Logic Programming Approach to Partial Satisfaction Planning

Tuan A. Nguyen

CSE 494/598 Course Project

Progress Report. Date: 11/14/2008

1. Introduction

Partial Satisfaction Planning [2] recently has been receiving notable attentions in planning community due to its applicability to some real-world applications where the task of modeling agent's goal is difficult, and also because of the fact that in many situation the agent with limited resources cannot achieve all pre-specific goals. In this project, we will extend the encoding based on Answer Set Programming to model Partial Satisfaction Planning and prove that the optimal stable model of the encoding corresponds to the plan with the largest *net-benefit*, a feature used to formalize optimality in Patial Satisfaction Planning.

2. ASP-based encoding for optimal cost planning

Optimal cost planning

A planning problem is characterized by a tuple $\langle s_0, A, s_G \rangle$ where s_0 is a set of positive literals representing a complete state (close-world assumption is used), s_G is a set of literals for goal conditions, and A is a set of (grounded) actions. Each action has a set of positive literals as its precondition, a set of literals as effects, and a positive execution cost.

A plan P is a sequence of actions $\langle a_0, a_1, \ldots, a_N \rangle$ such that executing this sequence of actions starting from s_0 will end up at a state where s_G is satisfied. The cost of the plan is defined as total cost of its actions. The optimal plan is one with the minimal cost.

Encoding for optimal cost planning

Smodels allows each literal to be represented with a specific weight, and the semantic of stable models now can be extended to include optimality sematic. Given a set of literals l_i with the weight $w_i (1 \leqslant i \leqslant k)$, there are 2 types of optimization statements in *smodels*:

$$minimize[l_1 = w_1, l_2 = w_2, \ldots, l_k = w_k]$$

$$maximize[l_1=w_1, l_2=w_2, \ldots, l_k=w_k]$$

The optimal stable model of a logic program with these optimization statements is the stable model with the minimal/maximal total weights of literals listed in the statements.

In the simplest case where all literals l_i have the same unity weights ($w_i=1 \forall i=1,2,\ldots,k$), we can write the optimization statements in a more simple form where the curly braces are used and weights are ignored:

$$minimize\{l_1, l_2, \ldots, l_k\}$$

$$maximize\{l_1, l_2, \ldots, l_k\}$$

Optimal cost planning problem now can be encoded in *smodels*, as briefly discussed in [1]. As an illustration, we consider the following simple example[1]:

Example 1: *There are three devices* d_1 , d_2 , d_3 *which can be turned on or off by toggling three switches a, b, c respectively. At first, all devices are turned on, and we want to search for a sequence of actions to turn off all of them with the smallest number of actions.*

It is easy to figure out that the optimal plan (w.r.t the number of actions executed) is *<toggle(a), toggle(b), toggle(c)>*. In addition to parts used to encode a usual planning problem, we need to introduce additional fluents, each of which represents the fact that a corresponding action is executed in the optimal plan (with a cost) or not. The additional part for optimal planning of this problem is as followed (the full code is in the *Appendix*):

```
occurs(S,I):-toggle(S,I),switch(S),level(I).
cost(toggle,S,I,1):-switch(S),level(I).
minimize [occurs(S,I):cost(toggle,S,I,C) = C].
```

The additional fluents here are *occurs(S,I)*, and the minimization statement uses it over all possible legal assigments for switches *S* and level *I*. Later on, we will extend this encoding for Partial Satisfaction Planning problem, where the status of devices at the final state of the plan is assumed to give us some reward values.

3. Partial Satisfaction Planning

Problem formulation

The Partial Satisfaction Planning (PSP) [2] is extended from optimal cost planning in the sense that all literals (in goal) are assigned reward values, and all of them do not need to be achieved. The *net-benefit* of a plan $P=\langle a_1, a_2, \ldots, a_l \rangle$ is defined as:

1 This is the exercise 6.5 in lecture 6, extended to 3 devices.

$$net-benefit(P)=\sum_{g\in G(P)} reward(g)-\sum_{i=1}^{i=l} cost(a_i)$$

which is the difference between total rewards the plan can achieve and the total cost incurred. The optimal plan in PSP problem is one with the largest net-benefit.

