Answer Set Optimization*

Gerhard Brewka
Comp. Sci. Institute
University of Leipzig
Leipzig, Germany
brewka@informatik.uni-leipzig.de

Ilkka Niemelä
Dept. of Comp. Sci. and Eng.
Helsinki University of Technology
Helsinki, Finland
Ilkka.Niemela@hut.fi

Mirosław Truszczyński
Dept. of Comp. Sci.
University of Kentucky
Lexington, KY 40506-0046, USA
mirek@cs.uky.edu

Abstract

We investigate the combination of answer set programming and qualitative optimization techniques. Answer set optimization programs (ASO programs) have two parts. The generating program P_{gen} produces answer sets representing possible solutions. The preference program P_{pref} expresses user preferences. It induces a preference relation on the answer sets of P_{gen} based on the degree to which rules are satisfied.

We discuss possible applications of ASO programming, give complexity results and propose implementation techniques. We also analyze the relationship between ASO programs and CP-networks.

1 Introduction

Answer set semantics [Gelfond and Lifschitz, 1991] describes the meaning of a logic program P in terms of sets of literals. The exact definition of answer sets depends on the kind of rules used in P, yet two properties are always required. Answer sets are closed under the rules of P, and they are grounded in P: each literal has a derivation using "applicable" rules from P. Answer set programming has become a popular knowledge representation tool. There are several reasons for this:

1. Logic programs are expressive enough to model many typical knowledge representation problems in AI. In particular, the availability of default negation in the body of rules makes it possible to represent defeasible information.
2. Many problems in reasoning about actions, planning, diagnosis, belief revision and product configuration have elegant formulations as logic programs so that models of programs, rather than proofs of queries, describe problem solutions [Lifschitz, 2002; Soininen, 2000; Baral, 2003].
3. The semantics of answer sets is intuitive and avoids the pitfalls of resolution-based systems like Prolog. For instance, it is independent of the order in which rules are written and correctly handles loops.
4. In the same time, the syntax of logic programs is restrictive enough to allow for fast implementations and several highly efficient answer-set provers have been developed. Most advanced among them are *Smodels* [Niemelä and Simons, 1997] and *dlv* [Eiter *et al.*, 1998].

To increase the ease of use of logic programs in knowledge representation researchers have suggested and investigated several extensions to the basic formalism. Well-known examples of such extensions include disjunctive logic programs and programs with cardinality and weight constraints [Simons *et al.*, 2002].

An important issue for many applications is the representation of preferences and reasoning about them. Researchers have investigated preferences among program rules [Schaub and Wang, 2001], among program literals [Sakama and Inoue, 2000], and context-dependent preferences among literals through the use of ordered disjunction [Brewka, 2002].

Representing and handling preferences in the formalism of logic programs is also the main topic of this paper. However, our approach differs from existing ones in an important aspect. Rather than specifying a preference relation among the rules or literals in a single logic program, we use two different programs. The first program, P_{gen}, is used to *generate* answer sets, that is, define the space of *acceptable* solutions. Context-dependent preferences are described in a second program, the preference program P_{pref}. These preferences are used to *compare* answer sets of P_{gen}, that is, to form a preference ordering of acceptable solutions. Intuitively, the rules of P_{gen} are hard constraints which specify conditions an answer set *must* satisfy; the rules of P_{pref} are soft constraints describing conditions under which one answer set is to be considered better than another.

The decoupling of answer-set generation and answer-set comparison has at least two advantages:

1. The method for comparing answer sets is independent of the type of the generating program P_{gen}. It may be any type of a logic program (for instance, normal, extended, disjunctive, involving cardinality or weight atoms), as long as it has a well-defined semantics given by a collection of sets of literals.
2. Preferences in P_{pref} (soft constraints) can be specified independently of P_{gen} (hard constraints). This makes preference elicitation easier since the task is broken into sepa-

*The authors acknowledge the support of DFG grant Computationale Dialektik BR 1817/1-5, Academy of Finland grant 53695 and NSF grant IIS-0097278, respectively.

rate and smaller subtasks. Moreover, it makes the overall setting better aligned with practical applications. Indeed, often what is possible is determined by external factors (available resources, constraints of the environment, design and engineering constraints for products) while what is preferred is described independently and by different agents (users, customers, groups of customers).

