's column in the matrix bid.

After the restricted master problem has been solved and the corresponding dual solution has been obtained, new columns with a non-negative reduced cost need to be added. The question remains how many such columns we should add. Again, adding too many new variables increases the computation time for solving the resulting restricted master problem, whereas adding too few variables can result in a large number of iterations for solving the pricing problem and re-optimizing. The strategy that proved to be the most efficient consists of adding for each bidder those variables whose reduced cost is at most 2% less than the most positive reduced cost for a variable from that bidder. Furthermore, the number of such variables that is added for each bidder cannot exceed the number of items. Notice that finding these variables demands very little extra computation time, since the LEDA libraries provide the distance from the source to each node in the graph, after having solved the shortest path problem.

Finally, when re-optimizing the restricted master problem, we start from the optimal base of the previous iteration. In order not to drag along too many columns for the remainder of the search tree, those columns that were added at some iteration, but never made part of any base solution are removed from the model. We keep the other columns, assuming that they will be useful again later.

4.4.2. A selection rule when branching on an item–bidder pair

The major issue in implementing this branching rule is to choose the item on which to branch and the bidder(s) to assign it to. We chose to branch on the item that is fractionally assigned to the highest number of bidders. For each of these bidders, a branch is constructed in which the bidder is assigned the item. A final branch is added where none of these bidders is allowed to receive the item. We opted for a depth-first strategy, where the branch where the item is assigned to the bidder with the highest fraction is explored first. Thus, the branch where bidders are disallowed to receive an item always comes last.

4.4.3. A selection rule when branching on a pair of successive items

With this branching rule, each node that needs further partitioning of the solution space leads to two branches. In the first branch, we enforce that for each bidder, if item p is present in a bid, q should be the next item in that bid, according to the ranking of that bidder. The second branch considers only bids for which p and q are no direct successors according to the bidder's ranking. We again chose a depth-first strategy, where the branch where $p \rightarrow_j q$ is imposed is explored first. The question remains how to select the items p and q. We opted to pick those items p and q for which $\sum_{j \in B} \sum_{S: S \supseteq \{p,q\} \wedge p \rightarrow_j q} y(S, j)$ is closest to 0.5.

4.4.4. Solving a tree node

Before we can start solving a node of the tree, we remove all columns that do not satisfy the latest branching decision. In case of backtracking, this branching decision expires and those columns are re-entered into the model, since we experienced that they often turn out to be useful in other branches of the tree.

The LP objective value of the node can be used as an upper bound to the integral solution that could be found further down the tree. Clearly, if this value is lower than the incumbent found so far, the node can be pruned. It may, however, require a large number of iterations to prove LP optimality. Vanderbeck and Wolsey [17] show that the Lagrangian relaxation can also be used as an upper bound. The Lagrangian upper bound can be computed as

$$\delta + \sum_{j \in B} \max_{S \subseteq G} (RC(S, j), 0), \tag{29}$$

where δ is the objective value of the restricted master and $RC(S, j)$ is the reduced cost of variable $y(S, j)$. Notice that the computation of this bound requires little additional computational effort, since the pricing problem, which is solved for every bidder j anyway, finds the variable with the highest reduced cost. This upper bound is referred to as the Lagrangian upper bound, since it equals the bound obtained by Lagrange relaxation (Lasdon [19]). If at any iteration in the column generation process, the Lagrangian upper bound is lower than the incumbent, we can prune the node, without any risk of missing the optimal solution.

Obviously, when we re-optimize the restricted master problem, we also start from the optimal base of the previous iteration. The first restricted master problem is solved starting from the base solution of the parent node. Furthermore, as in the root node, we delete the added columns that turned out not to be useful.

5. Computational results

In this section, we test the performance of the branch-and-price algorithms on randomly generated instances. After elaborating on the structure of these instances, we give an overview of the computational results and compare them with results from a branch-and-cut approach performed on the assignment formulation.

5.1. Structure of the instances

Unfortunately, real-life data for combinatorial auctions are not abundantly available for the public. It is therefore not uncommon in combinatorial auction literature to turn to randomly generated data (see for instance Leyton-Brown et al. [20], Parkes [21], and Sandholm [3]). For a thorough discussion on the empirical hardness of several data distributions commonly used for combinatorial auctions, we refer to Leyton-Brown et al. [22].