As an example[2], Figure 1 shows a logistic domain in which we have a truck initially at city A that is considering going to city B, C, D, E to achieve their rewards. The optimal plan here is P = <*move(A, C), move(C, D), move(D, E)*> with 160 net-benefit value.

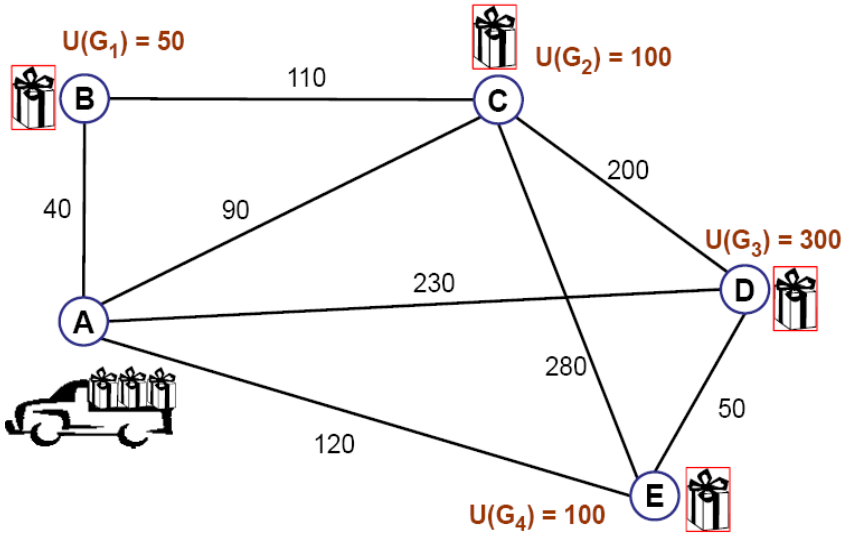


Figure 1: Logistic example for PSP problem.

ASP-based encoding for PSP

We now extend the encoding for optimal cost planning to PSP problem. We first describe the running example that will be used for illustration.

Example 2: *The problem in example 1 now is modified so that actions* toggle *have different costs and the status of devices have reward values. In particular:*

- toggle*(a) and* toggle*(b) have $1 cost.*
- toggle*(c) has $5 cost.*
- *The devices d1, d2 in off status give $2 reward, and d3 gives $4 reward. In case being turned on, all of them give $0 reward.*

In the initial state, we assume that all three devices are turned on, and we want to encode this problem so that the optimal stable model corresponds to the plan with the largest net-benefit. (Note that here the

2 This example is taken from [2].

goals are not specified, which means that all executable sequence of actions, including empty, is a valid plan.)

In example 1, *occurs* is used to represent the fact that an action *toggle* is executed with its cost, and the optimization is stated on this fluent over all possible legal assigments to switch variable *S* and level variable *I*. We now extend this fluent to represent the status of devices with the corresponding reward. The excerpt of the encoding is given as followed (the full encoding is in the *Appendix*):

```
1. occurs(toggle,S,I) :- toggle(S,I), switch(S), level(I).
2. occurs(on,X,I) :- on(X,I), device(X), level(I).
3. occurs(off,X,I) :- -on(X,I), device(X), level(I).

4. reward(toggle,a,I,-1) :- level(I).
5. reward(toggle,b,I,-1) :- level(I).
6. reward(toggle,c,I,-5) :- level(I).

7. reward(off,d1,I,0) :- level(I), I < lastlevel.
8. reward(off,d2,I,0) :- level(I), I < lastlevel.
9. reward(off,d3,I,0) :- level(I), I < lastlevel.

10. reward(off,d1,lastlevel,2).
11. reward(off,d2,lastlevel,2).
12. reward(off,d3,lastlevel,4).

13. reward(on,d1,I,0) :- level(I).
14. reward(on,d2,I,0) :- level(I).
15. reward(on,d3,I,0) :- level(I).

16. maximize [occurs(A,S,I) : reward(A,S,I,R) = R].
```