The rest of the paper is organized as follows. The next section introduces a new formalism, answer-set optimization programs, and defines their syntax and semantics. The subsequent section provides examples and additional observations on the basic formalism. In Section 4, we show how meta-preferences, that is, preferences on the preference rules, can be introduced and dealt with. Next, we discuss complexity of computational problems arising in the context of our formalism and propose implementation techniques. Our work is related to the approach proposed in [Boutilier *et al.*, 1999]. We investigate this relationship in Section 6. We conclude with additional discussion of other related work and possible extensions to the formalism presented here.

2 Optimization programs

We use two separate programs to describe the space of answer sets and preferences among them.

Definition 1 *Let A be set of atoms. A* preference program *over A is a finite set of rules of the form*

$$C_1 > \ldots > C_k \leftarrow a_1, \ldots, a_n, \mathrm{not}\ b_1, \ldots, \mathrm{not}\ b_m \qquad (1)$$

where the a_is and b_js are literals (expressions x and $\neg x$, where x is an atom in A), and the C_is are boolean combinations over A (to be defined below).

The rule intuitively reads: if an answer set S contains $a_1, \ldots, a_n$ and does not contain any of the literals $b_1, \ldots, b_m$ then C_1 is preferred over C_2, C_2 over C_3, etc. (we will give a precise semantics later in this section).

A *boolean combination* over A is a formula built of atoms in A by means of disjunction, conjunction, strong ($\neg$) and default (not) negation, with the restriction that strong negation is allowed to appear only in front of atoms, and default negation only in front of literals. For example, $a \wedge (b \vee \mathrm{not}\ \neg c)$ is a boolean combination, whereas $\neg(a \vee b)$ is not. Using boolean combinations rather than, say, literals in the heads of preference rules gives us additional expressiveness. Using conjunction we can express that certain combinations of properties are preferred over other combinations. Disjunction allows us to express that certain options are equally preferred. For instance, the rule $a > (b \vee c) > d \leftarrow f$ says that in case of f the best option is a; b and c are equally preferred second best options, and d is the least preferred option. Finally, we can use expressions like $a > b > c > (\mathrm{not}\ a \wedge \mathrm{not}\ b \wedge \mathrm{not}\ c)$ if we prefer to have one of properties a, b, c over not having any of them.

Definition 2 *Let S be a set of literals. Satisfaction of a boolean combination C in S (denoted $S \models C$) is defined as:*

$S \models l$ *(l literal)*	*iff*	$l \in S$
$S \models \mathrm{not}\ l$ *(l literal)*	*iff*	$l \notin S$
$S \models C_1 \vee C_2$	*iff*	$S \models C_1$ *or* $S \models C_2$
$S \models C_1 \wedge C_2$	*iff*	$S \models C_1$ *and* $S \models C_2$.

We next define the notion of an optimization program.

Definition 3 *An* answer-set optimization *(ASO) program is a pair (P_{gen}, P_{pref}), where P_{gen} is a logic program called the* generating program, *and P_{pref} is a preference program.*

As we already mentioned earlier, the program P_{gen} used for generating answer-sets can be of *any type*. We only require that the semantics be given in terms of sets of literals, or *answer sets*, that are associated with programs.

The key question is: how does the preference program P_{pref} determine a preference ordering on the answer sets described by the generating program P_{gen}? Let us consider an answer set S and a rule of the form (1). Given a set S of literals, three different situations are possible:

1. the body of r is not satisfied in S, that is, $a_i \notin S$ for some $i \in \{1, \ldots, n\}$, or some $b_j \in S$, for $j \in \{1, \ldots, m\}$
2. the body of r is satisfied in S and none of the C_is is satisfied in S
3. the body of r is satisfied in S and at least one C_i is satisfied in S.

