Answer Sets: From Constraint Programming Towards Qualitative Optimization

Gerhard Brewka

University of Leipzig
Dept. of Computer Science
Augustusplatz 10-11
04109 Leipzig, Germany
brewka@informatik.uni-leipzig.de

Abstract. One of the major reasons for the success of answer set programming in recent years was the shift from a theorem proving to a constraint programming view: problems are represented such that stable models, respectively answer sets, rather than theorems correspond to solutions. This shift in perspective proved extremely fruitful in many areas. We believe that going one step further from a "hard" to a "soft" constraint programming paradigm, or, in other words, to a paradigm of qualitative optimization, will prove equally fruitful. In this paper we try to support this claim by showing that several generic problems in logic based problem solving can be understood as qualitative optimization problems, and that these problems have simple and elegant formulations given adequate optimization constructs in the knowledge representation language.

1 Introduction

Answer sets [17], originally invented to define the semantics of (extended) logic programs with default negation, have proven to be extremely useful for solving a large variety of AI problems. Two important developments were essential for this success:

1. the development of highly efficient answer-set provers, the most advanced among them being *Smodels* [21] and *dlv* [12];
2. a shift from a theorem proving to a constraint programming perspective [20],[19].

Let us focus on the latter here. It turned out that many problems, for instance 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 [18, 28, 1, 13]. This view of logic programs as constraints on the sets of literals which may count as solutions has led to a new problem solving paradigm called answer set programming (ASP).

The major goal of this paper is to push this shift in perspective one step further: from a constraint programming paradigm to a paradigm of qualitative optimization. Many problems have natural formulations as optimization problems, and many problems which can be represented in a "hard" constraint programming paradigm have fruitful, more flexible refinements as optimization problems.

Consider planning as a typical example. We know how to represent planning problems as ASP problems. Moving from a constraint programming to an optimization perspective allows us to specify criteria by which we can rank plans according to their quality. This allows us to select good plans (or to generate suboptimal plans if there is no way to satisfy all requirements). An example for quantitative optimization is planning under action costs [15].

Our interest in optimization based on qualitative preferences stems from the fact that for a variety of applications numerical information is hard to obtain (preference elicitation is rather difficult) - and often turns out to be unnecessary. This paper contains several examples of this kind.

Of course, the use of optimization techniques in answer set programing is not new. There is a large body of work on preferred answer sets, see for instance [5, 26] and the references in these papers. Also some of the existing solvers have (numerical) optimization facilities: *Smodels* with weight constraints has *maximize* and *minimize* statements operating on weights of atoms in answer sets [27]. An interesting application of these constructs to modeling auctions can be found in [2]. *dlv* has weak constraints [8] of the form

$$\leftarrow a_1, \ldots a_n, not\ b_1, \ldots , not\ b_m. [w : l]$$

where w is a numerical penalty and l is a priority level. For each priority level, the sum of the penalties of all violated constraints (i.e., constraints whose bodies are satisfied) is computed. The answer sets with minimal overall penalty in one level are compared based on the overall penalties of the next level, etc. Such constaints were used, for instance, to implement planning under action costs as described in [15] and for knowledge based information-site selection [14].

However, a systematic treatment of answer set programming as *qualitative optimization* is still lacking. This paper is meant as a step into this direction. The major goal is to substantiate - by means of several generic examples - the claim that optimization techniques, and in particular qualitative techniques, are highly fruitful in a large number of applications.

This paper will not introduce any new formalism. Instead, it will use existing formalisms, co-developed by the author, to demonstrate how these formalisms can be used fruitfully for a variety of problems. We will use logic programs with ordered disjunction (LPODs) as described in [4] and [7], as well as the framework of optimization programs as described in [6]. Both have their merits, the second being more modular and general. However, in some cases the former admits more concise representations, as we will see.

The problems we describe in this paper are generic in the sense that they may arise in a number of different applications. We will first discuss the use of optimization techniques for abduction, that is, the generation of explanations for observed behaviour, and diagnosis. The major goal here is to give an example showing that it is indeed extremely helpful to have optimization techniques available as part of the semantics of the underlying formalism. We will give straightforward translations of abduction and also of consistency based diagnosis techniques to LPODs. Our translations are influenced by [13]. However, due to the availability of optimization constructs in the representation language our translations are considerably simpler than those used for the *dlv* diagnosis frontend.