For our instances, each matrix bid is composed according to a bid type, randomly chosen out of the following six possibilities used by Day [7], and a bid type that has non-increasing rows and columns.

Additive preference bidder: For every item i, the bidder has a price p_i. The bidder's valuation for a set S is then $\sum_{i\in S} p_i$.

Single-minded bidder: This bidder is interested in one particular set S for which he is willing to pay a price p. These single minded bids (S, p) are also known as flat bids or atomic bids (Nisan [5]).

Nested flat bidder: This bidder is a generalization of the single-minded bidder and expresses a chain of q exclusive single-minded bids $(S_1, p_1), (S_2, p_2), \ldots, (S_q, p_q)$ such that $S_1 \subset S_2 \subset \cdots \subset S_q$.

Nested k-of bidder: The k-of bid function consists of a bid (k, S, p), which is a bid of p on any subset of S of at least size k. Multiple k-of bids $(k_1, S, p_1), (k_2, S, p_2), \ldots, (k_q, S, p_q)$ on the same set S can be represented in a single matrix bid, provided that all k-values are different. This bid function is also known as the general symmetric bid function, in which the bidder specifies prices $p_1, p_2, \ldots, p_m$ where p_k is the price the bidder is willing to pay for the kth item won (Nisan [5]). The bidder's valuation for a set S is then $\sum_{i=1}^{|S|} p_i$.

Partition bidder: This bidder partitions the items into a number of groups of substitutes. The bidder gives a ranking of the groups and prices he is willing to pay for receiving exactly one item from each group, given that exactly one item from each higher ranked group has been received. This bid function can be generalized to accommodate an arbitrary given demand for each group of substitutes.

Add-on bidder: This bid function consists of a bid for an essential item, and extra prices the bidder is willing to pay for each number of items from a set of add-on items in which the bidder is interested.

In order to avoid auctions for which the exact solution of the winner determination problem is obvious, the matrix bids are constructed such that they are competitive. For more details on the bid types or on how the instances were generated, we refer to Day [7]. We performed experiments on matrix bid auctions with 5, 10, 25 or 50 items and 5, 10, 25, 50, 75 or 100 bidders. For each combination, 10 instances were generated and solved to optimality. All computational experiments were done on a desktop computer with a Pentium IV 2 GHz processor, with 512 MB RAM.

5.2. Results

Tables 1 and 2 give an overview of the average computation times needed to solve the matrix bid auction winner determination problem using branch-and-price with branching on an item–bidder pair (BOI) and branch-and-price with branching on a pair of successive of items (BOS), respectively. In Table 3, we give the average computation times

Table 1
Average computation times (s) for n bidders and m items using BOI

	$n=5$	10	25	50	75	100
$m=5$	*0.005*	0.007	0.008	*0.017*	*0.027*	*0.038*
10	*0.027*	0.038	0.053	0.088	0.118	*0.169*
25	*0.636*	*0.597*	*1.157*	4.292	*12.704*	49.155
50	247.224	60.711	437.951	557.083	622.591	*802.483*

Table 2
Average computation times (s) for n bidders and m items using BOS

	$n=5$	10	25	50	75	100
$m=5$	*0.005*	*0.006*	*0.006*	0.018	*0.027*	*0.038*
10	0.033	*0.037*	*0.044*	*0.067*	*0.104*	0.182
25	0.698	0.767	1.194	*3.814*	16.300	97.122
50	76.598	67.584	843.435	259.079	645.632	983.539

Table 3
Average computation times (s) for n bidders and m items using B&C

	$n=5$	10	25	50	75	100
$m=5$	0.030	0.027	0.049	0.052	0.070	0.102
10	0.050	0.069	0.140	0.278	0.524	0.748
25	0.757	1.391	3.598	10.689	17.584	*31.940*
50	*57.676*	*28.333*	*91.230*	*215.083*	*355.785*	811.960

that resulted from solving the assignment based formulation (2)–(6) with the Ilog Cplex 8.1 branch-and-cut algorithm with standard settings (B&C), which is basically the approach followed in Day and Raghavan [13]. Horizontally, the number of bidders n varies from 5 to 100, while the number of items m auctioned ranges from 5 to 50 vertically. All computation times are expressed in seconds, and the lowest average computation times over the three approaches are in italics.