In the case (1), the rule r is *irrelevant* to S because the rule does not apply. The case (2) is more subtle: the rule lists preferences among several options, yet none of the options holds. We consider this as *another* kind of irrelevance. Let us assume that a rule states that *red* is better than *green*, and S contains $blue$ and no other color. In this case the preference of *red* over *green* appears irrelevant to S since S does not mention these two colors at all. In the case (3), the preference expressed in the rule is satisfied to some degree (as at least one C_i holds in S). Thus, we define the *satisfaction degree* of r in S, $v_S(r)$, by setting $v_S(r) = I$, if r is irrelevant to S, and $v_S(r) = \min\{i : S \models C_i\}$, otherwise.

Concerning the relationship between I and the other satisfaction degrees, two viable options seem to exist. We can consider I as *incomparable* to other values, based on the view that "irrelevance" cannot be compared to *proper* satisfaction degrees. According to this view, selecting $blue$ in the example above is neither better nor worse than selecting $green$ or red. But one can also argue that $green$ violates preferences whereas $blue$ does not, and that $blue$ is thus preferable to $green$. We will adopt this latter view here and use the following *preorder* $\geq$ to compare satisfaction degrees (Fig. 1):

$$\begin{array}{c} 1, I \\ | \\ 2 \\ | \\ \ldots \end{array}$$

Fig.1: The preorder on satisfaction degrees

The values I and 1 are regarded *equally good* ($1 \geq I$ and $I \geq 1$) and better than all others. In addition, for each x, we have $x \geq x$. We write $x > y$ if x is *strictly* better than y.

Definition 4 *Let $P_{pref} = \{r_1, \ldots, r_n\}$ be a preference program and let S be an answer set. We say that S induces a* satisfaction vector $V_S = (v_S(r_1), \ldots, v_S(r_n))$.

We extend the preorder on satisfaction degrees to preorders on satisfaction vectors and answer sets as follows:

Definition 5 *Let S_1 and S_2 be answer sets. We write $V_{S_1} \geq V_{S_2}$ if $v_{S_1}(r_i) \geq v_{S_2}(r_i)$, for every $i \in \{1, \ldots, n\}$. We write $V_{S_1} > V_{S_2}$ if $V_{S_1} \geq V_{S_2}$ and for some $i \in \{1, \ldots, n\}$, $v_{S_1}(r_i) > v_{S_2}(r_i)$. In these cases we also write $S_1 \geq S_2$ and $S_1 > S_2$, respectively.*

We refer to this ordering as the *preference satisfaction ordering* (or ps-ordering, for short).

Definition 6 *A set of literals S is an* optimal *model of an ASO program (P_{gen}, P_{pref}) if S is an answer set of P_{gen} and there is no answer set S' of P_{gen} such that $S' > S$.*

3 Examples

The prototypical application for ASO programs are configuration problems where P_{gen} describes possible configurations and P_{pref} preferences among them. We use a dinner example similar to the one discussed in [Brewka, 2002]. It is convenient to use programs with cardinality constraints to generate answer sets. Such programs allow for the use of special atoms of the form $n\{a_1, \ldots, a_k\}m$, where the a_i are literals, to represent: at least n and at most m of the a_i are true. Although rules built from cardinality constraints can, in principle, be replaced by sets of rules without such constraints, they make problem specifications much more concise and readable. For the precise definitions we refer the reader to [Simons *et al.*, 2002].