We next discuss the problem of solution coherence. Consider a domain where solutions need to be computed ahead of time, but later on new constraints may have to be taken into account. A typical example (which was discussed at the Banff Workshop on Constraint Programming, Belief Revision and Combinatorial Optimization, May 03) is the scheduling of meetings in an organization. Persons who need to attend a particular meeting must be informed about this fact early enough. On the other hand, it may always happen that, due to unforeseen circumstances, new "hard" constraints arise. Now assume an automated scheduling system based on answer set programming were to recompute a new solution based on the new set of constraints from scratch. This could lead to a new solution of the scheduling problem with completely different time slots for a large number of meetings. This is certainly unwanted: what we would like to have is a new solution which is as close as possible to the original solution. This is obviously a qualitative optimization problem and we will show how this problem can be handled using optimization programs.

Finally, we will show that inconsistency handling can be viewed as an optimization problem. The basic idea is to re-represent the original information in such a way that rules can be arbitrarily disregarded. We then use preferences expressing that rules should not be disregarded. We can go further and add meta preferences between these preferences to distinguish between rules which should not be disregarded with higher priority. We can also introduce contradiction removal rules [23] which are applied only in case an inconsistency arises.

From the discussion of the examples we derive a number of requirements for a general specification language for qualitative optimization. Such a language should allow us to express preferences of a very general kind together with different combination strategies, and it should be possible to apply different strategies to different parts of the preference program. We strongly believe that extending answer set programming in this direction will greatly increase its impact as a new declarative programming paradigm.

We assume the reader is familiar with the notion of answer sets [17]. A short introduction to LPODs and optimization programs is given in the next section.

2 LPODs and optimization programs

In this section we briefly recall the basic notions underlying the formalisms to be used in this paper, namely LPODs [4, 7] and optimization programs [6]. LPODs use ordered disjunction $\times$ in the head of rules to express preferences among literals in the head: the rule $r = A_1 \times \ldots \times A_n \leftarrow body$ says: if *body* is satisfied then some A_i must be in the answer set, most preferably A_1, if this is impossible then A_2, etc. Answer sets are defined through split programs [25] containing exactly one option for each of the original LPOD rules, where, for $k \leq n$, option r^k of the rule above is $A_k \leftarrow body, not\ A_1, \ldots, not\ A_{k-1}$.

An answer set S can satisfy rules like r to different degrees, where smaller degrees are better: if *body* is satisfied, then the satisfaction degree is the smallest index i such that $A_i \in S$. Otherwise, the rule is irrelevant. Since there is no reason to blame an answer set for not applying an inapplicable rule we also define the degree to be 1 in this case. The degree of r in S is denoted $deg_S(r)$.

Based on the satisfaction degrees of single rules a global preference ordering on answer sets is defined. This can be done through a number of different combination strategies. Let $S^k(P) = \{r \in P \mid deg_S(r) = k\}$. For instance, we can use one of the following conditions to define that S_1 is strictly preferred to S_2:

1. there is a rule satisfied better in S_1 than in S_2, and no rule is satisfied better in S_2 than in S_1 (Pareto);
2. at the smallest degree j such that $S_1^j(P) \neq S_2^j(P)$ we have $S_1^j(P) \supset S_2^j(P)$ (inclusion);
3. at the smallest degree j such that $|S_1^j(P)| \neq |S_2^j(P)|$ we have $|S_1^j(P)| > |S_2^j(P)|$ (cardinality);
4. the sum of the satisfaction degrees of all rules is smaller in S_1 than in S_2 (penalty sum).

Note that S_1 is inclusion preferred to S_2 whenever it is Pareto preferred to S_2, and cardinality preferred to S_2 whenever it is inclusion preferred to S_2. The penalty sum strategy can be used to model *dlv*'s weak constraints, provided all weights are integers: a weak constraint $\leftarrow body$ with weight n can be represented as $\bot \times \ldots \times \bot \times \top \leftarrow body$, where the ordered disjunction has n disjuncts.