As could be expected, the computation time is determined more by the number of items in the auction than by the number of bidders. All instances with up to 10 items are solved in less than a second by all algorithms; here the branch-and-price algorithms clearly perform better. Auctions with 50 items are also solved in less than 20 min on average by all algorithms. The branch-and-cut algorithm seems on average the fastest way to solve these instances. Perhaps surprisingly, the computation times for some instances do not always increase when more bidders come into play. For instance, the average computation time is lower for 10 bidders than for 5 bidders for all algorithms for instances with 50 items. This can be explained by the fact that the computation times for the individual instances tend to vary considerably.

One way to get a more accurate view on what the underlying trend is to consider a larger sample set. Also, it is not uncommon in literature on combinatorial auctions to study the median instead (see for instance Sandholm et al. [23] and Hoos and Boutilier [24]). Tables 4–6 give an overview of the median computation times needed to solve the winner determination problem. Again the lowest median computation times over the three approaches are in italics. The tables show a clear trend of how the computation times rise with the number of bidders and the number of items, since the median is less affected by extreme values. Moreover, apart from two exceptions, the branch-and-price algorithms now perform better than the branch-and-cut algorithm. Clearly, the branch-and-price algorithms manage to solve the majority of the instances with many items a lot faster than reflected by the average computation times. The branch-and-cut algorithm seems to suffer less from instances with extreme computation times, since the median computation time is much closer to the average computation time. Thus, the running times of the branch-and-price algorithms display more variance. This may be caused by the presence of bidders of different types in the instance: we come back to this issue later in this section. The results also show that computation times for the branch-and-price algorithm with branching on an item–bidder pair rise more severely with an increasing number of items than those of the branch-and-cut algorithm.

Tables 7 and 8 give the average computation times for solving the LP-relaxation of the set packing formulation (7)–(10) and the assignment formulation (2)–(6), respectively. Recall that the former is used in both branch-and-price algorithms, while the latter is used in the branch-and-cut algorithm, and that the LP-relaxations of both formulations can be solved in polynomial time. Apart from two exceptions, the LP-relaxation of the set packing formulation is solved faster than the LP-relaxation of the assignment formulation. As mentioned in Section 4, this can be explained by the smaller number of variables that need to be dealt with in the column generation approach on the set packing formulation, compared to the assignment formulation. Between brackets, the number of instances out of 10 for which

Table 4
Median computation times (s) for n bidders and m items using BOI

	$n=5$	10	25	50	75	100
$m=5$	*0.000*	*0.010*	*0.010*	*0.020*	*0.030*	*0.040*
10	*0.015*	*0.020*	0.055	*0.055*	0.105	*0.130*
25	*0.480*	*0.460*	*0.760*	*2.445*	9.480	16.825
50	20.855	29.105	45.605	*129.870*	*227.370*	*353.970*

Table 5
Median computation times (s) for n bidders and m items using BOS

	$n=5$	10	25	50	75	100
$m=5$	*0.000*	*0.010*	*0.010*	*0.020*	*0.030*	*0.040*
10	*0.015*	*0.020*	*0.040*	*0.055*	*0.100*	*0.130*
25	0.485	0.495	0.815	*2.445*	*6.790*	*13.605*
50	20.855	29.215	*37.970*	*129.870*	238.785	514.370

Table 6
Median computation times (s) for n bidders and m items using B&C

	$n=5$	10	25	50	75	100
$m=5$	0.020	0.025	0.040	0.050	0.070	0.105
10	0.040	0.060	0.140	0.260	0.535	0.740
25	0.530	1.235	3.245	10.595	18.120	28.960
50	*14.665*	*22.615*	73.035	191.670	350.340	589.940

Table 7
Average computation times (s) for the LP-relaxation of the set packing formulation for n bidders and m items

	$n=5$	10	25	50	75	100
$m=5$	*0.01* [8]	*0.01* [8]	*0.01* [7]	*0.02* [9]	*0.03* [10]	*0.04* [10]
10	*0.01* [6]	*0.02* [9]	*0.03* [4]	*0.05* [8]	*0.09* [8]	*0.14* [7]
25	0.31 [3]	*0.41* [7]	*0.80* [8]	*2.12* [8]	*3.29* [5]	*4.71* [4]
50	*14.02* [5]	*14.96* [6]	*32.93* [4]	*74.46* [7]	*124.29* [5]	*116.19* [4]