Let us assume that P_{gen} consists of the rules:

$1\{soup, salad\}1$
$1\{beef, fish\}1$
$1\{pie, ice\text{-}cream\}1$
$1\{red, white, beer\}1.$

Each of these constraints enforces the selection of exactly *one* of the items it lists into an answer set. Thus, P_{gen} generates the space of $2 \times 2 \times 2 \times 3 = 24$ answer sets. Let us assume that P_{pref} is:

$white > red > beer \leftarrow fish$
$red \vee beer > white \leftarrow beef$
$pie > ice\text{-}cream \leftarrow beer.$

This preference program designates as non-preferred all answer sets containing $fish$ but not $white$, all answer sets containing $beef$ but not red or $beer$, and all answer sets containing $beer$ and not pie. We note that the objective is not to have these answer sets *eliminated*, which could be accomplished simply by adding the three constraints

$\leftarrow fish, \text{not } white$
$\leftarrow beef, \text{not } red, \text{not } beer$
$\leftarrow beer, \text{not } pie.$

to the original program. The role of the preference program is to define *soft constraints* and, moreover, to do so independently of the generating program. For instance, if we later learn that constraints $F = \{\leftarrow red;\ \leftarrow white;\ \leftarrow pie\}$ need to be included in P_{gen}, the additional constraints would lead to inconsistency. This does not happen in our approach. The answer sets generated by the extended program are:

$S_1 = \{ice\text{-}cream, beer, beef, soup\}$
$S_2 = \{ice\text{-}cream, beer, beef, salad\}$
$S_3 = \{ice\text{-}cream, beer, fish, soup\}$
$S_4 = \{ice\text{-}cream, beer, fish, salad\}$

Their satisfaction vectors are $V_1 = (I, 1, 2)$, $V_2 = (I, 1, 2)$, $V_3 = (3, I, 2)$, and $V_4 = (3, I, 2)$, respectively. Thus, S_1 and S_2 are equally good and are maximally preferred in the presence of F to S_3 and S_4, the latter two answer sets being also equally good.

4 Meta-preferences

The notion of optimality underlying our approach is somewhat weak. In general, many optimal answer sets may exist, and one often wants additional means to express that one preference (that is, one rule in the program P_{pref}) is more important than another. Here is a generalization of ASO programs where it is possible to express such meta-preferences:

Definition 7 *A* ranked *ASO program P is a sequence*

$$(P_{gen}, P^1_{pref}, \ldots, P^n_{pref})$$

consisting of a generating program P_{gen} and a sequence of pairwise disjoint preference programs P^i_{pref}.
The rank *of a rule $r \in P^1_{pref} \cup \ldots \cup P^n_{pref}$, denoted $rank(r)$, is the unique integer i for which $r \in P^i_{pref}$.*

Intuitively, preference rules with lower rank are preferred over preference rules with higher rank. We can now modify the definition of preference among answer sets by taking preferences among rules into account:

Definition 8 *Let $P = (P_{gen}, P^1_{pref}, \ldots, P^n_{pref})$ be a ranked ASO program. Let S_1 and S_2 be answer sets of P_{gen}. We define $S_1 \geq_{rank} S_2$ if for every preference rule r' such that $v_{S_1}(r') \geq v_{S_2}(r')$* does not hold, *there is a rule r'' such that $rank(r'') < rank(r')$ and $v_{S_1}(r'') > v_{S_2}(r'')$.*

Clearly, the preorder $\geq_{rank}$ extends (is stronger than) the preorder $\geq$. It is also easy to see that $\geq_{rank}$-optimal answer sets can be obtained in the following way: select all answer sets optimal wrt P_1, among those pick the ones optimal wrt P_2 and so on.

A further generalization of ranked to partially ordered ASO programs is straightforward and not presented here for lack of space.[1]

In some cases a natural ordering of the preference rules can be derived from the structure of the preference program. For each preference program P we define its dependency graph $G(P)$ as follows. The atoms appearing in P form the vertex set of $G(P)$. There is a directed edge from a vertex b to a vertex a in $G(P)$ if there is a rule r in P such that a appears in the head of r and b appears in the body of r.

If the graph $G(P)$ is acyclic, there is a natural ranking of its atoms. Namely, we define the rank of an atom a, $rank(a)$, recursively as follows: $rank(a) = 0$ for every atom a that has no predecessors in $G(P)$; otherwise, we define $rank(a)$

[1] Another possible extension concerns the setting of weighted preference rules. To compare answer sets one could use weighted sums of the violation degrees of preference rules.

as the maximum of the ranks of all predecessors of a in $G(P)$ incremented by 1.