In the LPOD approach the construction of answer sets is amalgamated with the expression of preferences. Optimization programs, on the other hand, separate the two consequently to allow for greater modularity, flexibility and generality. An optimization program is a pair (P_{gen}, P_{pref}) consisting of an arbitrary program P_{gen} used to generate answer sets, that is, P_{gen} can be any type of program as long as it produces sets of literals, and a preference program P_{pref}. The latter consists of rules of the form $C_1 > \ldots > C_n \leftarrow body$ where the C_i are boolean combinations of literals built from $\wedge, \vee, \neg$ and *not* . Again, if *body* is satisfied in an answer set S and some boolean combination C_i is satisfied, then the degree of the rule is the smallest such index. Otherwise the rule is irrelevant. Again we do not want to blame S for irrelevant rules and consider irrelevance as good a satisfaction degree as 1.

Again, the combination strategies described above - and many others - can be used to generate the global preference order. It is also possible to introduce meta preferences among the preference rules themselves by grouping them into subsets with different ranks. Preference rules with highest priority are considered first. Intuitively, rules with lower priority are only used to distinguish between answer sets which are of the same quality according to more preferred rules. The reader is referred to the original papers for more details.

In the rest of this paper the inclusion-based combination strategy, that is, strategy 2 in the list above, will be our default strategy. As we will see, subset minimality is useful in many cases. Note that inclusion and Pareto preference coincide whenever all preference statements (ordered disjunctions or heads of preference rules) speak about no more than two options. Whenever we need cardinality-based preference we will mention this explicitly.

3 Abduction and diagnosis

Abduction is the process of generating explanations for a set of observations. Most formal approaches to abduction are based on a fixed set of formulas H of possible explanations, a set of formulas K representing background knowledge, and a set of formulas O describing the observations to be explained [16, 24, 22, 9]. An explanation then is a minimal subset H' of H satisfying:

1. $H' \cup K$ is consistent and,
2. $H' \cup K \models O$.

For the context of answer set programming let us assume that K is a logic program, H and O are sets of literals, and $\models$ is the credulous inference relation under answer set semantics, that is, for the program K and observations O we have $K \models O$ iff there is an answer set S of K such that $O \subseteq S$.[1] An explanation for O can be computed via the following LPOD $P_{abd}(K, H, O)$:

$$K \cup \{\leftarrow not\ o \mid o \in O\} \cup \{\neg ass(h) \times ass(h) \mid h \in H\} \cup \{h \leftarrow ass(h) \mid h \in H\}.$$

Intuitively, $ass(h)$ reads: h is assumed. The use of constraints for eliminating answer sets where some observation is unexplained, and the use of a predicate like ass is taken from [13]. However, the use of ordered disjunction is new and makes the representation considerably simpler. We have the following proposition:

Proposition 1. *Let H and O be sets of literals and let K be a logic program. H' is an explanation for O given K and H iff there is a consistent answer set S of $P_{abd}(K, H, O)$ such that $H' = \{h \in H \mid ass(h) \in S\}$.*

Abduction has a direct application in diagnosis. Diagnostic systems are an important class of knowledge based systems. Their main task is to determine reasons for observed malfunctioning. In the literature 2 kinds of approaches are usually distinguished:

- *abductive diagnosis:*
  a model M of the system to be diagnosed describes the effects of possible failures (diseases in the medical domain). Given a set of potential failures F

[1] This view is basically taken from [13]

and a set of observations O, an abductive diagnosis is an explanation of O from F with background knowledge M. In other words, abductive diagnosis is a special case of abduction where the hypotheses are potential failures and the background knowledge is the description of the system to be diagnosed.
- *consistency based diagnosis:*
  here the model M describes the normal behaviour of the system, using abnormality predicates. A diagnosis is a minimal subset C' of the system components C such that assuming elements of C' to be abnormal and the others to be normal explains the observations. More formally, we are looking for a minimal subset C' of the components such that:
  1. $\{ab(c) \mid c \in C'\} \cup \{\neg ab(c) \mid c \in C \setminus C'\} \cup M$ is consistent and,
  2. $\{ab(c) \mid c \in C'\} \cup \{\neg ab(c) \mid c \in C \setminus C'\} \cup M \models O$.

