HPGP: An Abstraction-based Framework for Decision-Theoretic Planning

Letícia Friske and Carlos Henrique Costa Ribeiro

Divisão de Ciência da Computação - Instituto Tecnológico de Aeronáutica
12228-900 São José dos Campos – Brasil
{leticia, carlos}@ita.br

Abstract. This paper is a report on research towards the development of an abstraction-based framework for decision-theoretic planning. We make use of two planning approaches in the context of probabilistic planning: planning by abstraction and planning graphs. To create abstraction hierarchies our planner uses an adapted version of a hierarchical planner under uncertainty, and to search for plans, we propose a probabilistic planning algorithm based on Pgraphplan. The article outlines the main framework characteristics, and presents results on some problems found in the literature. Our preliminary results suggest that our planner can reduce the size of the search space, when compared with Pgraphplan, hierarchical planning under uncertainty and top-down dynamic programming.

Keywords: Probabilistic Planning, PGraphplan and Dynamic Programming.

1 Introduction

Robust plan creation in probabilistic domains is a complex problem that can use a variety of algorithms and techniques. Recent research in probabilistic planning is focused on MDPs [1] and Dynamic Programming [2] or in state-space search methods. An interesting proposal describes probabilistic problems on a Probabilistic Planning Domain Definition Language - PPDDL [3], so that can it be solved as a STRIPS planning problem. One limitation in this sense is that most existing STRIPS planners are for deterministic domains, forcing these solutions to be adapted for operation under uncertainties.

Some planners, like Buridan [4], Weaver [5], C-SHOP [6], Drips [7], PGraphPlan [8, 9] and TGraphPlan [8], Paragraph [10] and Prottle[11] propose extensions to STRIPS classical planners, to handle situations where action effects are probabilistic.

The Buridan planner uses partial-order planning to build plans that probably achieve a goal. Weaver is a planner that can handle uncertainty about actions taken by external agents, more than one possible initial state and non-deterministic outcomes of actions. It produces conditional plans and computes the plan’s probability of success automatically, through a Bayesian belief net. C-SHOP extends the classical hierarchical planner SHOP [12] to act in situations with incomplete and uncertain information about the environment and actions. Drips combines conditional planning with probabilistic distribution of effects, and accomplishes abstraction of probabilistic operators with common pre-conditions and effects. This mechanism makes it possible that planning happens in different hierarchical abstraction levels. Pgraphplan and Tgraphplan are extensions of the classical planner Graphplan to probabilistic planning. Paragraph and Prottle extends the Graphplan framework for concurrent probabilistic planning. They use the planning graph primarily for computing heuristic estimates for a forward search.

Unfortunately, the inclusion of probabilistic effects causes an explosive growth in the state-space search and dynamic programming methods. This leads us to investigate approaches that efficiently deal with probabilistic operators and large problems, which is the focus of this work. We consider two strategies to reduce the search: planning by abstraction [13] (e.g. ABSTRIPS [14], Alpine [15], HW [16]) and planning graphs [17], because both demonstrate significant reductions in the search space when compared with other search strategies.

We then propose a hierarchical probabilistic planner, named HPGP – Hierarchical Probabilistic Graphplan. HPGP automatically builds its abstraction hierarchy and executes the hierarchical control planning based on the hierarchical planner HIPU [18]. The search is based on an adaptation of PGraphplan [8].

In this article, we outline the main characteristics of the abstraction-based framework for decision-theoretic planning, and present case studies in the Blocks World problem and in the Flat-Tire domain, that are classical examples for autonomous manipulation planning. Manipulation planning is concerned with handling objects, e.g., to build assemblies. Actions include sensory-motor primitives that involve forces, touch, vision, range detection, and other sensory information. A plan might involve picking up an object from its marked sides, returning it if needed, inserting it into an assembly, etc [19]. The remainder of this paper is organized as follow. Section 2 presents basic definitions on planning and the notation that will be adopted along the article. Section 3 provides some introductory knowledge about Graphplan and Pgraphplan. Section 4 presents the automatic generation of abstraction hierarchies in HIPU and demonstrates the resolution of hierarchical problems. Section 5 discusses the HPGP framework. Empirical results are presented in Section 6. Finally, Section 7 proposes future work and concludes this paper.

2 Planning in Uncertain Environments

One of the common uncertainties that an agent can find regards the actions, which can have stochastic effects. In other words, the next environmental state is expressed as a probability distribution over states. Planning problems with this kind of uncertainties have been approached in Artificial Intelligence through adaptation of problems of classical planning and high-level languages, such as STRIPS, to act in uncertainty domains.

