Declarative specification and solution of combinatorial auctions using logic programming

Chitta Baral and Cenk Uyan

Department of Computer Sc. and Engg.
Arizona State University
Tempe, Arizona 85287
{*chitta,cuyan*}@*asu.edu*

Abstract. In a combinatorial auction problem bidders are allowed to bid on a bundle of items. The auctioneer has to select a subset of the bids so as to maximize the price it gets, and of course making sure that it does not accept multiple bids that have the same item as each item can be sold only once. In this paper we show how the combinatorial auction problem and many of its extensions can be expressed in logic programming based systems such as Smodels and dlv. We propose this as an alternative to the standard syntax specific specialized implementations that are much harder to modify and extend when faced with generalizations and additional constraints.

1 Introduction and Motivation

In a simple auction several bidders bid for an item and the auctioneer selects the highest bid. Often bidders need a bundle of items, where the worth of the whole bundle – to the bidder – may be more than the sum of the individual worth of each item in the bundle. For example, let A and B be two adjacent real estate plots. A single developer can often make more money developing both plots together than two different developers developing A and B separately without co-operating with each other. This happens if say both A and B are needed to create a lucrative golf course while A and B separately can only be used for less profitable purposes. The opposite may be true in some cases too. The cases that are often mentioned with regards to both are airport landing slots [9], bandwidth auctions, real estate auctions, and transportation exchanges [11].

In such cases participating in parallel or sequential auctions for each items in a bundle desired by a bidder is risky as the bidder may not win all items in the bundle. Moreover it would be difficult for him to individually price each item in the bundle. One way to avoid such problems is to have *combinatorial auctions* where bidders are allowed to bid on bundles. Although this is good for the bidder, the seller's problem of deciding which bids to accept becomes harder, as different bidders can make up their own bundle on which they bid on .

Recently, there has been a lot of interest in this problem because of its applicability in Internet based auctions, B2B exchanges, and multi-agent systems [16, 7]. There have been several papers [12, 13, 2, 4, 6, 15, 10] that analyze this problem and present algorithms and techniques to solve it and a few of its generalizations. One starting point that guides research on this is the result from [10] which shows the problem of finding the optimal set of bids (that maximize the seller's take) to be NP-Complete.

So far there are three different approaches for solving this problem: complete algorithms [12, 2, 10] that find an optimal solution in the general case, incomplete methods [4] that find high quality solutions quickly, and identification of tractable special cases and algorithms for those cases [15, 13]. The other possible approach of finding approximation algorithms is blocked by the result from [12] that shows that no polynomial algorithm can guarantee a solution that is close to optimal.

In this paper we follow the first approach of obtaining optimal solutions in the general case. Our methodology is different from the earlier approaches [12, 2, 10] in that we would like to represent the problem in a declarative knowledge representation language such that optimal 'models' of the representation correspond to optimal solutions. This is similar to the methodology of satisfiability based planning [5] where the planning problem is represented as a propositional theory, and each model of this theory encodes a plan. *The main motivation behind our approach of using a declarative knowledge representation language is that we would like the process of adding additional constraints, or making a generalization to be easier.* This differs from the other approaches [12, 13, 2, 6] where major changes were needed to move from single unit combinatorial auctions to multi-unit combinatorial auctions. Also, as mentioned in [13] additional generalizations necessitates change in the code, which requires the knowledge of the structure of the code and hence can only be done by people adequately familiar with the original code. In contrast we will show that when using a declarative knowledge representation language additional generalization, or addition of new constraints often leads to adding a few extra rules, without needing the detailed knowledge of the original code or its structure.

The declarative language that we will use throughout this paper is Smodels [8, 14][1], an extension of logic programming with answer set semantics [3]. It has new constructs such as cardinality and weight constraints, and optimization statements. It is preferable over propositional logic as it is more expressive in terms of being able to express transitive closure, causality, and aggregation. Moreover, it is a non-monotonic language and hence more suitable for knowledge representation and finally it includes optimization statements. (A more detailed argument about the advantages of logic programming with answer set semantics over other logics is given in the draft of a book by the first author available at http://www.public.asu.edu/∼cbaral/.) Smodels is preferable over ILP (integer linear programming) because it can represent logical specifications more easily. Although ILP can accommodate propositional logic, it has not been shown how it can accommodate non-monotonic features of a logic program.