Again we follow [13] in assuming that M is given as an (extended) logic program, F as a set of atoms, O as a set of literals, and that $\models$ is the credulous inference relation under answer set semantics, i.e. $P \models O$ says there is an answer set containing all literals in O. It is straightforward to express the conditions for both types of diagnosis using LPODs.

Abductive diagnosis is just a special case of abduction, and the program needed to compute diagnoses is just $P_{abd}(M, F, O)$ as described above.

For consistency based diagnosis we obtain the following program $P_{cd}(M, C, O)$:

$$M \cup \{\leftarrow \mathit{not}\ o \mid o \in O\} \cup \{\neg ab(c) \times ab(c) \mid c \in C\}.$$

Proposition 2. *Let C be a set of components, O a set of literals describing observations, and M a logic program describing the behavior of a system using the ab predicate. $C' \subseteq C$ is a consistency based diagnosis iff there is a consistent answer set S of $P_{cd}(M, C, O)$ such that $C' = \{c \mid ab(c) \in S\}$.*

Here is a small example due to [11]. Assume we have a battery connected in parallel to three bulbs. The program M consists of the facts:

$$\begin{array}{lll} battery(bat) & bulb(b_1) & connected(bat, b_1) \\ & bulb(b_2) & connected(bat, b_2) \\ & bulb(b_3) & connected(bat, b_3) \end{array}$$

together with the behaviour description:

$$\begin{array}{rcl} lit(X) & \leftarrow & bulb(X), \mathit{not}\ ab(X), \\ & & battery(Y), \mathit{not}\ ab(Y), connected(Y, X) \\ \neg ab(X) & \leftarrow & bulb(X), lit(X) \\ \neg ab(Y) & \leftarrow & bulb(X), lit(X), battery(Y), connected(Y, X) \end{array}$$

Assume the observations are $O = \{\neg lit(b_1), lit(b_3)\}$, that is, we have to add the constraints:

$$\begin{array}{l} \leftarrow \mathit{not}\ \neg lit(b_1) \\ \leftarrow \mathit{not}\ lit(b_3) \end{array}$$

Adding ordered disjunctions $\neg ab(c) \times ab(c)$ for $c \in \{bat, b_1, b_2, b_3\}$ we obtain a single preferred answer set containing $ab(b_1)$ and $\neg ab(c')$ for $c' \in \{bat, b_2, b_3\}$. The second answer set containing both $ab(b_1)$ and $ab(b_2)$ is non-preferred. The single diagnosis thus is b_1.

Eiter and colleagues also discuss single fault diagnosis [13]. Consistency based diagnosis can be restricted to the case where at most one component is faulty by adding the constraint $\leftarrow ab(X), ab(Y), X \neq Y$. However, in our case we can simply switch from an inclusion-based to a cardinality-based combination strategy if we want to minimize the number of components in a diagnosis. This has the advantage that we do not need to repeatedly replace the constraint with a weaker constraint allowing for more faults whenever a diagnosis with fewer faults does not exist.

It is illustrative to compare our simple formalizations with those in [13]. The lack of optimization facilities in the language used in that paper makes the translation of diagnosis problems to answer set programming considerably more difficult. There is a one page algorithm for the realization of subset minimality. We take this as support for our view that optimization facilities should be part of answer set programming languages.

4 Solution coherence

In answer set programming problems are specified as programs and problem solutions correspond to answer sets of these programs. Assume we have computed a solution S to a problem specified through program P. Now consider some modification P' of P. At least in certain contexts it appears to be inappropriate to compute a new solution from scratch, that is, some arbitrary answer set S' of P'. We often want the new solution S' to be a minimal modification of S.

An illustrative example is a meeting scheduling system where P describes meetings, required participants, time restrictions of participants etc. and where S describes the scheduled meetings. If for one of the required participants of a particular meeting a new time constraint c arises, then S may no longer be a valid solution of $P' = P \cup \{c\}$. It is obvious that in this example we are not looking for some arbitrary solution of P', but for a solution of P' which is *as close as possible to* S.

