Decomposition Trick for Planning with Answer Set Semantics

Michal Čertický

Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava, Slovakia
certicky@fmph.uniba.sk

Abstract. In recent years, we have seen several attempts to use the Answer Set Programming (ASP) for planning. Typically, the whole planning domain is encoded into a large ASP meta-program, where predicates $occurs(A,T)$ and $holds(F,T)$ express what holds and which actions are executed in all the time steps of our plan. Since the complexity of answer set computation is exponential in the number of input atoms, we propose a technique of dividing this meta-program into several smaller ones - one for every state transition. Actual plan finding can then be outsourced to a graph-search algorithm. This method, called *"decomposition trick"*, increases the performance of planning with ASP semantics dramatically. We introduce a prototype planner $GRASP$ using this technique, and provide the experimental comparison with an alternative planning system DLV^K.

Keywords: ASP, planning, answer set programming

1 Introduction

Background

During last two decades, we have seen several attempts to use ASP solvers for planning tasks. Semantics of ASP enables us to elegantly deal with incompleteness of knowledge, and makes the representation of action's properties easy, due to nonmonotonic character of negation as failure [Lifschitz, 2002].

As an example of using ASP for planning we can mention either the theoretic proposals in [Subrahmanian-Zaniolo, 1995] and [Lifschitz, 2002], or more implementation-oriented system descriptions published in [Eiter et al., 2002] or [Gebser et al., 2011].

Typical Approach

These systems or approaches, despite varying in some details, have one thing in common. They encode the planning domain into a single logic meta-program typically composed of the following:

(1) Representation of initial world state, goal, and (optionally) a background knowledge.
(2) Set of universal axioms representing domain-independent causal dynamics.
(3) Collection of facts enumerating all the valid time steps:

$$time(1).\ time(2).\ \dots\ time(maxTime).$$

(4) Description of preconditions and effects of every action of our domain.
(5) Action-time occurrence choice rules like these ones:

$$occurs(A,T) \leftarrow action(A), time(T), not\ -occurs(A,T).$$

$$-occurs(A,T) \leftarrow action(A), action(A'), time(T), occurs(A',T), A \neq A'.$$

Then in the end, they use one of available ASP solvers (DLV [Leone et al., 2006], SMODELS [Syrjänen, 2001], etc.) to compute the answer sets of this meta-program. Each of these answer sets then represents one resulting plan.

Choice rules (type (5)) cause solver to generate several answer sets, where different actions occur in different time steps. Rules of a type (2) and (4) then generate the facts representing successive world states at individual time steps, based on the initial world configuration (1). Finally, rules of a type (3) and (4) filter out all the answer set candidates representing invalid plans: plans where action occurred even though its preconditions were not met, plans not leading to a goal, etc.

Problem

Finding the answer sets is an NP-complete problem [Simons, 2002]. Current ASP planners generate the answer sets of a single meta-program, which represents the whole planning task. The problem is, that since the input meta-program represents the whole panning task, it is too big, making this technique unusable in larger domains.

Splitting the problem into several smaller logic programs allows us to solve the task per partes. Since the ASP solving algorithms are **exponential in the number of input atoms**, solving even a large number of small inputs is potentially faster than solving one big input program. See [Čertický, 2011] for detailed analysis of the time complexity.

2 Suggestion for Improvement

We suggest using more traditional methods (preferably a heuristic graph-search algorithm like A^* [Hart-Nilsson-Raphael, 1968]) for planning itself, and always using the ASP solvers only for calculating the **transitions between two consecutive world states** (small sub-problems). We call this technique a **"decomposition trick"**. It allows us to:

– Reduce the size of logic programs passed to the ASP solver.
– Solve a number of state transitions in parallel.
– Outsource a significant part of planning task to a faster graph-search algorithm.
– Use specialized planning heuristics while searching for plans[1].
– Preserve the expressiveness of knowledge representation, by still using the ASP semantics.

