Computing Diverse Optimal Stable Models

Javier Romero, Torsten Schaub, and Philipp Wanko

No Institute Given

Abstract. We

1 Introduction

- Answer Set Programming (ASP; [?]) has become a prime paradigm for solving combinatorial problems in the area of knowledge representation and reasoning.
- As a matter of fact, such problems have an exponential number of solutions in the worst-case.
  A first means to counterbalance this is to impose a preference relation among solutions in order to filter out optimal ones.
  Often enough, this still leaves us with a large number of optimal models.
- A typical example is the computation of Pareto frontiers for multi-objective optimization problems, as encountered in Design space exploration [?] or Timetabling [?]. 1
- Other examples include product configuration, planning, and phylogeny, as discussed in [?].
- This calls for computational support that allows for identifying small subsets of diverse solutions.
- The computation of diverse answer sets was first considered in [?].
- [?] deal with the analogous problem regarding optimal answer sets in the context of answer set optimization [?]
- Beyond ASP, the computation of diverse solution is also studied in SAT [?] and CP [?].
- Contributions
  -
    2: *TO BE FILLED*
  - Last but not least, our framework is easily customizable thanks to its implementation via multi-shot solving techniques. In particular, this abolishes the need for internal solver modifications that were partly necessary in previous approaches. We have implemented our approach as an extension to the preference handling framework *asprin*.
    asprin 2
- Although our elaboration concentrates on diversity, our approach applies just as well to to its dual concept of similarity. (This is also reflected by its implementation supporting both settings.)
- Design space exploration [?]
- Timetabling [?]
- *asprin* [?]

1 T: Here we could need specifics about the application areas

2 Background

– We consider logic programs P over some set $\mathcal{A}$ of atoms along with a strict partial order $\prec \subseteq \mathcal{A} \times \mathcal{A}$ among their stable models.
  Given two stable models X, Y of P, $X \succ Y$ means that X is preferred to Y.
– Then, a stable model X of P is *optimal* wrt $\succ$, if there is no other stable model Y such that $Y \succ X$.
– In what follows, we often leave the concrete order implicit and simply refer to a program with preferences and its optimal stable models.
– Note that an empty order yields all stable models of a program.
  Hence, our contributions also apply to this base case without further mention.
– [3]For simplicity, we consider a Hamming distance between two stable models X, Y of a program P over $\mathcal{A}$, defined as $d(X,Y) = |\mathcal{A} - X - Y| + |X \cap Y|$.
– Given a logic program P with preferences and a positive integer n, we follow [**?**] in defining a set $\mathcal{X}$ of (optimal) stable models of P as *most diverse*, if $\min\{d(X,Y) \mid X, Y \in \mathcal{X}, X \neq Y\} > \min\{d(X,Y) \mid X, Y \in \mathcal{X}', X \neq Y\}$ for every other set $\mathcal{X}'$ of (optimal) stable models of P.
– We are thus interested in the following problem:
  Given a logic program P with preferences and a positive integer n, find n most diverse optimal stable models of P.
– [4]
– [5]
– [6]

[3] T: How general can we be…? Discuss…?! JR: Let's talk about this hangouting.

OK

[4] JR: Example on *asprin*? (maybe, less(weight) and pareto would be helpful) TS

[5] JR: something about heuristics is also needed TS

[6] JR: a running example would be nice JR

preference progr—??

3 Overview of methods

– We summarize the methods developed, and the contributions.
– Complete methods:

0. maxmin

  • Enumerate all: Enumeration
    1. Enumerate all optimal models of the logic program P with preferences. This is implemented via *asprin* enumeration mode.
    2. Find among all optimal models, those n which are most diverse. This is implemented via a logic program with preferences.

  Contributions: *asprin* program Step 2, preference program for preference type $maxmin$.

  ~~Duplication~~ Replied

  • n copies:
    1. Translate the logic program with preferences into a generate and test problem.
    2. Translate the generate and test problem into a disjunctive logic program.
    3. Reify the logic program, and add a metaencoding such that every stable model of the metaencoding along with the reified program, correspond to n optimal models of the original logic program.
    4. Add a maxmin preference statement to select the stable models corresponding to n most diverse optimal stable models.

~ existing work

pref. program?