Given a logic programming language with optimization facilities it is straightforward to represent this. What we need is a distance measure between answer sets. Here is one such measure based on symmetric differences. Let L, L_1 and L_2 be sets of literals. We say L_1 is closer to L than L_2, denoted $L_1 \leq_L L_2$, iff

$$\{a \mid a \in L \setminus L_1 \text{ or } a \in L_1 \setminus L\} \subseteq \{a \mid a \in L \setminus L_2 \text{ or } a \in L_2 \setminus L\}.$$

Expressing that solutions should be as close as possible to the original solution S, according to this criterion, is a simple matter. Here is the corresponding preference program:

$$P_{pref(S)} = \{a > not\ a \mid a \in S\} \cup \{not\ a > a \mid a \notin S\}.$$

Proposition 3. *Let S be a set of literals, P a program. S_1 is a preferred answer set of the optimization program $(P, P_{pref(S)})$ iff it is an answer set of P maximally close to S.*

To be more specific, assume we have to schedule 3 meetings m_1, m_2, m_3. Moreover, there are 4 available non-overlapping time slots $s_1, \ldots, s_4$ and 4 persons $p_1, \ldots, p_4$. We use $slot(M, S)$ to express that the time slot for meeting M is S, $part(P, M)$ to express that person P needs to participate in meeting M, and $unav(P, S)$ to express that person P is unavailable at time slot S. Assume we have

$$\begin{array}{lll} part(p_1, m_1) & part(p_3, m_2) & unav(p_1, s_4) \\ part(p_2, m_1) & part(p_3, m_3) & unav(p_2, s_4) \\ part(p_2, m_2) & part(p_4, m_3) & unav(p_4, s_2) \end{array}$$

Using cardinality constraints [27] we can express that each meeting needs exactly one time slot:

$$1\{slot(M, S) : slot(S)\}1 \leftarrow meeting(M)$$

Furthermore, we need the following constraints expressing that participants have to be available and that one cannot attend different meetings at the same time:

$$\begin{array}{l} \leftarrow part(P, M), slot(M, S), unav(P, S) \\ \leftarrow part(P, M), part(P, M'), M \neq M', slot(M, S), slot(M', S) \end{array}$$

One of the answer sets of this program contains:

$$slot(m_1, s_1), slot(m_2, s_2), slot(m_3, s_3)$$

Now assume p_4 becomes unavailable at s_3:

$$unav(p_4, s_3)$$

The former solution is no longer valid. Among the new solutions we have:

$$\begin{array}{l} S_1 : slot(m_1, s_1), slot(m_2, s_2), slot(m_3, s_4) \\ S_2 : slot(m_1, s_2), slot(m_2, s_1), slot(m_3, s_4) \\ S_3 : slot(m_1, s_3), slot(m_2, s_2), slot(m_3, s_1) \end{array}$$

Clearly, S_1 is closer to the original solution than S_2. Indeed, our criterion is satisfied and our optimality program will prefer S_1 over S_2.

But what about S_1 and S_3? This time the symmetric difference between S and S_1 and that between S and S_3 are not in subset relation: the former contains $slot(m_3, s_4)$ which is not in the latter, the latter contains $slot(m_1, s_3)$ which is not in the former. Intuitively, we may prefer S_1 since it changes fewer meetings in terms of cardinalities. This can simply be achieved by switching from an inclusion-based to a cardinality-based combination rule for satisfaction degrees. S_1 is better than S_2 since the number of preference rules satisfied to degree 2 in S_3 is higher than the number of preference rules satisfied to degree 2 in S_1.

The specification of the corresponding distance measure is straightforward and therefore omitted.

Numerous further refinements can be thought of: we may determine closeness based on the set (or number) of people participating in meetings whose slot is changed rather than on the set (or number) of meetings. We may also use meta preferences and give higher preference to rules minimizing changes for, say professors, followed by rules minimizing changes for assistants, followed by rules minimizing changes for students. All these refinements are simple exercises if optimization constructs are part of the ASP formalism, and become clumsy if they are not.