The ranking of atoms implies the ranking of rules. Namely, we define the rank of a preference rule r, $rank(r)$, as the maximum rank of an atom appearing in the head of r.

We call preference programs with acyclic dependency graphs *acyclic preference programs*. They are important for two reasons:

1. They commonly appear in practice as preferences are often described according to some partial order on features defining answer sets, with some features being more important than others. For instance, in the dinner example, most users will start with the preferences concerning the main course. They could then condition their preferences concerning the appetizer and the beverage on the choice of the main course. And, finally, they may describe their preferences concerning the dessert based on earlier choices of the main course, appetizer and beverage.
2. As we just demonstrated, in the case when $G(P)$ is acyclic, there exists a natural ranking on the rules. This ranking is implied by the program itself and allows us to strengthen our ordering relation on answer sets, as described above in Definition 8. We call the resulting ordering determined by an acyclic preference program — the *canonical ps-ordering*.

5 Complexity and implementation

The complexity of ASO programs depends on the class of generating programs. To simplify the treatment we consider here only generating programs where deciding existence of an answer set is **NP**-complete. This class of programs includes ground normal programs (possibly extended with strong negation or weight and cardinality constraints) [Simons *et al.*, 2002].

The following two results indicate that allowing preferences adds an extra layer of complexity.

Theorem 1 *Let $P = (P_{gen}, P_{pref})$ be an ASO program and S an answer set of P_{gen}. Then deciding whether S is optimal is* **coNP**-*complete.*

Theorem 2 *Given an ASO program P and a literal l deciding whether there is an optimal answer set S such that $l \in S$ is $\mathbf{\Sigma_2^P}$-complete.*

The complexity results imply that (unless the polynomial hierarchy collapses) preference rules cannot be translated to generating rules in polynomial time, i.e., the problem of finding an optimal answer set for an ASO program cannot be mapped in polynomial time to a problem of finding an answer set of a program obtained by translating the ASO program to a set of generating rules only.

However, in [Brewka *et al.*, 2002] an implementation technique for computing optimal answer sets for logic programs with ordered disjunction on top of a standard answer set prover has been developed. A similar technique has earlier been used in [Janhunen *et al.*, 2000] to compute stable models of disjunctive logic programs using *Smodels*. The computation is based on a tester program that takes as input an answer set and generates a strictly better one if such an answer set exists. The computation starts with an arbitrary answer set generated by the generator. This answer set is given to the tester program. If the tester fails to generate a strictly better one, we have found an optimal answer set and we are done. If a strictly better answer set is discovered, the tester is run with the new answer set as input. This continues until an optimal answer set is reached.

This technique can be adapted for computing optimal answer sets of an ASO program (P_{gen}, P_{pref}) by choosing a suitable tester program P_T for a given answer set S_0. The tester program P_T is constructed by adding the following rules to the generator program P_{gen}:[2]

1. For each preference $r \in P_{pref}$ include a fact $v_0(r, d) \leftarrow$ where d is the satisfaction degree of r in S_0.
2. Include rules

$$\begin{array}{rcl} & \leftarrow & \text{not } better \\ better & \leftarrow & v_0(R, V_0), v_1(R, V_1), \\ & & geq(V_1, V_0), \text{not } geq(V_0, V_1) \\ & \leftarrow & v_0(R, V_0), v_1(R, V_1), \text{not } geq(V_1, V_0). \end{array}$$