Algorithm 1: $iterative(P, n)$

Input : P is a logic program ~~possibly~~ with preferences, n is a positive integer
Output : A set of solutions of P, or $\perp$

$\mathcal{X} = \{solve(P, \emptyset)\}$;
while $test(\mathcal{X})$ **do**
    $\mathcal{X} = \mathcal{X} \cup solve(P, \mathcal{X})$;
return $solution(\mathcal{X})$;

Contributions: translation from *asprin* program to generate and test problem, translation from generate and test to disjunctive logic program, metaencoding for n copies, maxmin preference statement for most diverse optimal models.

~~elaborate~~

– Approximation ~~methods~~: They are variations of Algorithm 2. local ?
In the basic case, $test(X)$ returns $true$ while there are less than n solutions in X, $solution(X)$ returns the set X, and the algorithm simply computes n solutions by calling $solve$. This can be further elaborated. The methods differ in the implementation of the $solve(P, n)$ ~~call~~
  • Find a solution most dissimilar to those in $\mathcal{X}$.
    1. Add a maxmin preference statement to maximize the minimal distance to any of the solutions in $\mathcal{X}$.
  Contributions: Extending *asprin* to support preferences on top of it, which is implemented extending *asprin* with support for queries.
  • Consider a partial interpretation I distant to $\mathcal{X}$, and find a solution close to I.
    1. Select a partial interpretation I: here we consider various possibilities, i.e., a random one.
    2. Add a maxmin preference statement to minimize the distance to I. ?
  Contributions: Same as above.
  • Find any other optimal model. ?
  • Heuristics may be combined with any of the previous methods. ?

4 Maxmin optimization in *asprin*

– All methods apply $maxmin$ optimization via *asprin* preference type $maxmin$.
– *asprin* preference type *maxmin* is defined as: $dom(maxmin)$ is $\mathcal{P}(\{g, w, t : F\})$, where g and w are integers, and t is a term tuple, F is a boolean formula, and $\mathcal{P}$ stands for the power set. We say that g appears in E if there is some preference element with g as the first term. Given a set of preference elements of that form, $maxmin$ maps these elements to the preference relation defined as follows. Given ~~an~~ stable model X, a set of preference elements E, and an integer g standing for a group, let $w(X, E, g)$ be

$$\sum_{(w,\mathbf{t}) \in \{w,\mathbf{t} | g,w,\mathbf{t}:F \in E, X \models F\}} w$$

\textdisplay

Then

$$X > Y \text{ if } \min\{w(X, E, g) \mid g \text{ appears in } E\} > \min\{w(Y, E, g) \mid g \text{ appears in } E\}$$

Basic reasoning modes
Advanced reasoning modes
technique

- Switching the signs of the weights in the preference statements, we get $minmax$ preference, and with only one group, it reduces to $more(weight)$ (or $less(weight)$, switching the signs).
- The preference type is implemented by the following preference program:

listing settings

```
#program preference(maxmin).
%%% gather groups
group(P,G) :- preference(P,maxmin), preference(P,_,_,_,(G,W,T)).

%%% holds must be better
better(P) :- preference(P,maxmin),
             better(P,G) : group(P,G).

better(P,G) :- preference(P,maxmin), group(P,G), valueh'(P,V),
               V < #sum { W,T : holds(X), preference(P,_,_,for(X),(G,W,T)).

%%% get min value for holds'
valueh'(P,G,V) :- preference(P,maxmin), group(P,G),
                  V = #sum { W,T : holds'(X), preference(P,_,_,for(X),(G,W,T)
valueh'(P,V) :- preference(P,maxmin), V = #min { VV : valueh'(P,G,VV) }.
```

- The naive implementation of this preference in *clingo* via $\#minimize$ statements, leads to large groundings, in the longer version of this paper, we investigate other possible encodings, and compare them with the *asprin* implementation.

too unspecific → more details!)