Our goal in this paper is to show how single unit and multi unit combinatorial auction problems can be specified and declaratively solved using Smodels, and how it is

[1] Strictly speaking, Smodels is a system that started of as implementing the answer set semantics of logic programs and now has several new constructs. By the Smodels language we refer to the extension of logic programs that is used by the Smodels system.
We would like to mention that some of the encodings in this paper can also be expressed in the language of the dlv system [1]. Due to lack of space we only focus on the Smodels system.

easy to add additional constraints and further generalizations to the original problem using Smodels. We hope this representation will serve as a benchmark to the logic programming, knowledge representation and declarative problem solving communities to develop more efficient implementations of the Smodels language such that the timing of obtaining solutions of combinatorial auction problems specified in Smodels is comparable to that of the specialized algorithms/programs in [12, 13, 2, 6].

2 Background: The Smodels language

A logic program is a collection of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \textit{not } a_{m+1}, \ldots, \textit{not } a_n \tag{1}$$

where a_i's are atoms. For an atom a, "*not* a" is referred to as a naf-literal. Intuitively, the above rule means that if $a_1 \ldots a_m$ are true and $a_{m+1} \ldots a_n$ can be assumed to be false then a_0 must be true. Logic programs whose rules do not have *not* in the body – referred to as definite programs – have unique answer sets, which are the least models of the theory obtained by treating rules of the form $a_0 \leftarrow a_1, \ldots, a_m$ as the classical formula $a_1 \wedge \ldots \wedge a_m \supset a_0$. Given a logic program P and a set of atoms S, the Gelfond-Lifschitz transformation P^S is defined as the set of rules obtained from P by removing all rules from P whose body contains *not* b such that $b \in S$, and then removing the naf-literals from the rest of the rules. *A set S of atoms is said to be an answer set of a logic program P if S is the answer set of the definite program P^S.*

In the Smodels language, each of the $a_0, \ldots, a_m$ can be replaced by cardinality expressions and weight expressions. An example of a cardinality expression is:

$3\ \{sold(X) : item(X)\}\ 6$

which is true in an answer set if the number of items that are sold is between (inclusively) 3 and 6. We can encode the value each item is sold by a weight declaration of the form: weight $sold(a) = 8$.

which would mean that item a was sold for \$8. Now the weight expression

$23\ [sold(X) : item(X)]\ 36$

will be true in an answer set if the total value of those items that are sold is between (inclusively) 23 and 36.

Optimization statements are syntactically similar to weight and cardinality expressions except that the left and right range are replaced by the label $maximize$ or $minimize$ in the left hand side. For example, if we want to obtain the answer sets where the number of items that is sold is maximum then we need to write the following:

$maximize\ \{sold(X) : item(X)\}.$

Smodels allows multiple optimization statements and treats them as a compound optimization through a lexicographic ordering among the optimizations statements. A more formal characterization of the Smodels language is given in [14].

3 Single unit combinatorial auction

We explain the single unit combinatorial auction problem through an example. The auctioneer has the set of items $\{1, 2, 3, 4\}$, and the buyers submit bids $\{a, b, c, d, e\}$ where a constitutes of $\langle\{1, 2, 3\}, 24\rangle$, meaning that the bid a is for the bundle $\{1, 2, 3\}$ and its price is \$24. Similarly b constitutes of $\langle\{2, 3\}, 9\rangle$, c constitutes of $\langle\{3, 4\}, 8\rangle$, d constitutes of $\langle\{2, 3, 4\}, 25\rangle$, and e constitutes of $\langle\{1, 4\}, 15\rangle$. The *winner determination* problem is to accept a subset of the bids with the stipulation that no two bids containing the same item can be accepted, so as to maximize the total price fetched. We now present an Smodels encoding (which is both a specification and a program.) of this example.