5 Inconsistency handling

In classical logic there are two major ways of handling inconsistency: (1) one can weaken the set of premises, that is, disregard some of the premises entirely or replace them with weaker formulas such that the inconsistency disappears, or (2) one can weaken the inference relation to make sure that inconsistent premises will not lead to the derivation of arbitrary formulas. The former approach is often based on the notion of maximal consistent subsets of the premises, the latter uses paraconsistent logics which are often based on multi-valued logic.

In nonmonotonic reasoning and logic programming there is a third alternative: one can add information to the effect that a rule giving rise to inconsistency is blocked. A simple example is the program $\{p \leftarrow \mathit{not}\ p\}$ which can be made consistent[2] by adding the fact p.

An excellent overview of paraconsistent logic programming is [10]. We will stick here to the other approaches and show how optimization techniques can be used to make sure the effect of weakening or of adding new information is minimized. Let us start with a weakening approach.

Let P be a, possibly inconsistent, logic program. Let n be a function assigning unique names taken from a set N of atoms not appearing in P to each of the rules in P. We first replace P by the weakened program $weak_n(P)$:

$$
\begin{aligned}
&\{head \leftarrow body, r \mid head \leftarrow body \in P, n(head \leftarrow body) = r\} \cup \\
&\{r \leftarrow \mathit{not}\ \neg r \mid r \in N\} \cup \\
&\{\neg r \leftarrow \mathit{not}\ r \mid r \in N\}
\end{aligned}
$$

The new program indeed is very weak. Basically, the new representation allows us to switch rules on and off arbitrarily. Let $Lit(P)$ denote the set of literals built from atoms in P. We have the following result:

Proposition 4. *If S is an answer set of $weak_n(P)$ then $S \cap Lit(P)$ is an answer set of some subset of P. Vice versa, if S is an answer set of a subset of P, then there is an answer set S' of $weak_n(P)$ such that $S' \cap Lit(P) = S$.*

[2] We use the term inconsistent both for programs having an inconsistent answer set and for programs having no answer set at all. Some authors call the latter incoherent, but the distinction will not matter here.

Existence of a consistent answer set follows directly from this proposition. Of course, we do not want to consider arbitrary subsets of the original program but maximal ones. Using optimization programs, this can be achieved by using the set of preference rules:

$$\{r > \neg r \mid r \in N\}.$$

Preferred answer sets now correspond, up to literals built from atoms in N, to answer sets of maximal consistent subsets of the original program P.

Obviously this is still a very simple strategy. Optimization programs offer the possibility to express preferences among the rules to be taken into account. For instance, assume r_1 is a domain constraint and r_2 an observation made by a potentially unreliable observer. One would certainly consider r_1 as more entrenched (the terminology is taken form the area of belief revision) than r_2. We can model this simply by adding a statement of the form $r_1 > r_2$ to our preference program. We can also express context dependent preferences among rules. For instance, if we have meta information about the sources of rules, say $source(r_1, Peter)$, $source(r_2, John)$, together with information about the reliability of sources, say $more{-}reliable(Peter, John)$, then we can use a rule of the form

$$R_1 > R_2 \leftarrow source(R_1, S_1), source(R_2, S_2), more{-}reliable(S_1, S_2)$$

to give preference to rules with more reliable sources. Similarly, if we have temporal meta-information about when a rule was obtained, we may have rules expressing that information obtained earlier in time is less preferred than information obtained later. Preferences of this kind are common in the area of belief revision. In any case, given adequate meta information, optimization facilities allow us to represent declaratively our strategy of handling inconsistencies.

As mentioned in the beginning of this section consistency in logic programming can sometimes be restored by adding rather than disregarding rules.[3] Assume a set of rules R to be used for this purpose is given together with the original program P. Borrowing terminology from [23] we call them contradiction removal rules. Again we build the weakened program $weak_n(P \cup R)$ using a naming function n. We use N_P to denote the names of rules in P and N_R to denote the names of rules in R.