5 Generate and Test in ASP

~ Eiter/Polleres

Definition 1 (Generate and Test). *Let P and Q be two nondisjunctive logic programs, and X an interpretation of P. X is a generate and test solution for $\langle P, Q\rangle$ if X is a stable model of P and $\{holds'(a) \mid a \in X\} \cup Q$ is unsatisfiable.*

- Generate and Test (GT) is a useful setting for representing problems at the second level of the polynomial hierarchy. [7]
- Example (quantified boolean cnf). Let $\exists X \forall Y \phi$ be a quantified boolean CNF formula, where ϕ is a CNF formula over atoms $X \cup Y$ such that $X \cap Y = \emptyset$. This can be represented in ASP via facts:
  - `clause(C)`: for every clause `C` in ϕ
  - `exists(V)`: for every variable `V` $\in X$
  - `forall(V)`: for every variable `V` $\in Y$
  - `pos(C,V)`: for every positive literal `V` in clause `C`.
  - `neg(C,V)`: for every negative literal `V` in clause `C`.

  Let P be the program:

```
{ holds(X) : exists(X) }.
```

  and Q be the program:

[7] JR: I put three examples here, but I don't know whether the first two should go. The first (2QCNF) is good for proving the hardness of the problem, the second (conformant planning) shows how to represent easily an interesting problem, and the third is asprin.

```
{ holds(X) : forall(X) }.
bot :- clause(C); not holds(X) : pos(C,X);
                      holds(X) : neg(C,X).
:- not bot.
holds(X) :- holds'(holds(X)).
```

The generate and test solutions of $\langle P, Q\rangle$ correspond one to one to the models of $\exists X \forall Y \phi$. The atom *bot* holds if the interpretation of the variables in $X \cup Y$ is not a model of ϕ. Informally, P guesses a solution S, then if $\{holds'(a) \mid a \in S\} \cup Q$ is unsatisfiable, there is no interpretation of the atoms in Y that makes ϕ false, which means that for all interpretations of the atoms in Y, ϕ is true, and the Boolean formula holds.

– Example (conformant planning). [8]Let $C = \langle F, A, T, I, G, n\rangle$ be a conformant planning problem with fluents F, actions A, transition function $T : F \times A \rightarrow F$, initial fluents $I \subseteq F$, goal fluent $G \in F$, and a positive integer n representing the plan length. The transition function T induces a transition diagram $D_T = \langle S, E\rangle$ with states $S = \{s \mid s \subseteq F\}$ and arcs from s_1 to s_2 labelled by a if $T(s_1, a) = s_2$). A solution to C is a sequence of actions $a_1, a_2, \ldots, a_{n-1}, a_n$ such that for all possible states $I' \in S$, if $I \subseteq I'$ then there is a path of length n in D_T from I' to a state s_f such that $g \in s_f$. Let P_T be a logic program representing all paths of length n in ~~the~~ D_T. Predicate `holds(F,T)` stands for fluent `F` being true at state `T` of the path, and `occurs(A,T)` stands for action `A` connecting states `T-1` and `T` of the path. Let P be the program:

[8] JR: If we want this to stay, I can make it much cleaner ✓

M

```
{ occurs(A,T) : action(A) } :- T=1..n.
```

and Q be the program:

```
:- not holds(F,0), initial(F).
:- holds(goal,n).
:- not occurs(A,T), holds'(occurs(A,T)).
```

The generate and test solutions of $\langle P, Q \cup P_T\rangle$ correspond one~~-~~to~~-~~one to the conformant plans of the problem.

semantics
syntax
implementation

– Example. Preferences in *asprin*. Let P be a logic program with signature $\mathcal{A}$, let s be a preference statement defining preference relation $\succ_s$ over $\mathcal{A} \times \mathcal{A}$, and Q a preference program for s. The generate and test solutions of $\langle P, P \cup Q \cup \{holds(a) \leftarrow a \mid a \in \mathcal{A}\}\rangle$ correspond to the $\succ_s$-preferred stable models of P.

?

– Implementation. [9]

[9] JR: I copy the explanation from the Draft of Preferences

- Eiter and Gottlob invented the *saturation* technique. The idea is to re-express the problem as a positive disjunctive logic program, containing a special-purpose atom *bot*. Whenever *bot* is obtained, saturation derives all atoms (belonging to a "guessed" model). Intuitively, this is a way to materialize unsatisfiability. For automatizing this process, we build upon the meta-interpretation-based approach in [**?**]. The idea is to map a program R onto a set $\mathcal{R}(R)$ of facts via reification.The set $\mathcal{R}(R)$ of facts is then combined with a meta-encoding $\mathcal{M}$ from [**?**] implementing saturation.