3.1 Specifying the domain

1. We specify the bid names and their values as follows:

   bid(a). weight sel(a) = 24. bid(b). weight sel(b) = 9. bid(c). weight sel(c) = 8.
   bid(d). weight sel(d) = 25. bid(e). weight sel(e) = 15.
2. We specify the items as follows: item(1..4).
3. We specify the composition of each bids – in terms of what items it consists of, as follows.

   in(1,a). in(2,a). in(3,a). in(2,b). in(3,b). in(3,c). in(4,c).
   in(2,d). in(3,d). in(4,d). in(1,e). in(4,e).

3.2 The general rules

We have the following general rules which together with the domain specific rules from the previous subsection, when run using Smodels will give us the winning bids.

1. The following two rules label each bid as either selected or not selected.

   $sel(X) \leftarrow bid(X), \textit{not}\ not\_sel(X).$
   $not\_sel(X) \leftarrow bid(X), \textit{not}\ sel(X).$

   They can be replaced by the following single Smodels rule: $\{sel(X)\} \leftarrow bid(X).$
2. The following enforces the constraint that two different bids with the same items can not be both selected.

   $\leftarrow sel(X), sel(Y), X \neq Y, in(I, X), in(I, Y).$
   The above Smodels rule does not follow the syntax of rules in Section 2. Such rules of the form
   $\leftarrow a_1, \ldots, a_m, \textit{not}\ a_{m+1}, \ldots, \textit{not}\ a_n.$
   with empty head mean that there can not be answer sets that evaluate the body true. Such rules can be thought of as the following rule, where f is a new atom, that satisfies the syntax of (1) in Section 2.

   $f \leftarrow \textit{not}\ f, a_1, \ldots, a_m, \textit{not}\ a_{m+1}, \ldots, \textit{not}\ a_n.$
3. The following optimization statement specifies that we must select bids such that their total price is maximized.

   $maximize\ [sel(X) : bid(X)].$

When we run the above program in the Smodels system using the command

lparse auc1.sm | smodels 0

the system first outputs the answer set $\{a\}$, and then outputs the optimal answer set $\{d\}$, indicating that the latter has a higher total price.

3.3 Formal characterization

In a combinatorial auction (single unit case), the auctioneer has m items $M = \{1, \ldots, m\}$ and the buyers submit n bids $B = \{B_1, \ldots, B_n\}$, where each bid is a tuple $B_i = \langle S_i, p_i \rangle$, with $S_i \subseteq M$, and p_i is a price. The *winner determination problem* [13] is an assignment of bids as *accepted* ($x_i = 1$) or not ($x_i = 0$), for $1 \leq i \leq n$ that satisfies the constraint

$$(\sum_{1 \leq i \leq n, j \in S_i} x_i) \leq 1 \qquad \text{for } 1 \leq j \leq m; \text{ and maximizes} \qquad \sum_{i=1}^{n} p_i \times x_i.$$

The above characterization can be related to the Smodels encoding as follows:

Theorem 1. *For a single unit combinatorial auction problem with integer prices, each solution to the winner determination problem corresponds to an optimal answer set of the encoding described in 3.1-3.2 and vice-versa.*

3.4 Encoding in dlv

The dlv system [1] is also an implementation of an extension of logic programming with additional constructs. It allows disjunctions in the head of rules and captures the second level of polynomial hierarchy. Among its additional constructs are weak constraints which are of the form:

:$\sim$ $p_1, \ldots, p_m, not\ q_1, \ldots, not\ q_n.[weight : level]$

Given a program with weak constraints its *best* answer sets are obtained by first obtaining the answer sets without considering the weak constraints and ordering each of them based on the weight and priority level of the set of weak constraints they violate, and then selecting the ones that violate the minimum. In presence of both weight and priority level information, the minimization is done with respect to the weight of the constraints of the highest priority, then the next highest priority and so on.

