RC 22166 (W0105-017) May 23, 2001
Mathematics

IBM Research Report

A Column Generation Approach for Combinatorial Auctions

Brenda L. Dietrich, John J. Forrest
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

A column generation approach for combinatorial auctions

Brenda Dietrich, John J Forrest
Mathematical Sciences Department
IBM T. J Watson Research Center
Yorktown Height, NY

Abstract

A column generation approach is used to determining the set of winning bids in a combinatorial auction for multiple, unique items. This approach is appropriate for auctions in which an individual agent places bids on multiple combinations, and will accept, subject to some restrictions, one or more of the specified combinations. Preliminary computational results appear promising.

Introduction

Auctions have been used in the sale of items for many years. Auctions provide a means of making scare items available to a large audience, and of ensuring that the seller receives the maximum revenue for the items. Many forms of auctions exist, including open-cry auctions, silent auctions, ascending bid auctions and descending bid auctions. With the advent of the Internet and other forms of electronic communications, auctions are becoming increasingly common.

Most auctions are for single items, or for multiple copies of the same item. Auctions for related items are often held simultaneously. There are numerous instances where the value to a buyer (or, in our terminology, an agent) of a set of items may not equal the sum of the value (to that agent) of the individual items. In some cases the value of a collection of items, such as an airline ticket, a hotel room, and theater tickets, may be higher than the sum of the value of the individual items. In this case the items (airline ticket, hotel, theater tickets) are said to complement one another. In other cases the sum of the values of the individual items, such as theater tickets and concert tickets for the same time and date, may be higher than the value of the combination. In this case the items are said to substitute for one another. Therefore an agent, attempting to obtain a collection of items, or to obtain one collection from a set of specified collections, through participation in several single item auctions is faced with a dilemma. She may bid on all the items in her desired collection, and obtain some, but not all, of the items in that collection. Even if the total amount she bids does not exceed her value for the collection, she may end up paying more for the items she receives than she values that subset of the collection. Similarly, if she bids for items that substitute for one another, such as the theater tickets and concert tickets, she may end up with both items, at a cost greater than her value for the pair, even if the amount she bids for each item does not exceed her value for that item.

Combinatorial auctions, in which an agent can bid for combinations of items, are beginning to be used. Unfortunately, whereas determining the winner of a single item auction is trivial, even when the auction is held over the Internet with thousands of participating agents, determining the winner of a combinatorial auction, even a relatively small combinatorial auction, is far more difficult. For n items, there are $2^n - 1$ possible combinations. Of course, it is not expected that bids will be placed on every possible combination. However, the problem of finding from among a large set of bids, each for a different combination of items, the set of bids that includes each item at most once is known to be computationally difficult. That is, it is a member of a collection of problems called NP-complete, which means that there are no known computational methods that are guaranteed to solve the problem in a number of computations that is bounded by a polynomial in the size of the input data. In practice, exact solution methods for NP complete problems typically require computation resources that grow very rapidly with the problem size, rendering them impractical for all but the smallest instance of the problem. Rothkopf [MR] provides an excellent survey of combinatorial auctions, including a discussion of particular forms of auctions that can be efficiently solved

For combinatorial auctions, where bidding is typically either continuous or in rounds, and each agent requires information about whether each of his bids is "currently" a winning bid, the set of winning bids must be computed quickly, and re-computed as new bids are added or the value bids are increased. We propose a column generation approach for determining the set of winning bids for combinatorial auctions. Our method allows for the use of certain forms of side constraints, and allows for rapid calculation of minimum bid values and the determination of tie solutions. Our formulation also allows for rapid updates as new bids are submitted, and provides a natural means for fast calculation of approximate feasible solutions rapidly. Preliminary computational experience is promising.

Our column generation approach supports two forms of combinatorial bidding, arbitrary conjunctions (ands) and a limited set of disjunctions (ors). Research in the area of combinatorial bid representation remains active. Nisan provides a lucid discussion of bid representations in [NN]. We illustrate our column generation approach with a representation chosen specifically to allow an agent to limit his total liability while placing bids on an arbitrary number of arbitrary combinations. Alternate representations can be accommodated by appropriately modifying the column generation procedure.