- In our case, we consider for a GT problem $\langle P, Q\rangle$ the positive disjunctive logic program
$$\mathcal{R}\big(Q \cup \{\{holds'(a)\} \mid a \in \mathcal{A}_{\mathcal{P}}\}\big) \cup \mathcal{M}\,.$$
- This program has a stable model (excluding bot) for each $X \subseteq \mathcal{A}_{\mathcal{P}}$ such that $\{holds(a) \mid a \in X\} \cup Q$ is satisfiable, and it has a saturated stable model (including bot) if there is no such X.
- For computing a solution to the GT problem, one just has to add the generator program P, map the atoms of P to their names in the positive disjunctive logic program, and inforce the atom bot
$$P \cup \mathcal{R}\big(Q \cup \{\{holds'(a)\} \mid a \in \mathcal{A}_{\mathcal{P}}\}\big) \cup \mathcal{M} \cup$$
$$\{holds(a) \leftarrow a \mid a \in X\} \cup \{not\ holds(a) \leftarrow not\ a \mid a \in X\} \cup \{\leftarrow\ not\ bot\}\,.$$
[10]

[10] JR: The rules generating $holds(a)$ and $not\ holds(a)$ are not exactly like that, I have to go again through it.

– Deciding whether there is a solution to a GT problem is Σ_2^p-complete. Membership comes from the translation to disjunctive logic programming, and hardness comes from the translation from quantified boolean CNF formulas.

6 Solving queries in *asprin*

Definition 2 (Query Problem). *Let P be a logic program over $\mathcal{A}$, let s be a preference statement, and q an atom of $\mathcal{A}$, decide if any $\succ_s$-preferred stable model of P contains q.*

Methods:

– (From Y. Zhu and M. Truszczyinski, LPNMR 2013) Enumerate optimal models until one contains q.

– (From Y. Zhu and M. Truszczyinski, LPNMR 2013) Enumerate possibly nonoptimal models containing q, and test each one for optimality.

– Enumerate optimal stable models of $P \cup \{\bot \leftarrow\ not\ q\}$, testing each for optimality on P.
TO BE ADDED: Justification of the algorithm.
1. Find an optimal model X of $P \cup \{\bot \leftarrow\ not\ q\}$. If none exists, return $false$, else goto 2.
2. Find a stable model Y of $P \cup \{\bot \leftarrow\ q\}$ better than X. If none exists, return $true$. If one exists, optionally Y can be further improved until an optimal stable model of P is produced. Add to P rules deleting the best stable model generated, and all stable models worse than it. Goto 1.

– Find a stable model with query, then another better without query, then another better with query…
TO BE ADDED: Justification of the algorithm.
1. Find an stable model X of $P \cup \{\bot \leftarrow\ not\ q\}$. If none exists, return $false$, else goto 2.

2. Find a stable model Y of $P \cup \{\bot \leftarrow \ q\}$ that is better than X. If none exists, return $true$, else goto 3. Optionally, if none exists, X can be improved until an optimal model of P is obtained.
3. Find an stable model X of $P \cup \{\bot \leftarrow \ \mathit{not}\ q\}$ that is better than Y. If one exists, goto 2. If none exists, optionally, Y can be improved until an optimal model of P is obtained. Add to P rules deleting the best stable model generated and all stable models worse than it. Goto 1.

Meta preferences?

7 Preferences over preferences

[11] JR: Not a good title…

[11]

Definition 3 (Preferences over preferences). *Let P be a logic program over $\mathcal{A}$, and let s and t be two preference statements, a stable model X of P is $\succ_{s,t}$-preferred if it is $\succ_s$-preferred, and there is no $\succ_s$-preferred stable model Y of P such that $Y \succ_t X$.*

[12] JR: Copy, paste and modify from Draft on Preferences

[12]Given a program P, define $q(P)$ as the program

$$(P \setminus \{r \in P \mid \mathit{head}(r) = \emptyset\}) \cup \{u \leftarrow \mathit{body}(r) \mid r \in P, \mathit{head}(r) = \emptyset\} \cup \{\ q \leftarrow \ \mathit{not}\ u\}$$

where u and q are new atoms.