The encoding in Section 3.1 and 3.2 can be alternatively encoded in dlv[2] as follows:

1. We have the bid atoms from part 1 of Section 3.1, and in atoms from part 3 of Section 3.1.
2. We can either have the rules in part 1 of Section 3.2 or use disjunction and have the rule:

   $sel(X) \vee not\_sel(X) \leftarrow bid(X).$

[2] In the future we plan to compare the timings using the dlv system with the timings using the Smodels system.

3. We have the constraint in part 2 of Section 3.2.
4. Finally instead of the optimization statement in part 3 of Section 3.3, we have the following weak constraints.

   :∼ *not* $sel(a).[24:1]$
   :∼ *not* $sel(b).[9:1]$
   :∼ *not* $sel(c).[8:1]$
   :∼ *not* $sel(d).[25:1]$
   :∼ *not* $sel(e).[15:1]$

4 Combinatorial auction with CNF bids

In this section we show how the single unit combinatorial auction specification can be generalized such that a bidder can specify some options between his bids. For example a bidder may want to specify that only one of his bids g and h be accepted, but not both. This can be generalized further to such that a bidder can specify a CNF[3] bid [4] which is a conjunction of (ex-or) disjunction of items such that one item from each of the conjuncts is awarded to the bidder. (An alternative way to achieve this is by opening up the CNF to several bids and adding a dummy [2] item to each of the bids so that exactly one of them is selected.) As before we show our encoding with respect to an example.

1. We will have the domain specification as in part 1 and 2 of Section 3.1 and the general rules in part 1 and 3 of Section 3.2.
2. Recall that a CNF bid is not a bundle of items, rather it could be of the following form:

   $a = (g1 \oplus h1) \wedge (g2 \oplus h2) \wedge (g3 \oplus h3)$

   which means that the bid a can be satisfied by granting one of the items $g1$ and $h1$, one of the items $g2$ and $h2$ and one of the items $g3$ and $h3$. We can represent this in Smodels as follows:

   conj(c1, a). disj(g1, c1). disj(h1, c1).
   conj(c2, a). disj(g2, c2). disj(h2, c2).
   conj(c3, a). disj(g3, c3). disj(h3, c3).

   This will replace the representation in part 3 of Section 3.1.
3. Now it is not enough to just label bids as selected or un-selected. After labeling a bid as selected we must identify which items are granted as part of that selected bid. We have the following rules to encode that.

   $other\_granted(X, C, G) \leftarrow granted(X, C, G'), G' \neq G.$
   $granted(X, C, G) \leftarrow sel(X), conj(C, X),$
   $\qquad disj(G, C), not\ other\_granted(X, C, G).$

   Intuitively, $granted(X, C, G)$ means that as part of the selection of bid X, to satisfy the conjunct C, item G is granted; and $other\_granted(X, C, G)$ means that some

[3] Although the use of CNF is somewhat misleading, we use it to be consistent with the original terminology in [4].

item other than G has been granted. The above two rules ensure that for any selected bid X, and its conjunct C exactly one item in that conjunct is granted in each answer set.

4. Because of the difference between a CNF bid and a simple bid consisting of a bundle, part 2 of Section 3.2 needs to be replaced by the following rule, so as to enforce that we should not select two bids and grant the same item with respect to both.

   $\leftarrow bid(X), bid(Y), granted(X, C, G), granted(Y, C', G), X \neq Y.$

5 Multi-unit combinatorial auction

Multi-unit combinatorial auction is a generalization of the single unit case, where the auctioneer may have multiple identical copies of each item and the bids may specify multiple units of each item. The goal here is same as before: to maximize the total price that is fetched; but the condition is that the bids should be selected such that for any item the total number that is asked by the selected bids should not be more than the number that is originally available for that item. As before, we describe our Smodels encoding with respect to an example: first the specification for a particular domain, and then a set of general rules.

5.1 Specifying the domain

1. The bid names and their values are specified as in part 1 of 3.1.

   bid(a). weight sel(a) = 23.
   bid(b). weight sel(b) = 9.
   bid(c). weight sel(c) = 8.
   bid(d). weight sel(d) = 25.
   bid(e). weight sel(e) = 15.
2. We specify the items and their initial quantities as follows:

   item(i). item(j). item(k). item(l).

   limit(i,8). limit(j,10). limit(k,6). limit(l,12).
3. We specify the composition of each bid as follows:

   in(i,a,6). in(j,a,4). in(k,a,4).