Table 8
Average computation times (s) for the LP-relaxation of the assignment formulation for n bidders and m items

	$n=5$	10	25	50	75	100
$m=5$	*0.01* [8]	0.02 [9]	0.02 [8]	0.03 [8]	0.04 [10]	*0.04* [9]
10	0.02 [6]	0.03 [9]	0.05 [5]	0.09 [7]	0.14 [7]	0.20 [8]
25	*0.28* [5]	0.51 [8]	1.61 [7]	4.47 [9]	6.03 [5]	8.35 [5]
50	9.40 [5]	20.01 [6]	43.93 [3]	159.46 [6]	313.76 [4]	461.91 [4]

the LP-relaxation resulted in an integral solution is indicated. Notice that Theorem 3 does not imply that these numbers should be at least as high for the assignment formulation than for the set packing formulation. Indeed, if there exists an integral optimal solution, the algorithms may not find it as there may be fractional solutions with the same objective value. Further, the number of instances for which an integral optimal solution was found remains more or less constant over the bidders, while it drops for instances with more items. Not surprisingly, instances with an integral LP-relaxation

Table 9
Performance details for the three algorithms (BOI, BOS, B&C)

Inst.	BOI			BOS			B&C
	A	B	C	A	B	C	A
5–5	2.2	3.9	33.3	2.4	4.7	34.4	1.0
5–10	1.3	3.7	68.4	1.4	3.4	68.3	1.0
5–25	2.5	4.5	142.2	1.6	3.2	141.6	1.0
5–50	1.3	3.2	266.9	1.2	3.1	266.9	1.0
5–75	1.0	2.4	435.9	1.0	2.4	435.9	1.0
5–100	1.0	2.0	565.0	1.0	2.0	565.0	1.0
10–5	7.6	18.3	124.2	9.4	27.5	93.7	1.2
10–10	7.6	16.5	201.8	5.4	16.8	193.8	1.5
10–25	4.9	10.2	363.8	2.8	8.4	352.0	1.0
10–50	3.7	8.0	788.9	1.8	5.4	782.2	1.0
10–75	2.3	6.9	1117.7	1.6	6.0	1113.8	1.0
10–100	2.3	7.0	1459.5	2.6	8.4	1455.4	1.0
25–5	7.7	72.3	1017.7	6.4	89.4	723.1	1.2
25–10	2.0	37.8	990.7	6.8	51.7	864.5	1.5
25–25	4.4	31.2	1793.3	3.8	33.2	1752.1	1.0
25–50	8.6	59.7	3703.2	5.2	55.1	3602.4	1.0
25–75	30.0	123.5	5402.7	32.8	143.1	5412.7	1.0
25–100	96.8	349.3	7564.0	163.0	635.3	7895.2	1.3
50–5	21.4	3095.8	4745.9	37.9	1279.2	2872.6	11.8
50–10	12.2	592.6	3963.3	27.9	611.7	4112.8	1.0
50–25	315.1	1494.0	11,752.4	1029.6	2806.7	10,141.0	1.2
50–50	361.6	938.5	16,278.1	67.6	468.4	14,359.3	1.0
50–75	102.5	828.5	20,773.4	106.7	852.0	20,776.0	1.0
50–100	96.0	839.5	30,538.0	100.4	995.4	31,120.3	5.7

have low computation times, especially for the set packing formulation. This explains the dominance of the branch-and-price algorithms over the branch-and-cut algorithms for instances with up to 25 items, and it also partially explains the fluctuations in average computation times for the instances with 25 and 50 items (see Tables 1 and 2).

Table 9 gives an overview of the performance details of the three algorithms. Column A gives the average number of nodes in the branching tree that was explored. Column B represents the average number of pricing rounds, and column C gives the average number of variables that was generated (these columns are not applicable for the branch-and-cut algorithm). On the rows, we find the instances, where the first number indicates the number of items and the second gives the number of bidders. There seems to be no systematic difference between the branch-and-price algorithms for any of the three parameters described in this table. The branch-and-cut algorithm solves very little nodes in its branching tree, compared to the branch-and-price algorithms. In many cases, the branch-and-cut algorithm prefers generating valid inequalities in the root node to branching; in fact, the majority of the instances is solved without any branching.