Each agent is allowed to place an arbitrary number of bids. Each bid can contain any combination of items. A bid consists of the following data elements: agent id, bid value, and items included in the bid. Each player can also specify a set of bid types, and can assign one or more type to each of his bids. Preference for combinations of items is expressed by placing the items in the same bid. Substitution of bids (that is, the desire to have one, but not both, or a pair of bids) is expressed by assigning the bids different types. A winning combination of bids will satisfy the following properties:
(A) each item is included in only one winning bid.
(B) all of the winning bids belonging to each agent must be included in a single type.

Furthermore, among all combinations of bids satisfying these two constraints, the winning combination must maximize total revenue, given as the sum of the values of the winning bids.

To understand property (B), note that we allow a bid to be assigned to more than one type. If a player has a bid *b1* that is assigned to type *t1*, a bid *b2* that is assigned to *t1* and to *t2*, and a bid *b3* that is assigned to *t2*, then a winning combination can include *b1* and *b2*, because they both belong to type *t1*. A winning combination could also include *b2* and b3, because they both belong to type *t2*. The pair *b2* and *b3* cannot appear in any winning combination because there is no type that includes both of these bids.

Standard combinatorial auctions as described in Rothkpopf's paper correspond to the case where each player assigns all of his bids the same type. Exclusive-or auctions, where each player is allowed to win only one combination, correspond to the case where each player assigns each bid a unique type.

We note that this specification can easily be converted to the case where each bid is assigned to only one type, through the use of duplicate bids. In the example above, bid *b2* would be submitted twice by the player, once assigned type *t1* and once assigned type *t2*. Allowing multiple types to be assigned to a bid allows for a smaller optimization problem, and may allow for more compact user interfaces.

We first present an obvious integer programming formulation of the winning bid selection program. This formulation is an extension, incorporating the type constraint, of the formulation given by Rothkopf.

Formulation

Let I denote the set of items, P denote the set of agents, and B denote the set of bids. For each $k \in B$ let $S_k \subseteq I$ denote the set of items in the bid, $v_k \geq 0$ denote the value of the bid, and $p(k) \in P$ denote the agent. We let T denote the set of types, and for each agent $p \in P$ and each type $t \in T$ we let $B_{p,t}$ denote the set of bids of type t made by agent p. Note that, as remarked above, for a pair of types $t \neq t'$, and an agent p the sets $B_{p,t}$ and $B_{p,t'}$ need not be disjoint (that is, a bid made by an agent can be of more than one type), but that for a pair of agents $p' \neq p$ the sets of bids submitted by the agents, $B_{p,t}$ and $B_{p',t'}$ are disjoint. That is, each bid is associated with one or more types, but with only one agent. We note that because of the type constraint, it is not possible to filter the bids so as to include, for each set of items, only a single bid or a single agent.

To designate whether a bid $k \in B$ is included in the winning combination of bids, we use decision variables x_k, each of which must take either the value 0 (indicating that the bid is not in the winning combination) or the value 1 (indicating that the bid is in the winning combination). We also introduce additional 0-1 variables to indicate which type is selected for each agent. For each $p \in P$ and $t \in T$ we let $y_{y,t}$ be 1 if the selected bids of agent p are from type t, and 0 otherwise.

The set of winning bids can be determined by solving the following integer program:

$$Max \sum_{j \in B} v_j x_j$$

Subject to

$$\sum_{i \in S_j} x_j \leq 1 \qquad \text{for all } i \in I \qquad (1)$$

$$\sum_{t \in T} y_{p,t} \leq 1 \qquad \text{for all } p \in P \qquad (2)$$

$$x_j - \sum_{j \in B_{p,t}} y_{p,t} \leq 0 \qquad \text{for all } j \in J \qquad (3)$$

$$x_j \in \{0,1\} \qquad \text{for all } j \in J \qquad (4)$$

$$y_{p,t} \in \{0,1\} \qquad \text{for all } p \in P, t \in T \qquad (5)$$

Constraint (1) says that each item is in at most one winning bid. If it is required that each item be in exactly one winning bid (that is, that all of the items be sold) then the inequality should be changed to an equality. In this case, if there is an item for which no bids are submitted, the integer program may be infeasible. To prevent infeasibility and to also represent a minimum value for each item, one can include a dummy bid for each individual item, with the value being the reserve value for that item. Constraint (2) says that at most one type is selected for each agent, and constraint (3) says that if a bid is selected, then for that agent a type that includes that bid has also been selected. Constraints (4) and (5) enforce integrality on the decision variables. If each agent uses only one type of bids, then the y variables for that agent are not required and constraints (2), (3) and (5) are deleted. Similarly, if for agent p each type includes at most one bid and each bid has a unique type, then the y variables for agent p are not required. We use B_p to designate the bids from agent p, delete constraint (3) for agent p and replace constraint (2) for agent p by the following, simpler constraint:

$$\sum_{j \in B_p} x_j \leq 1 \qquad (2')$$

In the case where each bid is assigned to one type, there will be a single y variable in constraint (3).