3. Add facts $geq(d_1, d_2) \leftarrow$ giving the preorder $\geq$ on the set $\{irr, 1, 2, \ldots\}$.
4. For each Boolean combination C_i in P_{pref} (and its non-atomic subexpressions) introduce a new atom c_i and add rules capturing the conditions under which the expression is satisfied. For example, if C_i is a disjunction $C_h \vee C_l$, then add rules $c_i \leftarrow c_h$ and $c_i \leftarrow c_l$.
5. For each preference $r \in P_{pref}$ of the form (1) add rules

$$\begin{array}{rcl} body(r) & \leftarrow & a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m \\ heads(r) & \leftarrow & c_i \qquad \text{(for each } C_i\text{)} \\ v_1(r, irr) & \leftarrow & \text{not } body(r) \\ v_1(r, irr) & \leftarrow & \text{not } heads(r), body(r) \\ v_1(r, 1) & \leftarrow & c_1, body(r) \\ v_1(r, 2) & \leftarrow & \text{not } v_1(r, 1), c_2, body(r) \\ & \ldots & \\ v_1(r, k) & \leftarrow & \text{not } v_1(r, 1), \ldots, \text{not } v_1(r, k-1), \\ & & c_k, body(r). \end{array}$$

If P_T has an answer set S, then S restricted to the original language of P_{gen} is an answer set for P_{gen} that is strictly preferred to S_0, and if P_T has no answer set, no such answer set exists.

6 Relationship to CP-networks

An important approach to the problem of eliciting and approximating preferences is that of CP-networks proposed and developed in [Boutilier *et al.*, 1999]. We will now review this approach and show how it relates to our work.

The approach of CP-networks is concerned with comparing vectors of feature values that we call *configurations*. Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a set of features (attributes). For each feature A_i, let D_i be its domain, that is, a finite and non-empty set of *values* or *selections* for A_i. We assume that all domains are pairwise disjoint. A *configuration* is a k-tuple $(v_1, \ldots, v_k)$ such that $v_i \in D_i$, $1 \leq i \leq k$.

[2] We follow the Prolog convention that terms starting with capital letters are variables and write irr rather than I to avoid confusion.

The user prefers some configurations to others. Since the number of configurations is, in general, exponential in $|\cup_{i=1}^{k} D_i|$, it may be impractical to directly elicit and store the user's preference ordering on the set of all configurations. Thus, the task is to identify partial (and, in some sense, basic) information about the user's preferences, and develop ways to approximate the preference ordering implied by this partial information. The formalism of CP-networks [Boutilier *et al.*, 1999] is an approach to accomplish that.

A CP*-network* over a set of features $\mathcal{A}$ is a pair $(G, \mathcal{P})$, where G is a directed graph whose vertices are features from $\mathcal{A}$. The edges of the graph G determine dependencies among features: preferences for a value for a feature A depend only on values selected for the parents of A in the network. Thus, for each feature A and for each selection of values for the parent features for A, a CP-network specifies a total ordering relation on the domain of A. All these total orderings form the component $\mathcal{P}$ of a CP-network[3].

The information contained in a CP-network implies preferences among configurations. A configuration V is *one-step preferred* to a configuration W if for some feature A (given a configuration U, by $U(A)$ we denote the value from the domain of A selected to U):

1. $V(B) = W(B)$ for every feature $B \in \mathcal{A} \setminus \{A\}$ (that is, for every feature other than A), and
2. $V(A)$ is strictly preferred to $W(A)$ in the ordering of the domain D of A specified by the network for the selection for the parents of A as given by V (or, equivalently, by W).

The configuration V is CP*-preferred* to W if V can be obtained from W by a sequence of one-step improvements. It is easy to see that for acyclic CP-networks (acyclic networks seem to arise in most situations occurring in practice) this ordering is indeed a partial ordering; we call it a CP-ordering[4].

We will now show that the information represented by a CP-network N (with features $\mathcal{A} = \{A_1, \ldots, A_k\}$) can be represented by means of ASO programs. First, we specify the space of all configurations as answer sets to the program $P_{gen}(N)$ that, for each feature $A_i \in \mathcal{A}$ $(1 \leq i \leq k)$ contains the following rule (we assume here that $D_i = \{d_1, \ldots, d_n\}$):

$$1\{d_1, \ldots, d_n\}1.$$

As already discussed in Section 3, such rule enforces the selection of exactly one of the listed values to an answer set. Thus, answer sets of the program $P_{gen}(N)$ are precisely the configurations of N.