As explained in Section 5.1, each instance consists of matrix bids that were randomly chosen out of seven possible types. The question arises though how the algorithms would perform on instances that are composed out of only one single bid type. To this end we generated 10 instances for every bid type with 10 bidders and 50 items. We chose instances with 10 bidders and 50 items, since the algorithms handle instances of this size in a similar computation time. We also investigated a random bid type, where the entries in the matrix bid are simply randomly picked numbers between 0 and 10.

Table 10 gives an overview of the average computation time needed by the three algorithms to solve instances with bids of the mentioned bid types. The last line in the table repeats the computation times mentioned in Tables 4–6 for instances with 50 items and 10 bidders and a combination of various bid types. The table clearly shows that the difficulty of the various bid types is diverse. Additive preference bids, single minded bids, nested flat bids and non-increasing bids are indeed solved much faster than the other bid types. For most of these bid types, branch-and-price with branching

Table 10
Average computation times (s) for various bid types

Bid type	BOI	BOS	B&C
Additive preference bid	12.71	7.88	5.91
Single-minded bid	0.13	0.13	9.75
Nested flat bid	0.14	0.14	12.11
Nested *k*-of bid	[a]	[a]	[a]
Partition bid	[a]	[a]	61.11
Add-on bid	73.49	85.34	82.14
Non-increasing bid	0.43	0.43	2.94
Random bid	81.72	106.43	282.02
Mixed bid types	60.71	67.58	28.33

[a]Not all instances could be solved.

on a pair of successive of items is the fastest algorithm. On the other hand, the instances with nested *k*-of bids and partition bids are a hard nut to crack for the branch-and-price algorithms, since not all instances could be solved (due to a lack of memory). The latter bid type turns out to be very difficult for the branch-and-cut algorithm as well, since for some instances, even over 80 h of computation time did not suffice to solve them to optimality. Finally, instances with add-on bids and random bids also require quite some computation time. For these instances, branch-and-price with branching on an item–bidder pair performs best; especially for the random instances, the difference with branch-and-cut is considerable. These results further explain the variance in Tables 1–3, and hint that the branch-and-price algorithm works well for some bid types, whereas for other bid types, the branch-and-cut algorithm is to be preferred.

6. Conclusion and further research

In this paper, we studied the winner determination problem for the matrix bid auction. We first looked at a special case of the matrix bid auction, namely where all bidders have an identical ranking of the items. For this auction, we found that there exists no polynomial-time approximation scheme for the winner determination problem, unless $P = NP$. However, there exists a polynomial-time algorithm in the case the number of bidders is fixed. Then, we compared two mathematical formulations for the winner determination problem of the general matrix bid auction. One formulation is based on the assignment problem, while the other is based on the set packing problem. We found that both formulations are equally strong. Moreover, an integral solution for one formulation can always be translated to an integral solution for the other formulation. We used the set packing formulation as a basis for a column generation approach where the pricing problem can be solved as a shortest path problem. This means that we are able to solve the LP-relaxation of the set packing formulation in polynomial time. Computational experiments show that this column generation approach works faster than the standard Ilog Cplex approach on the LP-relaxation of the assignment formulation. We then extended the column generation approach to two branch-and-price algorithms. In one algorithm, we branch on the items, while in the other, branching is done on the succession of the items. The pricing problem for these branch-and-price algorithms remains solvable as a shortest path problem throughout the search tree. These algorithms are tested on randomly generated instances with up to 50 items and 100 bidders, which they solved within 20 min (on average). Finally, the branch-and-price algorithms withstood the comparison with a branch-and-cut algorithm, based on Day and Raghavan [13]. We noticed that the performance of the various algorithms depends on the bid type used in the instances; the branch-and-price algorithms deal very well with some bid types, while for other bid types, the branch-and-cut algorithms are the better choice. Nevertheless, for instances with mixed bid types, we can conclude that the algorithms perform better on instances with up to 10 items, but are outperformed by the branch-and-cut algorithm on some of the larger instances. The increase in computation time, however, seems favorable for the branch-and-price algorithms, which indicates that they form at least a viable approach to solve instances of the matrix bid auction winner determination problem.

One possible extension of our work would be to allow a bidder to submit multiple matrix bids. With some small changes to ensure that at most one bid per bidder can be accepted, the branch-and-price algorithms could still be used in such a setting. Furthermore, an interesting topic for further research may be to develop a way to represent an arbitrary

set of bids using as few matrix bids as possible. If this number turns out to be small, the resulting winner determination problem may still be efficiently solvable in practice.