The integer program (1)- (5) is easy to formulate from the auction data. Any integer programming solver, such as OSL or CPLEX can be used to solve problems of this form. However, the computation time and the amount of computer memory required to solve even moderate sized auctions with this formulation may be excessive, as the number of items or bids increases.

We use a column generation formulation that, for small to moderate sized problems, appears to require remarkably little computation time and computer memory to solve the problem. We expect that these computational advantages will carry over to some larger problems, where it may not be necessary to specify the full optimization problem in order to find the winning combination. Furthermore, this approach permits a wide variety of side constraints on permissible combinations of bids form any one agent.

Rather than consider each bid individually, and use a number of constraints to indicate which combinations of bids can be selected simultaneously, we generate, for each agent, the combinations of that agent's bids that can be simultaneously accepted. We call such combinations proposals, because each represents a "proposed" set of bids to be awarded to the agent. Requirement (A) says that the bids belonging to a proposal can have no items in common. Requirement (B) says that the bids belonging to a proposal must all be included in a single type.

Various methods can be used to generate the set of proposals. If $B_{p,t}$ contains only a few bids, an enumeration and elimination process is very fast. For each $p \in P$ and $t \in T$, enumerate all subsets of $B_{p,t}$. From this collection of sets of bids, eliminate all sets C such that there exists $j, j' \in C$ with $S_j \cap S_{j'} \neq \phi$. Denote the collection of remaining sets of bid by $\Phi_{p,t}$. When $B_{p,t}$ contains many bids the following algorithm can be used to construct $\Phi_{p,t}$. We may assume that the bids in $B_{p,t}$ are labeled $1,2,3,...,n$.. For each positive integer $k = 1,2,3,...,n$ let $\Phi^k_{p,t}$ denote the collection of subsets of $\{1,2,3,...,k\}$ such that for each $j, j' \in C \in \Phi$, $S_j \cap S_{j'} \neq \phi$. Note that $\Phi^1_{p,t} = \{\phi, \{1\}\}$ is trivial to construct, and note that the set $\Phi^k_{p,t} = \Phi^{k-1}_{p,t} \cup \{C \cup \{k\} \ s.t. \ C \in \Phi^{k-1}_{p,t} \ and \ \overline{S}_C \cap S_k = \phi \ \}$ can be constructed from $\Phi^{k-1}_{p,t}$ by copying $\Phi^{k-1}_{p,t}$ and determining whether the bid k can be added to each element of $\Phi^{k-1}_{p,t}$.

For each $C \in \Phi_{p,t}$ we use $\overline{S}_C$ to denote the set of items in the bids in proposal C, that is, $\overline{S}_C = \cup_{k \in C} S_k$. The amount that player p would pay for the proposal C is denoted w_C and is equal to the sum of the value of the bids in C, given by $w_C = \sum_{j \in C} v_j$. Once the sets $\Phi_{p,t}$ are computed, the winning combination of bids can be determined by selecting at most one proposal for each agent, while requiring that each item is in at most one selected combination.

We let $\Phi_p = \cup_t \Phi_{p,t}$ be the proposals for agent p and we let $\Phi = \cup_p \Phi_p$ be the set of all proposals. For each $C \in \Phi$ we use a 0-1 decision variables z_C which indicates whether the proposal is selected (1) or not (0). We include constraints that allow only one proposal to be selected from each agent. However, since the type restrictions are satisfied by all of the generated proposals, we do not need to include constraints to enforce this restriction. Although each proposal is generated so that each item is contained in at most one bid in that proposal, proposals from different agents may include the same items. Therefore we require a constraint that ensures that the winning combination of proposals includes each item at most once. We let $\overline{S}_C$ be the set of items in the bids in proposal C, that is, $\overline{S}_C = \cup_{k \in C} S_k$.