   Intuitively, the above means that, bid ‘a’ is for 6 units of item ‘i’, 4 units of item ‘j’, and 4 units of item ‘k’.

   in(j,b,6). in(k,b,4).

   in(k,c,2). in(l,c,10).

   in(j,d,4). in(k,d,2). in(l,d,4).

   in(i,e,6). in(l,e,6).

5.2 The general rules

We have the following general rules which together with the domain specific rules of the previous subsection, when run using Smodels will give us the winning bids.

1. The following two rules label each bid as either selected or not selected.

   $sel(X) \leftarrow bid(X), not\ not\_sel(X).$
   $not\_sel(X) \leftarrow bid(X), not\ sel(X).$
2. The following rule defines $sel\_in(I, X, Z)$, which intuitively means that bid X is selected, and Z units of item I is in bid X.

   $sel\_in(I, X, Z) \leftarrow item(I), bid(X), sel(X), in(I, X, Z).$
3. The following weight declaration assigns the weight Z to the atom $sel\_in(X, Y, Z)$.

   $weight\ sel\_in(X, Y, Z) = Z.$

   The above weight assignment is used in the next step to compute the total quantity of each item in the selected bids.
4. The following rule enforces the constraint that for each item, the total quantity that is to be encumbered towards the selected bids must be less than or equal to the initial available quantity of that item.

   $\leftarrow Y'[sel\_in(I, X, Z) : bid(X) : num(Z)], item(I), limit(I, Y), Y' = Y + 1.$
5. As before we have the following optimization statement.

   $maximize\ [sel(X) : bid(X)].$

When the above program is run through Smodels using the command

lparse file.sm | smodels 0

the system first outputs the answer set $\{sel(d), sel(a)\}$, and then outputs another answer set $\{sel(e),\ sel(d),\ sel(b)\}$ and mentions that the latter one is optimal.

Thus the Smodels system starts off with a sub-optimal solution, and keeps giving better and better solutions until an optimal solution is found. We refer to this as exhibiting a weak anytime behavior as after the first solution is found, a user may interrupt the system at any time and get a sub-optimal solution which improves with time. Since there is no guarantee that the first solution will be found within a certain time bound we use the qualifier 'weak' with the adjective 'anytime'.

5.3 Formal characterization

In the multi-unit case, the auctioneer has u_j units of each item j, $1 \leq j \leq m$, and each bid B_i is of the form $\langle(\lambda_i^1, \ldots, \lambda_i^m), p_i\rangle$, where λ_i^j denotes the number of units of item j that is part of the bid B_i. In this case, the *winner determination problem* [13] is an assignment of bids as $accepted$ ($x_i = 1$) or not ($x_i = 0$), for $1 \leq i \leq n$ that satisfies the constraint

$$(\sum_{i=1}^{n} \lambda_i^j \times x_i) \leq u_j \qquad \text{for } 1 \leq j \leq m; \text{ and maximizes } \qquad \sum_{i=1}^{n} p_i \times x_i.$$

Theorem 2. *For a multi-unit combinatorial auction problem integer prices, each solution to the winner determination problem corresponds to an answer set of the encoding described in 5.1-5.2 and vice-versa.*

6 Combinatorial Exchanges

A combinatorial exchange is a further generalization, where we have buyers and sellers. The buyers bid as before, while the sellers offer their items for a price. The job of the exchange is to accept a subset of the bids of the buyers and sellers such that it maximizes the surplus (the amount it obtains from the buyers minus the amount it has to pay to the sellers), subject to the condition that for each item, the total number it obtains from the selected seller bids is more than what it has to give in lieu of the selected buyer bids. Note that the maximization condition guarantees that the exchange does not lose money outright. This is because by not accepting any bids the surplus will be zero. So when the exchange accepts some bids its surplus would have to be positive. We now describe our Smodels encoding for multi-unit combinatorial exchanges through a slight modification of the example in Section 5. The modification is that instead of specifying the initial quantity of each item, we create a seller f, who offers those quantities for a price.