References

[1] Cramton P, Steinberg R, Shoham Y. Combinatorial auctions. Cambridge, MA: MIT Press; 2005.
[2] Van Hoesel S, Müller R. Optimization in electronic markets: examples in combinatorial auctions. Netnomics 2001;3(1):23–33.
[3] Sandholm T. Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence 2002;135(1–2):1–54.
[4] de Vries S, Vohra RV. Combinatorial auctions: a survey. INFORMS Journal on Computing 2003;3:284–309.
[5] Nisan N. Bidding and allocation in combinatorial auctions. In: EC'00: proceedings of the 2nd ACM conference on electronic commerce, Minneapolis, USA. New York: ACM Press; 2000. p. 1–2.
[6] Tennenholtz M. Tractable combinatorial auctions and b-matching. Artificial Intelligence 2002;140(1/2):231–43.
[7] Day RW. Expressing preferences with price-vector agents in combinatorial auctions. PhD thesis, University of Maryland; 2004.
[8] Goossens D. Exact methods for combinatorial auctions. PhD thesis, K.U.Leuven; 2006.
[9] Leyton-Brown K, Shoham Y. A test suite for combinatorial auctions. In: Cramton P, Steinberg R, Shoham Y, editors. Combinatorial auctions. Cambridge, MA: MIT Press; 2005. p. 451–78.
[10] Rothkopf M, Pekeç A, Harstad RM. Computationally manageable combinational auctions. Management Science 1998;44(8):1131–47.
[11] Bleischwitz Y, Kliewer G. Accelerating Vickrey payment computation in combinatorial auctions for an airline alliance. In: WEA '05: proceedings of the 4th international workshop on experimental and efficient algorithms, Santorini Island, Greece. Berlin: Springer; 2005. p. 228–39.
[12] Goossens D, Maas AJT, Spieksma FCR, van de Klundert JJ. Exact algorithms for procurement problems under a total quantity discount structure. European Journal of Operational Research 2007;178(2):603–26.
[13] Day RW, Raghavan S. Matrix bidding in combinatorial auctions. Manuscript; 2006.
[14] Kann V. Maximum bounded 3-dimensional matching is max snp-complete. Information Processing Letters 1991;37(1):27–35.
[15] Petrank E. The hardness of approximation: gap location. Computational Complexity 1994;4(2):133–57.
[16] Dantzig GB, Wolfe P. Decomposition principle for linear programs. Operations Research 1960;8:101–11.
[17] Vanderbeck F, Wolsey L. An exact algorithm for ip column generation. Operations Research Letters 1996;19:151–9.
[18] Ryan D, Foster B. An integer programming approach to scheduling. In: Wren A, editor. Computer scheduling of public transport: urban passenger vehicle and crew scheduling. Amsterdam: North-Holland; 1981. p. 269–80.
[19] Lasdon L. Optimization theory for large systems. New York: Macmillan; 1970.
[20] Leyton-Brown K, Shoham Y, Tennenholtz M. An algorithm for multi-unit combinatorial auctions. In: AAAI/IAAI '00: proceedings of the 17th national conference on artificial intelligence and 12th conference on innovative applications of artificial intelligence, Austin, USA. Cambridge, MA: AAAI Press, MIT Press; 2000. p. 56–61.
[21] Parkes DC. iBundle: an efficient ascending price bundle auction. In: EC '99: proceedings of the ACM conference on electronic commerce, Denver, USA. New York: ACM Press; 1999. p. 148–57.
[22] Leyton-Brown K, Nudelman E, Shoham Y. Empirical hardness models for combinatorial auctions. In: Cramton P, Steinberg R, Shoham Y, editors. Combinatorial auctions. Cambridge, MA: MIT Press; 2005. p. 479–504.
[23] Sandholm T, Suri S, Gilpin A, Levine D. CABOB: a fast optimal algorithm for combinatorial auctions. Management Science 2005;51:374–90.
[24] Hoos H, Boutilier C. Solving combinatorial auctions using stochastic local search. In: AAAI/IAAI '00: proceedings of the 17th national conference on artificial intelligence and 12th conference on innovative applications of artificial intelligence, Austin, USA. Cambridge, MA: AAAI Press, MIT Press; 2000. p. 22–9.