The adaptations proposed here modify the definition of the classical planning problem, specifically regarding the operators. The space problem is defined by a set of probabilistic operators, where each operator consists of preconditions and one or more subsets of effects. For each subset of effects there is an associated occurrence probability $\wp_i(E^i_\alpha)$, such that $0<\wp_i(E^i_\alpha) \leq 1$ and $\Sigma_i^N \wp_i=1$. Figure 1 shows a probabilistic operator that has three subsets of effects (arrow destinations correspond to subsets). Each subset has add-effects and del-effects (propositions) that describe the state modifications generated by the application of the operator with an associated occurrence probability.

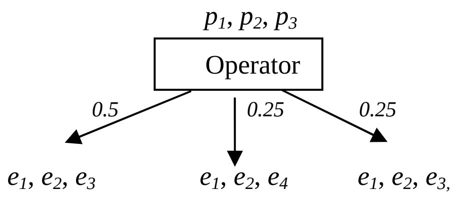

Fig. 1. Generic probabilistic operator

3 Graphplan Framework

The Graphplan planner [17] defines an efficient method of solving classical STRIPS problems. Graphplan is based on compiling a planning problem into a polynomial-size structure called *planning graph*. A planning graph is a layered graph alternating between proposition nodes layers and action nodes layers. Each level represents the union of what might be reachable at a given time step. The initial level consists of proposition nodes that represent the initial conditions. The next level of the graph has a node for each operator that might possibly be performed at the corresponding time step. Special actions are used to represent the persistence of a proposition from one time step to the next. The second proposition level consists of the add-effects of the first action level. Successive levels are generated by inference according to action preconditions and effects. Edges in a planning graph connect actions to their preconditions and their add and delete effects.

During graph building, the graph retains binary mutual exclusion information, indicating when two actions or propositions can not exist simultaneously. This information can be used to prune the search. Graphplan begins by creating a planning graph from the initial conditions until all the goals appear in the graph and none are pairwise mutually exclusive, then searches the planning graph for the plan solutions using a backward chaining search.

3.1 PGraphplan

A structural change to the standard planning graph is required in order to accommodate probabilistic actions. In this subsection we will describe this changes and the search procedure for PGraphplan [8, 9].

We consider probabilistic operators, such as presented in section 2. To support these kinds of operators, the graph is constructed in the normal manner except that each outgoing edge of an action has associated with it the event (subset of effects) which produced the edge.

3.2 Searching the Probabilistic Planning Graph

Given a bounded number of time steps tmax, Pgraphplan builds the planning graph and then performs a finite-horizon dynamic programming to find the optimal *tmax*-step contingent plan [9]. The optimal plan corresponds to the plan of highest expected utility. The utility function adopted in this paper corresponds to finding the plan with the highest probability of success. The complete algorithm and more details about Pgraphplan search can be found in [8, 9].

To speed up the search, Pgraphplan propagates two distinct kinds of information through the graph. Both kinds of information are used to tell the planner when the path it is currently exploring provably cannot reach the goal within the given time horizon and therefore it may safely return failure in its recursive calls [8]. The two types of information are summarized below.

1. Removing unneeded vertices: a "needed" node is the node from which there exists at least one path to the goal. Pgraphplan removes from the planning graph all nodes that do not have any paths to the goal literal (unneeded nodes). This occurs as follow. Are removed from the last propositional level (*tmax* level) of the graph, all propositions that not are in the goal. Now, consider the actions at *tmax*-1 action level. If all add-effects of the one action were removed, then that action too is unneeded and can be withdrawn from the graph. Working backwards, any proposition at time *tmax*-1 that has no out-edges can be removed as well, and so on.
2. Value propagation: the idea of value propagation is to propagate heuristic values (*hvalues*) through the nodes of the graph such that the heuristic value of any state *S*, defined to be the sum of the *hvalues* of the nodes of the states, is guaranteed to be greater than or equal to the true value of *S*. If the planner finds that the heuristic value of the current state at time *t* is less than *t*, then it can backtrack. This means that it cannot possibly reach the goal by time *tmax*.

4 HIPU - Hierarchical Planner under Uncertainty

HIPU – Hierarchical Planner under Uncertainty [18] is an extension of the Alpine method [15], adapted to act under uncertainty conditions. Regarding uncertainties, it allows a probabilistic distribution of possible operator effects and a probabilistic choice under possible initial states of the domain. HIPU automatically generates an abstraction hierarchy and plan from this hierarchy. Planning begins at the highest level of abstraction, and the solution found in this level is refined by lower levels. During refinement, the plan found so far is evaluated, verifying if the solution succeeds with probability equal or higher than a predefined value.