The major difference now lies in the preference program. Intuitively, we have to minimize the set of rules from P which are disregarded and to maximize the set of rules from R which are disregarded. It is a simple matter to express this using the preference program:

$$\{r > \neg r \mid r \in N_P\} \cup \{\neg r > r \mid r \in N_R\}.$$

Since it is the purpose of contradiction removal rules to be used whenever inconsistencies arise we may want to add meta preferences. If we give rules in the

[3] This approach is also pursued in CR-Prolog [3]. The method presented here differs from CR-Prolog in two respects: we can add as well as disregard rules, and preference programs allow us to describe flexible inconsistency handling strategies declaratively.

first of the two sets above preference over the rules in the second set, then the contradiction is removed by removal rules alone whenever this is possible. Only if the removal rules are insufficient, information from the original program is disregarded.

The reader will have observed that we used preference rules containing $>$ rather than LPOD rules containing $\times$ in this and the preceding section, whereas Sect. 3 was based on LPODs. This is not incidental. In Sect. 3 all preferences were about available options which needed to be generated anyway. The use of $\times$ is highly convenient in such cases. Here we need to be able to express, say, that a domain constraint r_1 is preferred over an observation r_2 *without* assuming that each answer set contains one of these two rules. The ability to express preferences without generating answer sets containing one of the options appears crucial here.

6 Requirements for a qualitative optimization language

The examples in the last three sections have shown that qualitative optimization is an important issue for a variety of problems. In further work we are planning to develop a language for stating problems of this kind. From the discussion of our examples we derive the following requirements for such a language:

1. *modularity*: we want to be able to express optimization policies independently of the type of program used for answer set generation. This is a feature already realized in optimization programs and should remain a feature of any future extension of the underlying language.
2. *generality*: both ordered disjunction and preference statements based on $>$ as in optimization programs have their merits. The latter can express preferences based on formulas satisfied in answer sets, even if corresponding generating programs are not available. The former are highly convenient whenever preferences are expressed among options which have to be generated anyway, as it was the case in abduction and diagnosis. Therefore, both should be available in a general language.
   We also believe that a further generalization to partial orders will be useful. Although optimization programs allow us to rank classes of answer sets depending on the boolean combinations they satisfy, these classes are still totally ordered by each preference rule. There will be relevant cases where a partial order is all we have.
3. *multiple combination strategies*: already the few examples discussed in this paper show that different strategies of combining preferences based on single rules to a global preference order on answer sets are necessary. We discussed in particular inclusion-based and cardinality-based combinations, but others may turn out to be useful as well.
   Since different strategies are useful the user should also have the possibility to combine different stategies in the same preference program. What we have in mind are, possibly nested, blocks of preference rules such that each block has its own user defined combination strategy.

4. *meta preferences*: in the preceding sections we have already discussed several useful applications of meta-preferences among preference rules. For optimization programs meta-preferences and their semantics were defined in [6] based on a partition of the rules. If a general language for qualitative optimization has some kind of a block structure, as indicated in item 3. above, then the blocks are adequate structures to define meta preferences on. A partial preference order on - possibly nested - blocks of preference rules would be a useful generalization of the meta preferences we have right now.

7 Conclusions

In this paper we have demonstrated that optimization constructs beyond those available in current answer set solvers can be highly fruitful for numerous problem solving tasks. To support our claim we discussed several generic problems: abduction and diagnosis, solution coherence and inconsistency handling. Each of these problems may arise in a large number of specific applications, and for each of them optimization facilities lead to drastic simplifications of problem specifications. We therefore believe that combining current logic programming languages with a general language for qualitative optimization, for which we derived several requirements, will greatly increase the impact of ASP on AI and computer science in general.

Acknowledgements

Thanks to Hans Tompits and the anonymous reviewers for helpful comments.