The set of winning bids can be determined by solving the following integer program:

$$Max \sum_{C \notin \Phi} w_C z_C$$

Subject to $\sum_{C \in \Phi_p} z_C \leq 1$ for all $p \in P$ (6)

$$\sum_{C \in \Phi, i \in \overline{S_C}} z_C \leq 1 \qquad \text{for all } i \in I \qquad (7)$$

$$z_C \in \{0,1\} \qquad \text{for all } C \in \Phi \qquad (8)$$

Constraint (6) says that at most one proposal from each agent is in the winning combination. Constraint (7) says that each item is in at most one selected proposal. If each item must be sold, then the inequality in (7) should be changed to an equality. Potential infeasibility is addressed as in the previous formulation, by including a dummy agent and a dummy proposal for each individual item. Constraints (8) enforces integrality on the decision variables. Although this formulation may potentially have far more decision variables than the original formulation, our computational experience with this new formulation indicates that for moderate size problems it can be solved with minimal computational effort. This faster computation time may be due in part to the fact that all of the constraints have the same structure (sum of a set of variables is less than or equal to 1), and that the variables partition naturally in to sets of variables associated with each player. This partition, into so-called "special ordered sets," can be taken advantage of in commercial integer programming software.

Once a winning combination $Q \subset \Phi$ has been determined, we can extend this formulation to check to see whether another solution with equal revenue exists by adding a constraint to the model to prevent the proposals in Q from all being selected simultaneously.

$$\sum_{C \in Q} z_C \leq |Q| - 1 \qquad (9)$$

In the case of duplicate proposals of differing types, we augment the set

Q to form $Q' = Q \cup \{C \in \Phi \; s.t \; \overline{S}_C = \overline{S}_{C'} \; and \; w_C = w_{C'} \; for \; some \; C' \in Q\}$ by adding proposals that have the same underlying sets and values, and replace Q by Q' in constraint (9). If the objective function value of this augmented integer program is the same as the objective function value of the original integer program, then we have a tie solution. Otherwise the winning solution is unique. Alternatively, we can use the "find all optimal solutions" function of OSL to find all optimal integer solutions to the original problem.

For any bid that is not in the winning solution, we can determine surrogate value for the bid by solving an integer program in which the bid is forced in to the winning solution. Let b be the bid in question, and let p be the agent placing the bid. To formulate this integer program IP(b), we restrict Φ_p to proposals containing the bid b, and we change the special ordered set inequality for agent p to equality. We also eliminate all other proposals from all other agents that contain any of the licenses in the bid b. The surrogate value for the bid b is the optimal value of the original IP, minus the optimal value of IP(b). The resulting IPs have far fewer constraints than the original IP and solve relatively quickly; however there are a large number of them. The reason for the fast solution time is, at least in part, attributable to the fact that the number of non-integer elements in a solution is bounded by the number of constraints. The solution to one IP may provide a lower bound on the solution of other IPs, but beyond this obvious connection we have not found a way to speed the solution of this collection of problems.

Our limited computational experience to date is based on sample data obtained from the FCC to test a solver for their upcoming auction 31. The FCC provided seven data sets, four of which appeared designed to test specific features of the solver, and two of which appeared to represent realistic bidding. We used the largest data set to synthesize additional data, by increasing the number of agents and items, and by increasing the density of items per bid.

Problem id	#agents	#items	# types	Avg # bid /player/type	Bid value Deviation
Easy1	27	12	1	8	0.199
Easy2	28	12	2	9.27	0.455
Easy4	29	12	4	7.86	0.599
Easy10	29	12	10	4.55	0.626
Easy16	29	12	16	3.12	0.622
Base1	16	12	1	9.81	0.146
Base2	18	12	2	8.28	0.181
/6/0/0.5	24	12	2	7.17	0.188
/12/0/0.5	30	12	2	6.37	0.196
/18/0/1.0	36	12	2	5.64	0.199
/6/0/1.0	24	12	2	7.9	0.199
/12/0/1.0	30	12	2	7.97	0.189
/18/0/1.0	36	12	2	8.03	0.178
/18/16/1.0	36	16	2	8.03	0.483
/18/24/1.0	36	20	2	8.03	0.487

For these problems we generated all valid proposals. Surprisingly, the LP relaxation of the IP formulation yielded integer solutions in all instances. The column labeled LP solve time indicates the total CPU time to set up and solve the IP on a 500Mhz Intel-based laptop computer using Version 3.0 of OSL. For each losing bid, we also solved the LP obtained by forcing that bid into the solution. The number of these surrogate problems ranged from 7 to 266 for the different data sets. A very large percentage of the surrogate problems also yielded integer solutions to the LP relaxation, and those for which the LP solution was fractional typically required remarkably few branch and bound nodes to reach the optimal integer solution.