4.1 Generating Abstractions in HIPU

The starting point for generating abstraction hierarchies in HIPU is Algorithm 1. This algorithm establishes the possible interactions among literals, creating a graph of constraints that will be used in the creation of the abstraction hierarchy.

```
Function Find-Constraints(graph,operators,goals):
Input: The operators of the apace problem and the goals of a problem.
Output: constraints to guarantee ordered monotonicity for the problem.
For each literal in the goals do
If not(Constraint-Determinated(literal, graph))
Constraint-Determinated(literal,graph)←TRUE;
  For each operator in Operators do
        Subset_relevant ←FALSE;
        For each Subset_Effects(operator) do
          If literal in Subset_Effects (operator) do
            Subset_Relevant←TRUE;
            For each Effect in Subset_Effects (operator) do
            Add_Directed_Edge(literal,Effect, graph);
        If (Subset_Relevant)
            preconds←Preconditions(operator);
            For each precondition in Preconds do
              Add-Directed-Edge(literal, precondition, graph);
            Find_constraint(graph, operators, Preconditions);
return(graph);
```

Algorithm 1. Algorithm for determining constraints

After creating the graph of constraints, we find the strongly connected components using depth-first search. The next step constructs a reduced graph where the nodes that comprise connected components in the original graph correspond to a single node in the reduced graph. Literals within a node in the reduced graph must be placed in the same abstraction space and the connections between nodes define a partial order. The partial order is transformed into a total order using a topological sort. The total order graph originates the abstraction hierarchy, each level of the total order graph corresponding to a hierarchical level.

4.2 Hierarchical Planning Solver

Given a hierarchy of abstractions, HIPU proceeds as follows. First the problem solver maps the original problem to the highest level of the hierarchy (level *i*) by deleting literals from the initial states, goal states and operators literals that are not relevant to the abstraction level. The planner then performs a depth-first search adapted to probabilistic operators to find a solution for that level. The solution found in level *i* will be used in the next abstraction level (level *i*-1), where the literals of the intermediate states serve as sub goals at level *i*-1. The problem solver then solves each of the intermediate subproblems, using the final state of one subproblem as initial state for the next subproblem. The process is repeated until the plan has been refined in all hierarchical levels.

During the search, when a plan that satisfies the goal state is found, the probability that the current plan will achieve it is computed using the *forward assessment algorithm* [4]. If the probability is high enough, then the plan is a solution and planning terminates successfully, otherwise the planner continues, choosing a new state to expand or returning fault.

5 HPGP – Hierarchical Probabilistic Graphplan

The main difference between HIPU and HPGP is that instead of depth-first search, HPGP uses PGraphplan to find plans. Some modifications are necessary so that PGP can handle the hierarchical planner HIPU.

The first adaptation to embed PGP into the HIPU framework regards the maximum time steps (*tmax*) for the plan search. In PGP the number of time steps is previously defined by the user. This requires knowledge about the domain of the problem or exhausting attempts until that ideal number is found. As HIPU is totally automated, we need to discover a manner to automatically define *tmax*. We propose an alternative using the planning graph. First we expand the probabilistic graph until all the goals appear in it. While the goal state is not reached, the graph expansion continues until it is leveled (two consecutive levels are identical), or a maximum steps number (a system attribute) is reached. When a graph is leveled without reaching all the goals, it means that no solution exists. The next step is to remove from the graph all unneeded vertices and attribute to *tmax* the number of different actions (minus the persistent ones) existing in the graph. Although this heuristic seems reasonable, it can be conservative in specific cases, because the idea is to define *tmax* as the minimum number of operators necessary to reach the goal state. For example, consider the Blocks World problem presented in Appendix A. The operator *paint_block* reaches the goal "painted block" with probability 0.8, but there exists a probability of finishing the paint with the block not painted. However, if we carry the paint and try to paint the block again, the probability of concluding the task with success increases. Certainly, these two new steps (operators) will not be included in the plan if we expand the graph until the level of the goal state level is reached. Another situation in which the heuristics can be conservative is when an operator removes preconditions of other operators, making it necessary to include or repeat actions. But this situation happens because the probabilistic graph plan is a relaxed model (that ignores delete effects), and not exactly a consequence of the heuristics adopted for the *tmax* calculation. This problem could be solved with the inclusion of mutual exclusion relations in the graph, as suggested in [20]. Nevertheless, in this paper we ignore this improvement. When the method to define *tmax* fails during planner execution, the user will be required to inform *tmax*, or it will be given the maximum value allowed.