The *occurs* fluents representing on and off status of devices are defined in line 2 and 3. As eventually we want to maximize the net-benefit by summing the weights of *occurs* fluents in the stable model, which is the difference between the total rewards and costs, in lines 4 – 6 we define cost for *toggle* actions as *negative* reward. Line 7 – 9 define $0 reward for off status of devices in level before the last one, and their positive rewards at final level are encoded in line 10 – 12. The last three lines define rewards for on status of devices.

The optimal plan for this problem is *P* = <*toggle(a), toggle(b)*>. Note that even though turning off the device d3 gives $4 reward, but costs $5 which makes the net-benefit decreases.

4. Correctness of the encoding

In this section, we are interested in the correctness of the encoding proposed. For a given consistent answer set S of a logic program corresponding to a PSP problem, let A_i^S be set of grounded actions corresponding to action literals in S such that the second argument (i.e., argument representing level) is i. Similarly, let s_i^S be set of facts corresponding to grounded fluents in S such that the second argument (i.e., argument representing level) is i. The correctness of proposed encoding is based on the following two theorems.

Theorem 1: *Let PP be a PSP problem, and lp(PP) be its correponding ASP encoding.*

a) Assume that PP has a solution plan $P=\langle A_0, A_2, \ldots, A_{l-1}\rangle$, *and* $T=\langle s_0, s_1, \ldots, s_l\rangle$ *be corresponding sequence of states such that* s_{i+1} *is resulting state of applying set of actions* A_i *at state* s_i $(0\leqslant i\leqslant l-1)$, *and* $s_l \models s_G$. *Then, the corresponding ASP enconding has an answer set* S *such that* $A_i=A_i^S$ $(0\leqslant i\leqslant l-1)$, $s_i=s_i^S$ $(0\leqslant i\leqslant l+1)$, *and the net-benefit of* P *is equal to cumulative reward of action literals and grounded fluents in* S.

b) Assume that the ASP encoding representing a PSP problem PP has an answer set S. *Let* $P=\langle A_0, A_2, \ldots, A_{l-1}\rangle$ *and* $T=\langle s_0, s_1, \ldots, s_l\rangle$ *such that* $A_i=A_i^S$ $(0\leqslant i\leqslant l-1)$ *and* $s_i=s_i^S$ $(0\leqslant i\leqslant l+1)$. *Then* P *is a solution plan of the PSP problem, i.e.,* s_0 *is initial state of PP,* s_{i+1} *is resulting state of applying set of actions* A_i *at state* s_i $(0\leqslant i\leqslant l-1)$, $s_{l+1} \models s_G$, *and the net-benefit of* P *is equal to cumulative reward of action literals and grounded fluents in* S.

In [3] and [1], Eiter et al proves a very similar theorem about the corresponding between optimal solution of action cost planning and optimal answer set of its ASP encoding. The proof can be broken down into two parts:

- Part 1: There is a corresponding between a solution of cost-optimal planning and an answer set of its encoding.
- Part 2: The cumulative cost of action in a solution plan is equal to the cumulative weights of literal *cost* in corresponding answer set.

The first part is proved by considering a simplified encoding of the original one in which all rules and constraints related to *cost* literals are stripped off, and applying *splitting sequence theorem* [4]. Since our encoding for PSP problem extends one for cost optimal planning problem by modeling action costs as negative rewards, a slight change in this proof is enough to show a similar result: *there is a*

corresponding between a solution of PSP planning problem and an answer set of its encoding.

