A Graph-based Approach for POCL Planning[1]

Laura Sebastia, Eva Onaindia and **Eliseo Marzal**[2]

Abstract. In this paper, we show that despite the great success of some planning approaches, partial-order planning is still an efficient and valid approach for tackling planning problems. The goal of this paper is to show that the effort needed by of a partial-order planner (POP) to solve a problem can be dramatically reduced. By properly exploiting the problem knowledge, it is possible to obtain an approximate plan which is afterwards used as initial plan input to a POP. This plan will always contain actions which must necessarily appear in a valid solution and, therefore, the task of the POP will be simply to add the missing actions thus leading to a significant reduction in search space. In this paper, we will focus on the modifications achieved on a standard partial-order planner to adapt it to this new planning approach.

1 INTRODUCTION

Over the last few years, the use of Partial-Order Causal Link (POCL) planners has been relegated in favour of more efficient plannning approaches such as Graphplan [3] or SATPLAN [8]. This new tendency is totally justified as it has been shown that both approaches highly outperform POCL planners. However, a recent study has shown that POCL planning can compete with other planning approaches. Specifically, UCPOP [1] has shown to outperform BlackBox [9], IPP [10], SGP [14] and STAN [5] for certain domains [11]. Consequently, there is no planner that has yet demonstrated to be superior.

The main limitation of POCL planning is well known: the size of the search space in a plan generation process can be very large. However, Graphplan-like expansion can also be so great as to make the problem insolvable [11]. We have tested BlackBox v3.6 [2] and IPP v4.0 [7] on large *blocksworld* problems and both were unable to solve problems involving more than fifteen blocks.

Moreover, recent works on planning are devoted to the design of new techniques to exploit domain knowledge for efficient planning [6]. This is a very relevant issue as the planning process can benefit from a significant search space reduction.

In this paper, we present a new planning system, GPOP (Graph analysis Partial Order Planner). We use a preprocessing technique based on graph analysis which is able to obtain an approximative graph. This basic graph will comprise a set of actions that belong to a valid solution plan. From this basic graph, the system builds a basic plan, which is used as input to a partial order planner (BP-POP, Basic Plan Partial Order Planner). BP-POP works on this basic plan, which:

1. can contain ALL of the actions needed to solve the problem. In this case, BP-POP only has to sort them.
2. can contain LESS actions than needed to solve the problem. In this case, BP-POP adds the missing actions.

The following section gives an overview of the creation of the basic graph. Section 3 shows the process to obtain a basic plan from a basic graph. Section 4 focuses on the modifications achieved on a standard partial order planner in order to adapt it to this process. Section 5 shows the results we have obtained and, finally, section 6 draws conclusions from this work and summarizes some directions for further work.

2 PREPROCESSING TECHNIQUE

Our preprocessing technique consists of two stages:

1. First, a *problem graph* (PG) is generated. This graph contains two types of nodes (literals and actions) and two types of edges (precondition and add edges). It is organized in levels, alternating action levels and literal levels. This is not a planning graph [3] since this process does not take into account the delete effects of the actions. Thus, mutual exclusion relationships among nodes are ignored. Another difference is that an action level does not stand for a time step, but for a instantiation step: an action level contains all action instantiations which are applicable from the previous literal level and do not appear in a previous action level, ignoring the possible incompatibilities among nodes.
2. Afterwards, the process obtains a more refined graph which only contains a subset of actions which necessarily appear in a correct solution. This set of actions forms the *basic graph*.

2.1 Obtaining a Basic Graph

The *basic graph* (BG) is a directed, layered graph of action nodes, with two more levels than the PG, both comprising only one action. The former contains an initial action A_0 having only effects (the literals in the initial situation). The latter includes a final action A_n with only preconditions (the goal literals). An edge or causal link $A_i \rightarrow^p A_j$ denotes that the precondition p of A_j is solved by means of an add effect of A_i.

The backward chaining process starts with the preconditions of A_n and attempts to find a set of actions in the same or any previous action level having these goals as add effects. The preconditions of these actions form a new set of subgoals and this operation is repeated until each literal has been processed.

In order to find a consistent causal link for each subgoal, the search method applies the following property:

Property 1 (literal consistency) *A literal p required by an action A_k ($p \in \mathsf{Pre}(A_k)$) is said to be consistent if these two requirements hold:*

[1] This work has been partially funded by the Spanish government CICYT-FEDER project 1FD7-0887

[2] Dept. Sistemas Informaticos y Computacion, Technical University of Valencia, Valencia, Spain, email: {lstarin, onaindia, emarzal}@dsic.upv.es.

- *There is a sequence of actions $A_i \rightarrow A_{i+1} \ldots A_{k-1} \rightarrow A_k$ such that $p \in \mathsf{AddEff}(A_i)$ and $p \notin \mathsf{DelEff}(A_j)$ $\forall j \in [i+1, k-1]$.*
- *For each action A_i such that $p \in \mathsf{DelEff}(A_i)$ there is a sequence $A_i \rightarrow A_{i+1} \ldots A_{k-1} \rightarrow A_k$ with an action $A_j, j \in [i+1, k-1]$, such that $p \in \mathsf{AddEff}(A_j)$.*

Definition 1 (Basic Graph) *Let $\mathcal{S}$ be the set of actions that constitute a solution plan for a given problem. A* basic graph *is a pair of elements $\mathcal{G}(\mathcal{N}, \mathcal{E})$, where:*

- *$\mathcal{N}$ is the set of graph nodes, so that $\mathcal{N} \subseteq \mathcal{S}$.*
- *$\mathcal{E}$ is the set of graph edges that represent causal links between the actions in $\mathcal{N}$. Suppose that $n_i, n_j \in \mathcal{N}$ and p is a precondition of n_i. $e(n_j, n_i, p) \in \mathcal{E}$ if and only if p is an add effect of n_j and it is used to achieve the precondition p of n_i.*

The achievement of the literal consistency property guarantees that, when a problem is solvable, all of the actions in the BG belong to a correct solution. However, it is not possible to prove that such a solution exists or that the set of actions in the BG yields an optimal solution plan (the shortest). In spite of this, for most of the tested cases, the obtained solution was the optimal one.