Problem id	# proposals	LP cpu sec	Surrogate value problems	Avg cpu Per surrogate	% non int
Easy1	36722	1.5	205	1.58	0
Easy2	49777	2.4	235	1.83	0
Easy4	57799	2.8	169	1.95	0
Easy10	60003	2.7	36	2.19	0
Easy16	60004	2.1	7	3.29	0
Base1	23845	1	152	1.04	18.42
Base2	29116	1.4	135	1.60	33.33
/6/0/0.5	19531	1.2	156	1.21	39.1
/12/0/0.5	13761	0.9	178	0.85	35.97
/18/0/1.0	13640	0.9	185	0.81	29.19
/6/0/1.0	30485	1.9	180	1.79	32.11
/12/0/1.0	33324	1.9	226	2.26	20.35
/18/0/1.0	38570	2.3	266	2.62	26.69
/18/16/1.0	38570	3	260	2.04	0
/18/24/1.0	38570	3.2	260	2.59	13.46

These small problems were surprisingly easy to solve. Additional examination of these and other data sets is required to understand whether the ease of solution is a result of the proposal–based formulation, or due to some other aspect of the problem.

Larger problems

For larger auctions, we propose a column generation approach in which only a subset of the feasible proposals are considered. The bids are used to generate an initial set of proposals. This initial set of proposals should, if possible include at least one proposal for each agent and at least one proposal for each item. In general, the initial set of proposals should include proposals that are of high value, relative to their contents. If information about the relative value of each item is available, this can be used to select combinations. Otherwise, one can make various estimates for the value of an item based on the value and number of items of each bid that contains the item. For each item $i \in I$ let n_i be the number of bids that contain i. If $n_i = 0$, then there are no bids that contain the item and clearly its value can be estimated to be 0. Otherwise one can estimate the value of i, denoted π_i, by averaging the value per item ratio of the bids that contain i:

$$\pi_i = \frac{1}{n_i} \sum_{k \in B, i \in S_k} \frac{v_k}{|S_k|}$$

Using such an estimate for the value of each item, and defining the "excess" value of a bid to be the value of the bid minus the sum of the estimated value of the items in the bid, we compute "excess values" of a bid as $\tilde{v}_k = v_k - \sum_{i \in S_k} \pi_i$. Using these values, together with a greedy algorithm, one can quickly compute an initial set of proposals that have high "excess" value. Various methods can be used to ensure that there is at least one proposal for each item (that has a bid) and at least one proposal for each agent or agent-type combination. If each item must be sold, then for each item select the bid containing that item that has highest excess value, and add the proposal consisting of only that bid to the set Φ of proposals. If at least one proposal from each player must be considered, then for each player (or for each player, type combination) we generate the proposal consisting of the bid from that agent (or agent-type combination) with the highest excess value and add that proposal to Φ. To generate additional high value proposals, consider only the bids with positive excess cost. For each agent *p* and each type t, sort the positive excess value bids in $B_{p,t}$ in order of decreasing excess cost, $l(1), l(2), ..., l(n)$. Generate the proposal $C = \{l(1)\}$ and add it to the set of proposals. For each bid $k = 2,3,...,n$ such that $\bar{S}_C \cap S_{l(k)} = \phi$, set $C = C \cup \{l(k)\}$ and add the proposal C to the set Φ. If additional bids are required, a randomized greedy strategy, in which the bids are considered in other orders, or in which eligible bids are added to the proposal according to some probability distribution, can be used.

Once a sufficient number of proposals have been generated, the IP corresponding to these proposals are constructed. We solve only the linear programming relaxation of this IP, and retrieve the dual variables associated with each constraint. For an item i, we can use the dual variable π_i of the corresponding constraint (7) as an estimate of the value of the item. Using these new estimates, we can again compute the "excess" value of bids. We use the dual variables λ_p associated with the agent constraints as a threshold for the acceptance of a new proposal. A proposal C for agent *p* can increase the objective function value of the linear programming relaxation of the integer program if and only if

$$w_C - \sum_{i \in \bar{S}_C} \pi_i > \lambda_p \quad . \qquad (10)$$

Expanding w_C according to its definition, we can rewrite this condition in terms of the "excess" value of the bids.