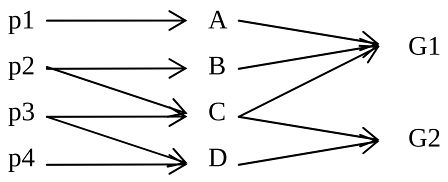
$$\begin{tikzcd}
p1 \arrow[r] & A \arrow[dr] & \\
p2 \arrow[r] \arrow[dr] & B \arrow[r] & G1 \\
p3 \arrow[r] \arrow[dr] & C \arrow[ur] \arrow[dr] & \\
p4 \arrow[r] & D \arrow[r] & G2
\end{tikzcd}$$

Fig. 2. Operators with redundant effects.

Contrasting with previous cases (where the heuristic is conservative), we now discuss a situation in which the heuristic will overestimate *tmax*, causing a search space augmentation. The value *tmax* would be overestimated if there are redundant propositions in the operator effects. An example is illustrated in Figure 2. The graph showed has four actions (*A*, *B*, *C* and *D*), three of these (*A*, *B* and *C*) reach a goal literal G1 and two (*C* and *D*) goal G2. In the plan, just one of them will be really executed (action *C*). As each action has add-effects linked to a goal literal, none of them will be deleted during removal of the unneeded nodes, causing a increase (from 1 to 4) in the number of time steps (operators) to reach the goal state. This problem would be attenuated by considering only one operator when more than one reaches the same propositions.

A key difference between HIPU and PGP is the type of proposed solution. HIPU solves the problem by finding a sequence of operators (concatenation of partial plans) that reach the goal state with an associated probability of success. PGP creates an optimal contingent plan (an optimal finite-horizon policy), i.e., a mapping from states to operators with the goal of maximizing an expected utility for some utility function (probability of success). Those search models are incompatible. We now propose two alternatives to solve this problem, as follows.

The first alternative modifies HIPU to search contingent plans and not sequential plans. This implies refining all the valid plans found during the search. However, working with contingent plans would cause an increase in the plan space, and we cannot guarantee that it will lead to a significant reduction through hierarchization. We consider then a second solution: to refine the plan with the highest success probability. If the plan can't be completely refined, the second plan with highest probability of success will be chosen, and so forth. Algorithm 2 summarizes HPGP.

```
Performs HPGP (Initial State, Goal State, Operators):
 1.Generate the abstraction hierarchy exactly as HIPU.
 2.Create the abstraction spaces for each hierarchical level by deleting from the initial state, goal
     states and operators propositions that are not relevant to the abstraction level.
   For each hierarchical level and respective subproblems:
  3.  Produce the probabilistic planning graph.
  4.  Remove unneeded nodes.
  5.  Calculate the number of actions in the graph and attribute the value to tmax.
  6.  If it is not possible to reach the goal state in the graph, backtrack (return fail), else continue.
  7.  Reinsert the temporary excluded nodes in the graph.
  8.  Continue the expansion until the level tmax is reached.
  9.  Exclude unneeded nodes and propagate heuristic values.
 10.  Perform a finite-horizon dynamic programming to find the optimal contingent plan (see
      complete algorithm in [8]).
 11.  Choose the plan with highest probability of success.
 12.  Refine the plan found (refinement is the same performed by HIPU, presented in Subsection
      4.2) executing steps 3 to 12 for each subproblem.
```

Algorithm 2. High level HPGP algorithm.

6 Empirical Results

HPGP was partially implemented. The code was generated in C (for planning) and Lisp (for hierarchy determination). We performed experiments for the Blocks World and Flat-Tire Domains (Appendix A), and compared results with Pgraphplan, HIPU, and standard top-down Dynamic Programming TDDP [8]. HIPU executes standard top-down Dynamic Programming for searching plans, and considers the same time horizon (*tmax* values) as HPGP.

For the first case study (Blocks World Domain), the abstraction hierarchy generated by HIPU (or HPGP) contains two levels and the respective literals: Level 1 - Paint and color(x), and Level 2 - on(x,y), onTable(x), clear(x) and clear(y). Numerical results are in Table 1. The number of states of HIPU and HPGP is the sum of states generated at each abstraction level.