1. We have part 1 and part 3 of Section 5.1 and only the items listing of part 2 of 5.2. We do not have the description of the initial quantity for the items. Instead the bid for the seller f is specified as follows:

   bid(f). weight sel(f) = -50.

   in(i,f,-8). in(j,f,-10). in(k,f,-6). in(l,f,-12).

   A sellers bid is distinguished from a buyers bid by having a negative price for the whole bid (meaning the seller wants money for those items, instead of being ready to give a certain amount of money), and similarly the atom $in(i, f, -8)$ means that the seller f has 8 units of item i to *sell*, while $in(i, a, 6)$ would mean that the buyer a wants to *buy* 6 units of item i.
2. We have part 1, 2, 3, and 5 of Section 5.2 and we replace part 4 by the following rule.

   $\leftarrow Y\ [sel\_in(I, X, Z) : bid(X) : num(Z)]\ Y, item(I), Y > 0.$

   The above rule enforces the constraint that for each item I, the *total number* encumbered with respect to the selected buyer bids should be less than or equal to the sum that is available from the selected seller bids. Note that the use of the same variable Y as the upper and lower bound of the weight constraint serves the purpose of computing the aggregate. [4]
3. Although the following rule is normally not necessary, as it is taken care of by the maximize statement, by having it we can exploit the weak anytime behavior. It also eliminates selections, where the exchange may lose money, earlier in the process.

   $\leftarrow Y[sel(X) : bid(X)]Y, Y < 0.$

[4] But the Smodels requirement of having a domain variable for Y (not shown in the above rule) makes it an inefficient way to compute aggregation. Having an efficient computation of aggregates together with the answer set semantics remains a challenge.

When we run the above program through Smodels it tells us to not select any bids. This is expected because the maximum amount that can be obtained from the buyers is $49 by selecting b, d and e; but to satisfy that we have to select f, which costs $50, resulting in a net loss to the exchange. On the other hand if we change our example, and assign -45 as the weight of sel(f), then the Smodels output is indeed to select b, d, e, and f.

6.1 Formal characterization

In case of a combinatorial exchange, instead of a single auctioneer, we have many sellers, who also present bids, but in their bids the λ_i^js and p_is are negative numbers denoting the fact that they want to sell (instead of buy) those items and they want to be paid (rather than they are willing to pay). Here the *winner determination problem* [13] is an assignment of bids as $accepted$ ($x_i = 1$) or not ($x_i = 0$), for $1 \leq i \leq n$ that satisfies the constraint

$$(\sum_{i=1}^{n} \lambda_i^j \times x_i) \leq 0 \qquad \text{for } 1 \leq j \leq m; \text{ and maximizes } \quad \sum_{i=1}^{n} p_i \times x_i.$$

Theorem 3. *For a multi-unit combinatorial exchange problem with integer prices, each solution to the winner determination problem corresponds to an optimal answer set of the above encoding and vice-versa.*

7 Expressing additional constraints

In this section we show how further generalizations and additional constraints can be easily expressed in Smodels.

1. Suppose we would like to express the constraint that item 1 must be sold. We can achieve this by adding the following rules:

   $sold(X) \leftarrow item(X), bid(Y), sel(Y), in(X, Y).$
   $\leftarrow$ *not* $sold(1).$
2. Suppose we would like to have reserve prices[5] in the single unit combinatorial auction. This can be encoded by the following modification of the program in Section 3. The main change is that we replace $bid(X)$ by $bid(X, Y)$ where Y was originally the weight of $bid(X)$. This change allows us to compare the sum of the reserve prices of the items in a bid with the bid price, which now is the parameter Y instead of the weight of $bid(X)$.

   As regards to the specification of the domain, the bids are specified as follows:

   bid(a,24). bid(b,9). bid(c,8). bid(d,25). bid(e,15).

   The composition of items and bids are as in part 2, and 3 of 3.1. The general rules, as described below are different from the ones in 3.2.