Proposition 1. *If program P is stratified, P is satisfiable iff $q \in X$, where X is the stable model of $q(P)$.*

– The Method:
  1. Find $\succ_s$-preferred model X of P optimizing $p1$. If P is unsat, return unsat, else goto 2.
  2. TODO

8 Complete methods

8.1 Enumerate all

– Enumerate all optimal stable models of P (with *asprin*), and afterwards find, among all those stable models, the n most diverse (with *asprin* again).
– This method may be exponential in space, given that we may have to compute and store an exponential number of solutions.
– For the first step, we simply enumerate all optimal stable models of P with *asprin*.
– For the second step, let $\mathcal{X} = \{X_1, \ldots, X_m\}$ be the set of m optimal stable models of P. This set may be represented in ASP via the set of atoms $A_{\mathcal{X}} = \{\mathit{holds}(a, i) \mid a \in X_i\}$. Consider the *asprin* encoding E: [13]

[13] JR: I put two encodings, the first one for asprin 1.0, the second (nicer) for asprin 2.0

```
n { select(I) : model(I) } n.
#preference(p,maxmin) {
  (I,J),1,X :: select(I) & select(J) :
holds(A,I), not holds(A,J), model(I), model(J), I < J;
  (I,J),1,X :: select(I) & select(J) : not holds(A,I),
holds(A,J), model(I), model(J), I < J
}.
```

Theren

Consider the *asprin* encoding E:

```
n { select(I) : model(I) }n.
#preference(p,maxmin) {
  (I,J),1,X :      holds(A,I), not holds(A,J), select(I), select(J), I < J;
  (I,J),1,X : not holds(A,I),
holds(A,J), select(I), select(J), I < J
}.
```

Then the optimal stable models of $A_{\mathcal{X}} \cup E$, computed by *asprin*, correspond to most diverse solutions of P.

8.2 *n* copies

Duplicate

- First, translate the *normal* input logic program with preferences P into a disjunctive logic program without preferences D_P using `asprin`. This is done applying a general framework for generate and test in ASP.
- Second, reify the resulting logic program with $reify$ tool into a set of facts F_{D_P}.
- Consider a metaencoding $meta$ such that the stable models of $F_{D_P} \cup meta$ correspond one to one to the stable models of D_P.
- For the case where D_P contains no choice rules or weight constraints, $meta$ is:

```
lits(B) :- normal(A,B).
lits(B) :- disjunction(A,B).
body(B) :- lits(B),
              hold(L) : lits(B, L), L > 0;
          not hold(L) : lits(B,-L), L > 0.
hold(A)               :- normal(A,B), body(B).
hold(A) : atoms(H,A)  :- disjunction(H,B), body(B).
```

- Consider metaencoding $meta(n)$ such that given a positive integer n, from every stable model of $F_{D_P} \cup meta(n)$, n stable models of P may be extracted.
- More technically, the stable models of $F_{D_P} \cup meta(n)$ correspond one to one to the elements of the set $\underbrace{SM(D_P) \times \ldots \times SM(D_P)}_{n}$, where $SM(D_P)$ stands for the set of stable models of D_P.
- For the case where D_P contains no choice rules or weight constraints, $meta(n)$ is:

SM(D_P)^n

redundant

n=1

```
model(1..n).
lits(B) :- normal(A,B).
lits(B) :- disjunction(A,B).
body(B,M) :- lits(B), model(M)
                hold(L,M) : lits(B, L), L > 0;
            not hold(L,M) : lits(B,-L), L > 0.
hold(A,M)               :- normal(A,B), body(B,M).
hold(A,M) : atoms(H,A)  :- disjunction(H,B), body(B,M).
```

- Note that with this basic encoding every set of n models ~~will~~ appear in $n!$ stable models. For having one stable model for every set of n models, we add the following set of rules:

vs

TO BE ADDED

- For computing most diverse solutions, we add the following preference specification:

```
#optimize(p).
#preference(p,maxmin) {
  (I,J),1,X :     hold(A,I), not hold(A,J), model(I), model(J), I < J;
  (I,J),1,X : not hold(A,I),     hold(A,J), model(I), model(J), I < J
}.
```

[14]

[14] JR: If we decide to keep the encodings, I can choose better predicates or print them nicer.

- This method does not work if P is disjunctive.

9 Approximate methods

[15]

[15] JR: I made no changes after this point.

The following methods approximate n most dissimilar solutions. They are variations of Algorithm 2.