References

1. C. Baral. *Knowledge representation, reasoning and declarative problem solving*, Cambridge University Press, 2003.
2. C. Baral, C. Uyan, Declarative specification and solution of combinatorial auctions using logic programming, *Proc. LPNMR'01*, Springer Verlag, 2001.
3. M. Balduccini, M. Gelfond, Logic Programs with Consistency-Restoring Rules, In P. Doherty, J. McCarthy, M.-A. Williams (editors), *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, Mar 2003.
4. G. Brewka, Logic programming with ordered disjunction, In *Proc. AAAI-02*, 100-105, Morgan Kaufmann, 2002.
5. G. Brewka and T. Eiter, Preferred answer sets for extended logic programs, *Artificial Intelligence* 109: 297-356, 1999.
6. G. Brewka, I. Niemelä, and M. Truszczynski, Answer set optimization, *Proc. IJCAI-03*, 867-872, Acapulco, 2003.
7. G. Brewka, I. Niemelä, and T. Syrjänen, Implementing ordered disjunction using answer set solvers for normal programs, *Proc. JELIA 2002*, Springer Verlag, 2002.
8. F. Buccafurri, N. Leone, P. Rullo, Enhancing disjunctive datalog by constraints, *IEEE Transactions on Knowledge and Data Engineering*, 12(5), 845-860, 2000.

9. L. Console, D.T. Dupre, P. Torasso, On the relation between abduction and deduction, *Journal of Logic and Computation* 1(5):661-690, 1991.
10. C. V. Damasio and L. M. Pereira, A Survey on Paraconsistent Semantics for Extended Logic Programas, in: D.M. Gabbay and Ph. Smets (eds.), *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, vol. 2, pp 241-320, Kluwer Academic Publishers, 1998.
11. O. Dressler, P. Struss, The consistency-based approach to automated diagnosis of devices, in: G. Brewka (ed), *Principles of Knowledge Representation*, CSLI Publications, 1996.
12. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The kr system dlv: Progress report, comparisons and benchmarks, *Proc. Principles of Knowledge Representation and Reasoning, KR-98*, Morgan Kaufmann, 1998.
13. T. Eiter, W. Faber, N. Leone, G. Pfeifer, The Diagnosis Frontend of the dlv System, *AI Communications* 12(1-2): 99-111, 1999.
14. T. Eiter, M. Fink, G. Sabbatini, H. Tompits, A generic approach for knowledge-based information-site selection, *Proc. Principles of Knowledge Representation and Reasoning*, KR-02, Toulouse, Morgan Kaufman, 2002
15. T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, Answer set planning under action costs, *Proc. JELIA 2002*, Springer Verlag, 2002, extended version to appear in Journal of Artificial Intelligence Research.
16. K. Konolige, Abduction versus closure in causal theories, *Artificial Intelligence* 53: 255-272, 1992.
17. M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases, *New Generation Computing*, 9:365–385, 1991.
18. V. Lifschitz, Answer set programming and plan generation, *Artificial Intelligence*, 138(1-2):39–54, 2002.
19. V. Marek and M. Truszczynski, Stable models and an alternative logic programming paradigm, in: *The Logic Programming Paradigm: a 25-Year Perspective*, 1999.
20. I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241-273, 1999.
21. I. Niemelä and P. Simons, Efficient implementation of the stable model and well-founded semantics for normal logic programs, *Proc. 4th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*, Springer Verlag, 1997.
22. Y. Peng and J. Reggia, Abductive inference models for diagnostic problem solving, *Symbolic Computation - Artificial Intelligence*, Springer, 1990.
23. L. M. Pereira, J. J. Alferes and J. Aparicio. Contradiction Removal within Well Founded Semantics, in: A. Nerode, W. Marek, V. S. Subrahmanian (editors), *Logic Programming and Nonmonotonic Reasoning*, pages 105-119, MIT Press, 1991.
24. D. Poole, An architecture for default and abductive reasoning, *Computational Intelligence*, 5(1): 97-110, 1989.
25. C. Sakama and K. Inoue, Prioritized logic programming and its application to commonsense reasoning, *Artificial Intelligence*, 123(1-2):185–222, 2000.
26. T. Schaub and K. Wang, A comparative study of logic programs with preference, In *Proc. Intl. Joint Conference on Artificial Intelligence, IJCAI-01*, 2001.
27. P. Simons, I. Niemelä, and T. Soininen, Extending and implementing the stable model semantics, *Artificial Intelligence*, 138(1-2):181–234, 2002.
28. T. Soininen, *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks*, PhD thesis, Helsinki University of Technology, Finland, 2000.