The *splitting set theorem* [4] is applied in order to prove the correctness of statement in part 2. The simplified encoding use for part 1 now is considered as the *bottom* part of original encoding, whose answer set can be extended to a *unique* answer set of original program (due to the fact that the *top* part is a positive program). Again, this idea can be applied for our encoding representing PSP problem: while the bottom part of our encoding is the same as one used in [3], the top part is comprised of rules and constraints in which literals *reward*, *occurs* and optimization statement *maximize* are involved.

Given that (i) there is a correspondence between solution plan of a PSP problem and answer set of its ASP encoding; and (ii) the net-benefit of solution plan equals to cumulative weight of *reward* literals in the corresponding answer set, it is easy to conclude the existence between the optimal net-benefit plan of a PSP problem and the optimal answer set of its encoding:

Theorem 2: *For a PSP problem, the optimal net-benefit solution corresponds to the optimal answer set of the corresponding ASP encoding.*

Appendix

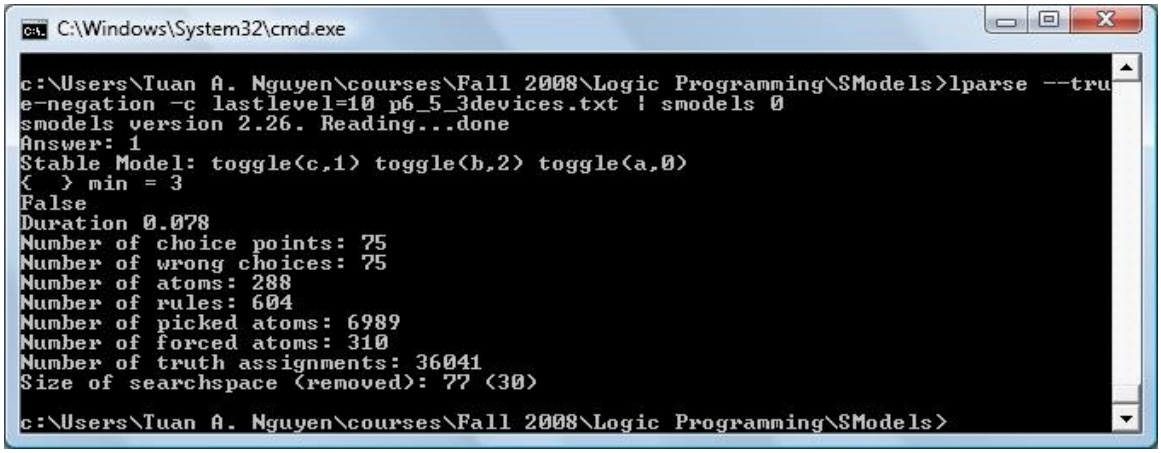
1. Encoding for example 1

```
level(0..lastlevel).

device(d1). device(d2). device(d3).
switch(a). switch(b). switch(c).
control(a,d1). control(b,d2). control(c,d3).

% Each device is on or off in initial state
1 {on(X,0),-on(X,0)} 1 :- device(X).

% At each level, there is at most one action executed
0 {toggle(S,I) : switch(S)} 1 :- level(I), I < lastlevel.

% Each switch is toggled or not at each level
{toggle(S,I)} :- level(I), switch(S), I < lastlevel.

% Define two consecutive level
next(I,J) :- level(I), level(J), J - I == 1.
```

```
% The effects of action "toggle"
-on(X,I1) :- on(X,I0), toggle(S,I0), device(X), level(I0), level(I1), next(I0,I1), switch(S), control(S,X).
on(X,I1) :- not on(X,I0), toggle(S,I0), device(X), level(I0), level(I1), next(I0,I1), switch(S), control(S,X).

% If there is a change between 2 consecutive levels, there must be a corresponding toggle action executed.
toggle(S,I0) :- on(X,I0), -on(X,I1), level(I0), level(I1), next(I0,I1), switch(S), control(S,X).
toggle(S,I0) :- -on(X,I0), on(X,I1), level(I0), level(I1), next(I0,I1), switch(S), control(S,X).

% Inertia
on(X,I1) :- on(X,I0), not -on(X,I1), level(I0), level(I1), next(I0,I1), device(X).
-on(X,I1) :- -on(X,I0), not on(X,I1), level(I0), level(I1), next(I0,I1), device(X).

% Initial state
on(d1,0).
on(d2,0).
on(d3,0).

% Goal state
-on(d1,lastlevel). -on(d2,lastlevel). -on(d3,lastlevel).

%%%%
% This is used for optimal plan: we want to define "lastlevel" as an upper-bound, but the plan returned is
% optimal w.r.t the number of "non-empty" action levels (i.e. level with one action executed), and there is
% not any "empty" level preceding a "non-empty" one (therefore, we don't need to compress the resulting plan)
%%%%

% First, we define "non-empty" level and a choice rule
-empty_level(I) :- toggle(S,I), switch(S), level(I).
1 {toggle(S,I) : switch(S)} 1:- -empty_level(I), level(I).
1 {empty_level(I), -empty_level(I)} 1 :- level(I).
```

```
% Second, we ensure that there is not any "empty" level preceding a "non-empty"
one
:- empty_level(I0), -empty_level(I1), level(I0), level(I1), I0 < I1.

% Third, we need to define the cost of action "toggle" as unity.
occurs(S,I) :- toggle(S,I), switch(S), level(I).
cost(toggle,S,I,1) :- switch(S), level(I).

% Finally, optimal plan is one with the least number of "toggle" actions executed
minimize [occurs(S,I) : cost(toggle,S,I,C) = C].

% Display statements
hide.
show toggle(_,_).
```