For the experiments presented in Table 1, *tmax* was automatically calculated using the heuristic discussed in Section 5. Table 2 shows results with a manual definition of *tmax*. Only Pgraphplan and top-down Dynamic Programming (contingent planners) were considered in these experiments. Results are compared in terms of the number of states searched and the probability of success of the generated plans.

Table 1. Results in the Blocks World Problem, 4 and 7 blocks, t*max* automatically generated.

Planner	**States**		**Probability**	
	4 blocks *tmax* = 9	7 blocks *tmax*=18	4 blocks	7 blocks
HIPU	224	123469	0.4096	0.209715
HPGP	90	56037	0.4096	0.209715
Pgraphplan	787	146652	0.4096	0.209715
TDDP	12507	>33600000	0.4096	0.209715

Table 2. Results in the Blocks World Problem, 4 and 7 blocks, t*max* user-defined.

Planner	**States**		**Probability**	
	4 blocks *tmax*= 12	7 blocks *tmax*=22	4 blocks	7 blocks
Pgraphplan	7547	20229508	0.73728	0.738198
TDDP	23164	Timeout	0.73728	Timeout

HPGP empirically searches through less states than HIPU. Both hierarchical planners have advantages (in search states) when compared with Pgraphplan and TDDP. Contingent planners (Pgraphplan and TDDP) are able to find plans with higher probability of success when we increase the number of time steps (*tmax)*, but with a higher cost (number of states searched).

Table 3. Results in the Flat-Tire domain, t*max=14*, automatically calculated.

Planner	**States**	**Probability**
HIPU	2657	0.8145
TDDP	6350	0.8145
HPGP	1213	0.8145
Pgraphplan	2886	0.8145

Experiments realized in the Flat-Tire domain are showed in Table 3. Hierarchization divided the problem in two hierarchical levels, with respective literals: Level 1 - on(nut,hub), on-ground(nut), free(hub), in(wheel,container), have(wheel), on(wheel,hub). Level 2 – open(container), close(container). The number of time steps *tmax* was automatically calculated, it is the same (14) for all planners.

HIPU and TDDP use the same search strategy, however, the hierarchical process (HIPU planner) produces better results than the flat planner (TDDP). HPGP was significantly better (less search states) than HIPU and Pgraphplan.

7 Conclusions and Future Work

This paper presented HPGP, an abstraction-based framework for probabilistic planning. The abstraction hierarchy is automatically generated by a hierarchical planner under uncertainty, and the plan search proposed here is an extension of the Pgraphplan planner, adapted to handle hierarchical planning. We proposed a heuristic to automatically generate time steps (*tmax*) to finite-horizon search.

We related preliminary experimental results on the Block's World and Flat-Tire Domains. An analysis of the results demonstrates that HPGP can reduce significantly the space search in probabilistic planning. However, to guarantee that the heuristic that automatically defines *tmax* is viable would be premature. We prefer to leave the definition of the number of steps as an open question until improvements (some of which are proposed in this article) are implemented.

Proposals for future work are adapting HPGP to search contingent plans and executing Tgraphplan [8] to search plans in our hierarchical framework. Tgraphplan finds the highest probability trajectory from the start state to the goal, and produces potentially sub-optimal policies. We are working at the implementation of a future Hierarchical Tgraphplan Planner (HTGP). Likewise, a deeper evaluation of HPGP performance through comparisons with other planners and tests on other domains are activities for future research.

We also suggest using different search strategies and heuristics to increase the hierarchical planner efficiency.

References

1. Geffner, H., Bonet, B.: Solving Large POMDPs Using Real Time Dynamic Programming. Working Notes Fall AAAI Symposium on POMDPS (1998)
2. Bonet, B., Geffner, H.: Planning and Control in Artificial Intelligence: A Unifying Perspective. Appl. Intell. 14(3) (2001) 237-252
3. Younes, H.L.S., Littman, M.L.: PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Pittsburgh (2004)
4. Kushmerick, N., Hanks, S., Weld, D.: An Algorithm for Probabilistic Planning. Artificial Intelligence. vol. 76, (1994) 239-286
5. Blyte, J.: Planning Under Uncertainty in Dynamic Domains. Doctor Thesis. Carnegie Mellon University. Pittsburgh, PA, May (1998)
6. Bougerra, A., Karlsson, L.: Hierarchical Task Planning under Uncertainty. In 3rd Italian Workshop on Planning and Scheduling. Perugia, Italy (2004)
7. Doan, A., Haddawy, P.: Decision-Theoretic Refinement Planning: Principles and Application. Technical Report TR 95-01-01, February (1995)
8. Blum, A.L., Langford, J.C.: Probabilistic planning in the graphplan framework. In proceedings EPC (1999) 319-332.
9. Blum, A.L., Langford, J.C.: Probabilistic planning in the graphplan framework. In AIPS98 Workshop on Planning as Combinatorial Search, 8-12, June (1998)