To specify preferences, we proceed as follows. Let A be a feature and let $\mathcal{B} = \{B_1, \ldots, B_r\}$ be the set of its parents in the CP-network. For every selection $(v_1, \ldots, v_r)$ of values of features $B_1, \ldots, B_r$, respectively, the network specifies an ordering, say $d_1 > \ldots > d_n$, on the domain $D = \{d_1, \ldots, d_n\}$ of A. We represent that fact by including a rule

$$d_1 > \ldots > d_n \leftarrow v_1, \ldots, v_r$$

in the preference program $P_{pref}(N)$.

Thus, at the level of syntax, our approach extends that of CP-nets. In particular, [Boutilier *et al.*, 1999] is concerned only with one fixed space of all configurations that contain for every feature exactly one value from its domain. In contrast, in our approach we have a substantial flexibility in defining answer sets by varying the generator program.

In addition, our approach is more robust when user preferences are inconsistent, the situation that often occurs in practice. For instance, the user may specify a preference rule $a > b \leftarrow body_1$ and $b > a \leftarrow body_2$ unaware of situations where both $body_1$ and $body_2$ are satisfied. Such inconsistencies cannot be modeled by CP-networks. In our approach they can with the effect that one of the preference rules will be violated whenever both bodies are true.

The key question is that of the relationship between the semantics of the two approaches. First, we observe that they are different. Let us consider the following example from [Boutilier *et al.*, 1999]. There are two features A and B with values a, a', and b, b', respectively. Feature B depends on feature A and the preferences are specified as follows (we give them in the notation of ASO programs):

$a > a'$
$b > b' \leftarrow a$
$b' > b \leftarrow a'$.

In the CP-ordering, $S_1 = \{a, b'\}$ is preferred over $S_2 = \{a', b'\}$ as there is a one-step improvement leading from S_2 to S_1. On the other hand, none of S_1 and S_2 is strictly preferred over the other in the ps-ordering (the corresponding satisfaction-degree vectors are: $(1, 2, I)$ and $(2, I, 1)$, respectively). The reason is that the meaning of the rules in the two approaches is slightly different. In the CP-net approach, $x > x' \leftarrow body$ means (x and x' form the domain of a feature X): among all answer sets satisfying $body$, answer set S_1 is better than S_2 if both agree on all features except X and S_1 makes x true and S_2 makes x' true. In our approach $x > x' \leftarrow body$ is more like a soft constraint expressing: whenever $body$ is true there is a reason to prefer x over x'. Here, among the answer sets satisfying $body$, an answer set S_1 is preferred over S_2 if $x \in S_1$, $x' \in S_2$, and the other rules of the program are satisfied at least as well in S_1 as in S_2. In a sense, what is different is the interpretation of the ceteris paribus — other things being equal — phrase, more precisely, the interpretation of what is meant by the "other things" that have to be equal. In the CP-network approach the "other things" are the selections of values for other features which have to be the same. In our approach it is the degree of satisfaction of the other rules in the program.

In general, our basic approach yields a weaker ordering. Namely, we have the following theorem.

Theorem 3 *Let N be an acyclic CP-network and let V and W be two configurations. If V is strictly preferred to W under the ps-ordering determined by the ASO program $(P_{gen}(N), P_{pref}(N))$, then V is strictly preferred to W under the CP-ordering implied by N.*

[3] The restriction that domains be totally ordered can be dropped. We adopt it, following [Boutilier *et al.*, 1999] to keep the discussion simple.

[4] This ordering captures the intuition *ceteris paribus* or *all other things being equal*, the aspect that is discussed in detail in [Boutilier *et al.*, 1999].

It is easy to see that the representation of an acyclic CP-network in our syntax results in an acyclic preference program. In this case, we have the following result.