Output by Smodels:



As we can see, the stable model is {toggle(a,0), toggle(b,1), toggle(c,2)}, corresponding to the optimal plan *P = <toggle(a), toggle(b), toggle(c)>*

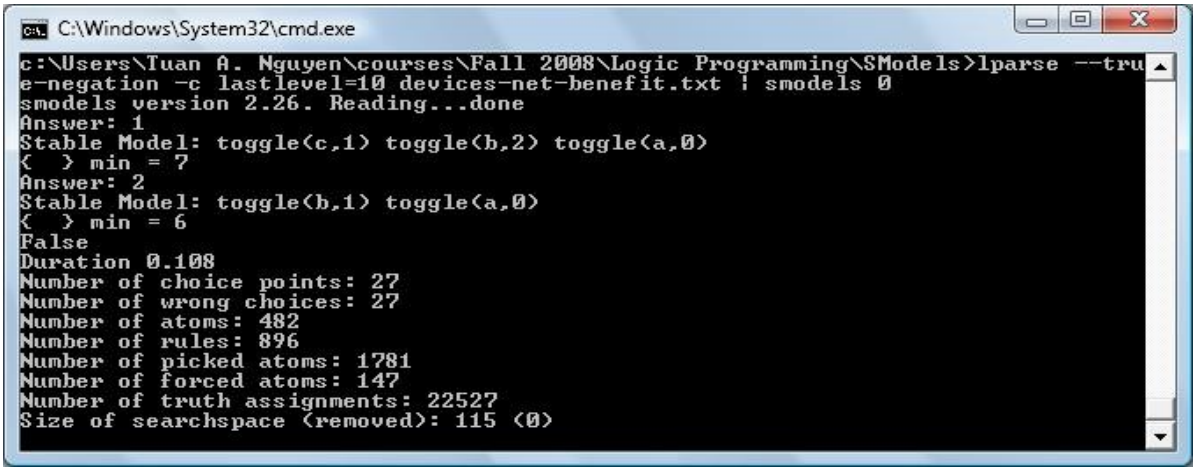
2. Encoding for example 2

```
level(0..lastlevel).
device(d1). device(d2). device(d3).
switch(a). switch(b). switch(c).
control(a,d1). control(b,d2). control(c,d3).

% Each device is on or off in initial state
1 {on(X,I),-on(X,I)} 1 :- device(X), level(I).
```

```
% At each level, there is at most one action executed
0 {toggle(S,I) : switch(S)} 1 :- level(I).

% Each switch is toggled or not at each level
{toggle(S,I)} :- level(I), switch(S).

% Define two consecutive level
next(I,J) :- level(I), level(J), J - I == 1.

% The effects of action "toggle"
-on(X,I1) :- on(X,I0), toggle(S,I0), device(X), level(I0), level(I1), next(I0,I1),
switch(S), control(S,X).
on(X,I1) :- -on(X,I0), toggle(S,I0), device(X), level(I0), level(I1), next(I0,I1),
switch(S), control(S,X).

% If there is a change between 2 consecutive levels, there must be a corresponding
toggle action executed.
toggle(S,I0) :- on(X,I0), -on(X,I1), level(I0), level(I1), next(I0,I1), switch(S),
control(S,X).
toggle(S,I0) :- -on(X,I0), on(X,I1), level(I0), level(I1), next(I0,I1), switch(S),
control(S,X).

% Inertia
on(X,I1) :- on(X,I0), not -on(X,I1), level(I0), level(I1), next(I0,I1), device(X).
-on(X,I1) :- -on(X,I0), not on(X,I1), level(I0), level(I1), next(I0,I1), device(X).

% Initial state
on(d1,0).
on(d2,0).
on(d3,0).

% No goal conditions are specified for Partial Satisfaction Planning problem!

% These rules are used to "compress" resulting plan
-empty_level(I) :- toggle(S,I), switch(S), level(I).
1 {toggle(S,I) : switch(S)} 1:- -empty_level(I), level(I).
1 {empty_level(I), -empty_level(I)} 1 :- level(I).