Algorithm 2: $iterative(P, n)$

Input : P is a logic program possibly with preferences, n is a positive integer
Output : A set of solutions of P, or $\perp$

$\mathcal{X} = \{solve(P, \emptyset)\}$;
while $test(\mathcal{X})$ **do**
  $\mathcal{X} = \mathcal{X} \cup solve(P, \mathcal{X})$;
return $solution(\mathcal{X})$;

In the basic case, $test(X)$ returns $true$ while there are less than n solutions in X, $solution(X)$ returns the set X, and the algorithm simply computes n solutions by calling $solve$. This can be further elaborated. For example, $test(X)$ may return $true$ until k ($k \geq n$) solutions are in X, and $solution(X)$ returns the n most dissimilar solutions among those in X. The algorithm is complete if $test(X)$ returns $true$ until all solutions have been computed (in which case the algorithm reduces to **enumerate all** above).

The methods differ in the implementation of the $solve(P, n)$ call. Below, every method is more imprecise than the previous ones, i.e. the solutions given are more similar than with the previous methods.

n? $\mathcal{X}$?

9.1 Find a solution most dissimilar to those in $\mathcal{X}$.

- [1]

[1] For future work, when $test(X)$ allows computing more than n solutions, we could find a solution along with at most $n-1$ solutions in X, such that they altogether are most dissimilar. In

? use

– Add maxmin optimization to P to compute a solution that maximizes the minimal distance to any of the solutions in $\mathcal{X}$.
– Implementation: Without preferences, using Maxmin Optimization (see next subsection). With preferences, using the method for preferences over `asprin`, that uses the method for queries (see next subsection).

9.2 Consider a partial interpretation I distant to $\mathcal{X}$, and find a solution close to I.

– [2]
– Select a partial interpretation I:
  1. A Random one.
  2. According to *pguide* heuristic from (A. Nadel, SAT 2011). An atom is true if among the solutions in $\mathcal{X}$ it is $false$ more times than $true$, and it is false in the opposite case. In case of a tie, it does not appear in I.
  3. The most dissimilar to the solutions in $\mathcal{X}$ (computed using maxmin optimization in ASP).
  4. Different to the last added element L of $\mathcal{X}$ (for this, $\mathcal{X}$ should be a list). I may be the result of changing all signs of L ($\{\neg a \mid a \in L\} \cup \{a \mid \neg a \in L\}$), or taking only the positive atoms of L and changing the signs ($\{\neg a \mid a \in L\}$), or similarly with the negative atoms of L ($\{a \mid \neg a \in L\}$).
– Apply minimization to compute a solution as close to I as possible.
– Implementation: Without preferences, using normal optimization. With preferences, using the method for preferences over `asprin`, that uses the method for queries (see next subsection).

9.3 Find any solution of P.

? a priori model

– No optimization here, but we expect that heuristics alone give a good approximation.
– Implementation: Without preferences, add a rule to delete the last model. Alternatively, we can simply enumerate models. With preferences, use `asprin` option `--input-optimal` to delete the last computed optimal models, and all models worse than them. Alternatively, we can simply enumerate optimal models.

9.4 Heuristics

They may be combined with any of the previous three methods:

– Fix the sign of the atoms to their value in a partial interpretation I selected by any of the methods above (1–4).

this way, we make choices on the solution we look for, and on which of the previous solutions are also selected.

[2] For future work, one could consider looking for a solution close to I for a number of conflicts, and if no solution is found, pick another partial interpretation I' and continue from there.

- Adding to modifying the signs, give priority 1 to the atoms relevant for dissimilarity, or to the atoms in the partial interpretation I. Furthermore, different priorities may be given depending on the $pguide$ heuristic value (i.e., the priority of atom a is $abs(|\{Y \in \mathcal{X} | a \in Y\}| - |\{Y \in \mathcal{X} | \neg a \in Y\}|)$).
- Adding to modifying the signs, apply the dynamic heuristic. This heuristic, when the current assignment is very close to a previous solution, modifies the signs to get away from it.
- Different default sign heuristics could also be tried. For example, it would be interesting to try a random sign heuristic.

10 Experiments

11 Discussion

- Generalize existing approaches to ~~[illegible]~~ general preference programs
- Propose new approaches based upon max
-

This article was processed using the comments style on March 21, 2016.
There remain 15 comments to be processed.