Theorem 4 *Let N be an acyclic CP-network and let V and W be two configurations. If V is strictly preferred to W under the CP-ordering determined by the CP-net N, then V is strictly preferred to W under the canonical ps-ordering determined by the ASO program $(P_{gen}(N), P_{pref}(N))$.*

These two theorems show that we can approximate the CP-ordering by means of orderings implied by our approach. It is important as the relationship between two answer sets with respect to (ranked) ps-ordering can be verified in polynomial time, while it is not known whether polynomial time algorithms exist in the case of the CP-ordering (in fact, there are examples of configurations that require an exponentially long chain of one-step improvements to demonstrate that one is preferred to the other).

7 Further related work and conclusions

In this paper we have introduced ASO programs which combine a generating program with a preference program. The combination allows us to specify possible solutions of a problem (answer sets) together with preferences among specific aspects of solutions. The preference program orders the answer sets according to the satisfaction degrees of its preference rules.

Numerous papers which introduce preferences to logic programming exist in the literature. For an overview of some of them see for instance [Schaub and Wang, 2001]. Closest to ours is the proposal in [Brewka, 2002; Brewka *et al.*, 2002]. Here, preferences are expressed through ordered disjunction. Ordered disjunction is a non-commutative kind of disjunction which gives preference to the first disjunct over the second. The approach presented here differs from ordered disjunction in the following respects:

1. generation and comparison of answer sets are separated,
2. preference handling works independently of the type of program used for answer set generation,
3. more general preferences can be stated due to our use of boolean combinations.

In fact, it is not difficult to show that logic programs with ordered disjunction ($LPOD$s) are a special case of our approach. Let P be an $LPOD$, P_{gen} a logic program possessing the same answer sets as P, and let

$$P_{pref} = \{c_1 > \ldots > c_n \leftarrow body : c_1 \times \ldots \times c_n \leftarrow body \in P\}$$

The preferred answer sets of P under the Pareto criterion (cf. [Brewka *et al.*, 2002]) and those of the corresponding ASO program coincide. We can also capture the inclusion based preference criterion from [Brewka *et al.*, 2002] using appropriate ranked ASO programs. For each rule $c_1 \times \ldots \times c_n \leftarrow body \in P$ and for each $i \leq n$ we have to include $c_1 \vee \ldots \vee c_i > \top \leftarrow body$ in the preference program P^i_{pref}. This gives us exactly the inclusion-preferred answer sets of P.

In future work we plan to investigate further generalizations of our approach where rule heads may contain arbitrary partial orders on boolean combinations. We intend also to study in depth the distinction between options being "equally good" and "incomparable". Finally, we will study extensions of the one-step improvement concept from CP-networks to the setting of general ASO programs.

References

[Baral, 2003] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003. ISBN 0521818028.

[Boutilier *et al.*, 1999] C. Boutilier, R.I. Brafman, H.H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *Proc. UAI-99*, 1999.

[Brewka *et al.*, 2002] G. Brewka, I. Niemelä, and T. Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *Proc. JELIA 2002*. Springer Verlag, 2002.

[Brewka, 2002] G. Brewka. Logic programming with ordered disjunction. In *Proc. AAAI-02*. Morgan Kaufmann, 2002.

[Eiter *et al.*, 1998] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system dlv: Progress report, comparisons and benchmarks. In *Proc. Principles of Knowledge Representation and Reasoning, KR-98*. Morgan Kaufmann, 1998.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Janhunen *et al.*, 2000] T. Janhunen, I. Niemelä, P. Simons, and J.-H. You. Unfolding partiality and disjunctions in stable model semantics. In *Proc. Principles of Knowledge Representation and Reasoning, KR-00*, pages 411–419. Morgan Kaufmann, 2000.

[Lifschitz, 2002] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence Journal*, 138(1-2):39–54, 2002.

[Niemelä and Simons, 1997] I. Niemelä and P. Simons. Efficient implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*. Springer Verlag, 1997.

[Sakama and Inoue, 2000] C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.

[Schaub and Wang, 2001] T. Schaub and K. Wang. A comparative study of logic programs with preference. In *Proc. IJCAI-01*, 2001.

[Simons *et al.*, 2002] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[Soininen, 2000] T. Soininen. *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks*. PhD thesis, Helsinki University of Technology, Finland, 2000.