:- empty_level(I0), -empty_level(I1), level(I0), level(I1), I0 < I1.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This part is used for optimal answer set w.r.t total rewards of literals %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
occurs(toggle,S,I) :- toggle(S,I), switch(S), level(I).
occurs(on,X,I) :- on(X,I), device(X), level(I).
occurs(off,X,I) :- -on(X,I), device(X), level(I).

reward(toggle,a,I,-1) :- level(I).
reward(toggle,b,I,-1) :- level(I).
reward(toggle,c,I,-5) :- level(I).

reward(off,d1,I,0) :- level(I), I < lastlevel.
reward(off,d2,I,0) :- level(I), I < lastlevel.
reward(off,d3,I,0) :- level(I), I < lastlevel.

reward(off,d1,lastlevel,2).
reward(off,d2,lastlevel,2).
reward(off,d3,lastlevel,4).

reward(on,d1,I,0) :- level(I).
reward(on,d2,I,0) :- level(I).
reward(on,d3,I,0) :- level(I).

% Finally, optimal plan is one with the least number of "toggle" actions executed
maximize [occurs(A,S,I) : reward(A,S,I,R) = R].

% Display statements
hide.
show toggle(_,_).
```

Output by Smodels:



As we can see, the optimal stable model is *{toggle(b,1), toggle(a,0)}*, which corresponds to the optimal plan *P* = <*toggle(a), toggle(b)*>.

References

1. *Answer Set Planning under Action Costs*. Thomas Eiter, Wolgang Faber, Nicola Leone, Gerald Pfeifer, Axel Polleres. Journal of Artificial Intelligence Research 19 (2003).
2. *Anytime Heuristic Search for Partial Satisfaction Planning*. J. Benton, Minh Do, Subbarao Kambhampati. Artificial Intelligence Journal 2008 (to appear).
3. *A logic programming approach to knowledge-state planning, II: The* DLV^K *system*. Thomas Eiter, Wolgang Faber, Nicola Leone, Gerald Pfeifer, Axel Polleres. Artificial Intelligence 144 (2003) 157 – 211.
4. *Splitting a logic program*. Vladimir Lifschitz, Hudson Turner. Proceedings of the eleventh international conference on Logic programming (ICLP), 1994.