Describing graph-search algorithms in detail is not necessary for the purposes of this paper. It is sufficient to explain the relation between our planning problem and an oriented edge-labeled graph $G = (N, E)$:

Any **node** $s \in N$ of a graph corresponds to a single **world state**. A **labeled edge** $(s_1, s_2) \in E$ with a label a then means, that the execution of **action** a in a world state s_1 transitions the world into a state s_2. Existence of such a node also means, that the action a is executable in s_1. Any **path** within such graph leading from an initial node s_0 to one of the nodes corresponding to a goal state represents a resulting plan.

During the search, we traverse the graph node by node. We don't have enough space for an extensive description of our algorithms (see the full version of this paper [Čertický, 2011]), but the following pseudocode of main reasoning function *expandNode* should should give you a rough idea.

```
def expandNode(S,BK,INST):
        children = $\emptyset$
        for $\forall$ action instances $A \in$ INST:

                # get the constraint representation
                # of all the preconditions of A
                PC = getPreConstraints(A)

                # encode a background knowledge to a meta-program
                # form required for our reasoning
                BK = getMetaBK(BK)

                # compute new world states
                # using ASP solver
                NewS = applyEffects(S,A,PC,BK)

                # add these new states to a set of results
                children = children $\cup$ NewS

        return children
```

Fig. 1. Function **expandNode(S,BK,INST)** generates the children of the node S within a search graph by encoding two consecutive states, action instance, and background knowledge to logic meta-program and using ASP solver.

[1] See the chapter 11 of [Russel-Norvig, 2003] for examples of planning heuristics.

3 Conclusion

The *decomposition trick* improves the performance ASP-based planning in complex domains, while preserving and utilising the expressive capabilities of ASP semantics. In order to demonstrate the speedup, we have created an experimental planner called *GRASP* (Graph-search based ASP Planner). It can be found at http://grasp-planner.net23.net/. For the description of experiments, see the full version of this paper.

We encourage the authors and programmers of mainstream ASP planning systems to use this *decomposition trick* in their future implementations.

References

[Čertický, 2011] M. Čertický. *Enhancing ASP-based Planning by Heuristic Graph-search Techniques.* Technical Reports, Comenius University, Bratislava. 2011.

[Eiter et al., 2000] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres. *Planning Under Incomplete Knowledge.* Lecture Notes in Computer Science, Volume 1861: 807-821. 2000.

[Eiter et al., 2002] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres. *The DLV^K Planning System: Progress Report.* Lecture Notes in Computer Science, vol. 2424/2002: 541-544. 2002.

[Gebser et al., 2011] M. Gebser, R. Kaminiski, M. Knecht, T. Schaub. *plasp: A Prototype for PDDL-Based Planning in ASP.* In Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11), LNAI 6645: 358-363, Springer Verlag. 2011.

[Hanks-McDermott, 1987] S. Hanks, D. McDermott. *Nonmonotonic logic and temporal projection.* Artificial Intelligence, 33(3):379-412. 1987.

[Hart-Nilsson-Raphael, 1968] P. E. Hart, N. J. Nilsson, B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics SSC4 4(2): 100–107. 1968.

[Leone et al., 2006] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S Perri, F. Scarcello. *The DLV system for knowledge representation and reasoning.* ACM Transactions on Computational Logic (TOCL), Volume 7, Issue 3. 2006.

[Lifschitz, 2002] V. Lifschitz. *Answer Set Programming and Plan Generation.* Artificial Intelligence, vol. 138. 2002.

[Nilsson, 1980] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto. 1980.

[Russel-Norvig, 2003] S Russell, P Norvig. *Artificial Intelligence: A Modern Approach, 2nd edition.* Prentice Hall. 2003.

[Simons, 2002] P. Simons, I. Niemela, and T. Soininen. *Extending and Implementing the Stable Model Semantics.* Artificial Intelligence, 138(1-2):181–234. 2002.

[Slaney-Thiebaux, 2001] J. Slaney, S. Thiebaux. *Blocks World revisited.* Artificial Intelligence 125: 119-153. 2001.

[Subrahmanian-Zaniolo, 1995] V. S. Subrahmanian, C. Zaniolo. *Relating Stable Models and AI Planning Domain.* In proceedings of ICLP-95. 1995.

[Syrjänen, 2001] T. Syrjänen. *The Smodels System.* Logic Programming and Nonmotonic Reasoning. 2001.