10. Little, I., Thiébaux, S.: Concurrent Probabilistic Planning in the Graphplan Framework. In proceedings: 16th International Conference on Automated Planning and Scheduling - ICAPS-06 (2006)
11. Little, I., Aberdeen, D., Thiébaux, S.: Prottle: A probabilistic temporal planner. In Proc. of the 20th American Nat. Conf. on Artificial Intelligence - AAAI'05 - (2005)
12. Nau, D., Cao, Y., Lotem, A., Munoz-Avila, H.: Shop: Simple hierarchical ordered planner. In Procs of the Int. Joint Conference on AI (1999) 968-973
13. Giunchiglia, F., Villaorita, A., and Walsh, T.: Theories of abstraction. AI Communication, 10(3-4):167-176 (1997)
14. Knoblock, C.: An analysis of Abstrips. In Artificial Intelligence Planning Systems: Proceedings of the first international conference (1992)
15. Knoblock, C.: Automatically generating abstractions for planning. Artificial Intelligence, vol. 68(2), p. 243-302 (1994)
16. Armano, G., Cherchi, G., Vargiu, E.: A Parametric Hierarchical Planner for Experimenting Abstraction Techniques. IJCAI 2003: (2003) 936-941
17. Blum, A., Furst, M.: *Fast Planning through Planning Graph Analysis*. Artificial Intelligence, 90 (1997) 281-300
18. Friske, L.M., Ribeiro, C.H.C.: Planning Under Uncertainty with Abstraction Hierarchies. Lecture Notes in Computer Science, vol. 4224, (2006) 1057-1066
19. Ghallab, M., Nau, D., Traverso, P.: Automated Planning Theory and Pratice. Morgan Kaufmann Publishers. ISBN 1-55860-856-7, (2004).
20. Wenxiang, G., Rixian, L., Huajie, O., Minghao, Y.: An Improved Probabilistic Planning Algorithm Based on Pgraphplan, Proceedings of the Third International Conference on Machine Learning and Cybernetics. vol. 4 (2004) 2374- 2377

Appendix A

Blocks World Domain augmented with probabilistic *colorblock* operator. This domain consists of a set of cubic blocks sitting on a table. The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block, paint the block, and move it to another position, either on the table or on top of another block. The arm can only pick up one block at time, so it cannot pick up a block that has another one on it. The goal will always be to build one or more stacks of painted blocks, specified in terms of what blocks are on top of what other block. This domain has four operators, as follow:
Op: stack (*x y*), Preconds: clear (x), clear (y), on-table (x), Effects: on (x y), ~clear(y), ~on-table(x)
Op: unstack (*x y*), Preconds: on(x y), clear (x), Effects: on-table(x), clear(y), ~on(x y)
Op: colorblock(*x*), Preconds: clear(x), on-table(x), paint, no-color(x), Effects: 0.8(color(x), ~no-color(x)), 0.2(~paint, no-paint)
Op: chargepaint, Preconds: no-paint, Effects: paint, ~no-paint.

Flat-Tire Domain considers the problem of fixing a flat tire. The domain has the following eight action schemata:
Op: open(*x*), Preconds: closed(x), Effects: ~closed(x), open(x)
Op: close(*x*), Preconds: open(x), Effects: ~open(x), closed(x)
Op: fetch (*x y*), Preconds: in(x y), open(y), Effects: ~in(x y), have(x)
Op: put-away(*x y*), Preconds: have(x), open(y), Effects: in(x y), ~have(x)
Op: remove-wheel(x y), Preconds: on(x y), on-ground Nut1, on-ground Nut2, on-ground Nut3, on-ground Nut4, Effects: have(x), free(y), ~on(x y)
Op: put-on-wheel(x y), Preconds: have(x), free(y), Effects: on(x y), ~free(y), ~have(x)
Op: remove-nut(x y), Preconds: on(x y), Effects: 0.95(~on(x y), on-ground(x))
Op: fixit-nut(x y z),Preconds: on-ground(x), on(z y), Effects: ~on-ground(x), on(x y)