[5] In simple auctions reserve price of an item is the minimum price a seller would accept for that item. Its extension [13] to combinatorial auctions will become clear below.

(a) The following two rules label each bid as either selected or not selected. The third rule assigns a weight to $sel(X, Y)$.

$sel(X, Y) \leftarrow bid(X, Y), \textit{not } not\_sel(X, Y).$
$not\_sel(X, Y) \leftarrow bid(X, Y), \textit{not } sel(X, Y).$

$weight\ sel(X, Y) = Y.$

(b) The following enforces the constraint that two different bids with the same items can not be both selected.

$\leftarrow sel(X, N), sel(Y, N'), X \neq Y, item(I), in(I, X), in(I, Y).$

(c) We have the following optimization statement.

$maximize\ [sel(X, Y) : bid(X, Y)].$

(d) We express the reserve price of each item by the following:

rp(1,2). rp(2,8). rp(3,8). rp(4,12).

(e) The following rules compute the sum of the reserve prices of bids and compare them with the bid price and eliminate possible answer sets where the bid price of a selected bid is less than the sum of the reserve prices of items in that bid.

$in\_rp(Item, Bid, Res\_pr) \leftarrow in(Item, Bid), rp(Item, Res\_pr).$

$weight\ in\_rp(Item, Bid, Res\_pr) = Res\_pr.$

$item\_num(X, Y) \leftarrow item(X), num(Y).$

$\leftarrow C\ [in\_rp(Item, Bid, Res\_pr) : item\_num(Item, Res\_pr)]\ C,$
$bid(Bid, Bid\_pr), sel(Bid, Bid\_pr), Bid\_pr < C.$

3. Suppose we would like to have a constraint that item 1 and 3 must not go to the same bidder. In the simple case if we assume that each bid is by a different bidder we can encode this by the following rule.
$\leftarrow bid(X, Y), sel(X, Y), in(1, X), in(3, X).$
4. In the more general case where each bid has an associated bidder we first need to express this association as follows:

bidder(a, john). bidder(b, mary). bidder(c, john).
bidder(d,mary). bidder(e, peter).

Next we need the following rules:

$goes\_to(Item, Bidder) \leftarrow in(Item, X), bidder(X, Bidder), sel(X, Y).$

$\leftarrow goes\_to(1, B), goes\_to(3, B).$

Similarly, if we want to specify that the items 1 and 3 must go to the same bidder, then the last rule can be replaced by the following rules.

$\leftarrow goes\_to(1, B), \textit{not } goes\_to(3, B).$
$\leftarrow goes\_to(3, B), \textit{not } goes\_to(1, B).$

5. Suppose we would like to represent the constraint that every bidder must return happy, i.e., at least one of her bid must be satisfied. This can be expressed by the following:

$happy(Bidder) \leftarrow bidder(X, Bidder), bid(X, Y), sel(X, Y).$
$\leftarrow bidder(Bid, Bidder), \textit{not } happy(Bidder).$

6. Suppose the seller wants to only deal with whole sellers. I.e., it wants to have the constraint that it only selects bids of a bidder if the total money to be obtained from that bidder is more than \$100. This can be achieved by adding the following rules.

   $sel(Bid, Value, Bidder) \leftarrow bid(Bid, Value), sel(Bid, Value), bidder(Bid, Bidder).$

   $weight\ sel(Bid, Value, Bidder) = Value.$

   $total(Bidder, C) \leftarrow C\ [sel(Bid, Value, Bidder) : bid(Bid, Value)]\ C.$

   $\leftarrow total(Bidder, C), C < 100.$
7. Suppose the seller wants to avoid bid 'a' as it came late, unless it includes an item that is not included in any other bids. This can be expressed by the following rules.

   $ow\_covered(Bid, Item) \leftarrow in(Item, Bid'), Bid \neq Bid'.$
   $not\_ow\_covered(Bid) \leftarrow in(Item, Bid), \textit{not}\ ow\_covered(Bid, Item).$

   $\leftarrow sel(a, Value), \textit{not}\ not\_ow\_covered(a).$
8. To check inventory costs the seller may require that no more than 5 unsold items should be left after the selection. This can be expressed by the following rules.