$$w_C - \sum_{i \in \bar{S}_C} \pi_i = \sum_{k \in C} (v_k - \sum_{i \in S_k} \pi_i) = \sum_{k \in C} \tilde{v}_k > \lambda_p \quad . \qquad (11)$$

Intuitively, since we can select only one proposal per agent, and a large dual variable λ_p would indicate that there are many "good" proposals for that agent already being considered, we should give preference to agents for whom the dual variable λ_p is small. Determining whether for a given agent p and type t such a proposal exists is equivalent to determining whether there is a disjoint collection of bids in the set $B_{p,t}$ that has total excess value greater than λ_p. Since the sets $B_{p,t}$ will typically contain a fairly small number of bids, and can be expected to involve only a small set of the items in the auction, the existence of such a collection can be, if necessary, verified by complete enumeration of the bids having excess value greater than $Min\{\lambda_p, 0\}$. Various efficient search strategies can also be used. In many cases, if such a collection exists it can be quickly found by applying a greedy algorithm.

If a new proposal C is found, it may also be desirable to generate additional proposals that "complete" C, especially if some, or all of the items must be sold. Clearly, since only one proposal per agent can be selected, the additional proposals to "complete" C should be for agents other than the agent p associated with C. Various strategies can be used to generate these additional proposals. A simple strategy would be to increase the excess value of bids from agents other than p that do not contain any items in $\overline{S}_C$, and then apply the usual proposal generation strategy. Additionally, computational experience with column generation methods for other large problems has taught us that it is often helpful to add randomly generated columns (in this case, proposals) to the restricted problem. Computational experiments, preferably with real auction data, are required to determine the most promising strategies.

If no proposal that satisfies (11) exists, then the current set of columns is used to formulate the integer program and the solution process continues as in the case of complete proposal enumeration. If proposals satisfying (10) are found, then the corresponding columns are added to problem, the LP is solved, and the column generation process continues.

If all possible proposals are generated, then the approach is guaranteed to find the mathematically proven optimal solution. If only a subset of the possible columns are generated, linear programming duality theory proves that the linear programming relaxation of the integer program is solved to optimality. However, the fact that there are no additional columns that can improve the linear programming objective function does not imply that there are no additional columns that can improve the integer programming objective function value.

In many cases, such as in the early rounds of an auction, it may be sufficient to provide a near optimal solution. In later rounds, branch-and-price method can be used to identify additional columns in the course of searching for an integer solution. However, since we need to determine the existence of tie solutions, and surrogate values for bids not in a winning combination, we suggest the following approach, which uses information about the solution to restricted IPs (containing only a subset of the columns) to either identify additional columns, or to prove optimality of the solution to one of the restricted IPs..

Once a our column generation procedure stops with a set of optimal (LP) columns and an integer solution has been found for this set of columns, we add constraint (9), which was used to detect the presence of a tie solution, to the LP and resolve. Adding this constraint cuts off the optimal solution to the restricted IP. It may or may not cut off the optimal solution to the LP. If it does cut of the optimal solution to the LP, then the column generation process, applied to this new IP may identify additional columns. It is also possible, that no new columns are found, but the LP objective is decreased enough to prove optimality of the IP solution. Finally, it is also possible that adding this constraint changes the LP optimal solution, but does not aid in either identifying new proposals or in proving optimality of the current solution.

For each losing bid b, we determine a feasible integer solution that selects the bid. For the agent p placing the bid, we consider only proposals containing bid b, while for other agents, we consider only proposals that do not intersect with bid b. We use the set of known proposals to formulate the IP, solve its LP relaxation, and apply column generation, with the restriction that any proposals for agent p must contain bid b, while any proposals for other agents must not intersect with bid b. These problems can all be solved in parallel. If solution time is critical, especially in the early rounds of an auction, it may not be essential to solve each of the problems to optimality. We add all proposals generated in this manner to the original IP and resolve. Then for each losing bid associated with this new solution, we compute an estimate of the surrogate cost as the value of the (new) IP minus the largest value among the feasible solutions that select the bid.

References

Nisan, N., “Bidding and Allocation in Combinatorial Auctions,” presented at the 1999 Conference “Northwestern's Summer Workshop in Microeconomics.”

Rothkopf, M., “Computationally Manageable Combinatorial Auctions,” *Management Science,* **44,** 1998, 1131-1147.

Tsukiyama, Ide, Ariyoshi and Shirakawa in “A New Algorithm for Generating all the Maximal Independent Sets,” *Siam Journal on Computing*, **6**, 1977, 505-517.