   $sold(I) \leftarrow item(I), bid(X, Y), sel(X, Y), in(I, X).$
   $unsold(I) \leftarrow item(I), \textit{not}\ sold(I).$

   $\leftarrow C\ \{unsold(I) : item(I)\}\ C, C > 5.$
9. To contain shipping and handling costs the seller may require that bids should be accepted such that at least 5 items go to each bidder. This can be expressed by the following rules.

   $count(Bidder, C) \leftarrow C\ \{goes\_to(Item, Bidder) : item(Item)\}\ C,$
   $\qquad bidder(B, Bidder).$

   $\leftarrow bidder(B, Bidder), count(Bidder, C), C < 5.$
10. If item 'a' is a family treasure the seller may require that it can only be sold to bidder john or mary, his relatives. This can be expressed by the following rule.

    $\leftarrow goes\_to(a, X), X \neq john, X \neq mary.$

The above shows how additional constraints and generalizations can be easily expressed as new Smodels rules and often we do not have to change the original program, but just have to add new rules.

8 Conclusion

In this paper we showed[6] how the combinatorial auction problem and its generalizations can be expressed and solved using the declarative knowledge representation language of

[6] The Smodels system [8, 14] requires each rule to be strongly range restricted. That means every variable that appears in a rule must appear in a positive domain literal in the body of a rule. For simplifying the presentation we have omitted domain literals in some of the rules. For clarification, some of the Smodels and dlv code that we developed while testing the programs in this paper are available at http://www.public.asu.edu/∼cbaral/comb-auc/.

Smodels. We argued that the declarativeness of Smodels allows us to easily make generalizations and add additional constraints. Although our focus was more on knowledge representation, we ran some experiments with respect to synthetic examples following the approach of [6, 12]. In case of single-unit bids, our results have been comparable to those reported in [12]. In case of multi-unit bids with synthetic data drawn from a decay distribution [6] we obtained reasonable timings for bundle sizes up to 1500, with 150 items. Our timings were worse than [6] though. We did not compare with the timings in [4, 13], as the first one is about an incomplete algorithm and the second one does not report timings. We hope the programs in this paper would serve as a benchmark and a challenge to researchers in logic programming, declarative problem solving and knowledge representation in terms of having faster implementations of Smodels.

Acknowledgments This work was supported by the NASA grant NCC2-1232 and the NSF grants IRI-9501577 and NSF 0070463. We thank the anonymous reviewers for their comments.

References

1. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In *Proc. of KR 98*, pages 406–417, 1998.
2. Y. Fujisima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions. In *Proc. of IJCAI 99*, pages 548–553, 1999.
3. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press, 1988.
4. H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *Proc. of AAAI'00*, 2000.
5. H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI-92*, pages 359–363, 1992.
6. K. Leyton-Brown, Y. Shoham, and M. Tennenholtz. An algorithm for multi-unit combinatorial auctions. In *Proc. of AAAI'00*, 2000.
7. D. Monderer and M. Tennenholtz. Optimal auctions revisited. *Artificial Intelligence*, 2000.
8. I. Niemela and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429, 1997.
9. S. Rassenti, V. Smith, and R. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell J. of Economics*, 13:402–417, 1982.
10. M. Rothkopf, A. Pekec, and R. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.
11. T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. of AAAI 93*, pages 256–262, 1993.
12. T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proc. of IJCAI 99*, pages 542–547, 1999.
13. T. Sandholm and S. Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *Proc. of AAAI'00*, 2000.
14. P. Simons. Extending the stable model semantics with more expressive rules. In *Proc. of International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'99*, 1999.
15. M. Tennenholtz. Some tractable combinatorial auctions. In *Proc. of AAAI'00*, 2000.
16. M. Wellman, W. Walsh, P. Wurman, and J. Mackie-Mason. Auction protocols for decentralized scheduling. *Games and Economic bahavior*, 1999.