Basically, the BG comprises optimal sequences of actions to achieve each subgoal literal independently. When property 1 does not hold for all of the action preconditions it means that some of the subgoal literals cannot be satisfied and, consequently, the BG is not created.

The process to create the BG is very fast as it is only necessary to satisfy property 1 for all of the literals. However, interactions among actions of different sequences [3] are not taken into account and, therefore, it might be impossible to set a logical order for all of the actions in the BG. In this case, the POP must introduce new actions to solve these conflicts between actions. Thus, two different types of basic graphs can be obtained:

- *complete basic graphs*, where $\mathcal{N} = \mathcal{S}$. In this case, a logical order can be established among all the actions in $\mathcal{N}$.
- *incomplete basic graphs*, where $\mathcal{N} \subset \mathcal{S}$. In this case, there are inconsistencies between actions of different sequences. Therefore, the POP would need to add new actions.

3 FROM A BASIC GRAPH TO A BASIC PLAN

In this section, the process of building an input plan to the POCL planner from the generated basic graph is explained. Let $\mathcal{G}(\mathcal{N}, \mathcal{E})$ be a basic graph.

Definition 2 (Basic plan) *A basic plan is a tuple $\Pi(\Lambda, \Sigma, \Theta, \Phi, \Gamma)$, where:*

- *$\Lambda = \mathcal{N}$ is a set of actions that belong to the basic plan.*
- *$\Sigma = \mathcal{E}$ is a set of causal links between the actions in Λ.*
- *Θ is a set of ordering constraints between the actions in Λ, resulted from the causal links in Σ.*
- *Φ is a set of non-satisfied preconditions of the actions in Λ (agenda) which in the case of a basic plan is an empty set.*
- *Γ is a set of conflicts between the actions in Λ. $\gamma(A_n, A_a, A_t) \in \Gamma$ if $A_a \rightarrow^{\rho} A_n \in \Sigma$, ρ codesignates with a negative effect of A_t and A_t can occur between A_n and A_a.*

[3] Two actions are in different sequences when there is no explicit order between them.

As we said above, there are two different types of graphs, complete or incomplete basic graphs. This difference is specially important from the point of view of BP-POP. In the case of complete basic graphs, BP-POP only has to sort the actions in the basic plan, whereas in the case of incomplete basic graphs, some actions have to be added by BP-POP, that is, some of the causal links in Σ need to be modified. In both cases, the remaining operations (ordering between actions and addition of new actions) are detected through the existence of threats in Γ.

If a plan is built using the actions in $\mathcal{N}$, we can assure that this plan will lead to a valid solution provided that such a solution exists, because a POP cannot guarantee terminating on unsolvable problems.

4 BP-POP

BP-POP is based on a POCL planner [13], which has similar characteristics as UCPOP [12]. The main features of BP-POP are the following:

- It is implemented in C.
- It uses a structured planning language based on frames to facilitate the definition of planning domains.
- It uses the concept of variable domains, which permit a reduction of the search space.

BP-POP takes a basic plan instead of an empty initial plan as input. This fact has two main effects:

1. We are able to get dramatical reductions in terms of search space, which is the main drawback in POCL planning.
2. The completeness of a POP is guaranteed when starting from an empty initial plan. However, additional operations are necessary to guarantee the completeness of our BP-POP.

Generally speaking, the planning algorithm for BP-POP is very similar to the planning algorithm for standard POCL planners. However, there are some important differences that we will point out in this section.

4.1 Dealing with threats

4.1.1 Symmetrical threats

Definition 3 (Symmetrical threats) *Let be $\gamma_i(A_{n_i}, A_{a_i}, A_{t_i})$ and $\gamma_j(A_{n_j}, A_{a_j}, A_{t_j})$ two threats in Γ. γ_i and γ_j are* symmetrical threats *if $A_{n_i} = A_{t_j} \wedge A_{n_j} = A_{t_i} \wedge A_{a_i} = A_{a_j}$.*

As we explained above, when the basic graph is being created, the process does not take into account possible interactions between actions in different sequences. This is the reason why some links between actions in the basic graph are incorrect. For example, let's suppose that the system obtains a graph like the one shown in Figure 1. Actions A_1 and A_2 are in different sequences. Therefore, the possible inconsistencies between them are not taken into account while creating the basic graph.

However, when BP-POP builds the basic plan, it is able to detect inconsistencies between A_1 and A_2 by means of the existence of symmetrical threats. That is, in the basic plan in Figure 1, we find the threats:

- $\gamma(A_1, A_3, A_2)$
- $\gamma(A_2, A_3, A_1)$

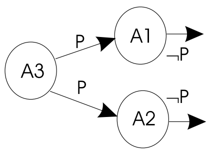

Figure 1. Symmetrical threats

It is obvious that A_1 and A_2 interfere with each other. Therefore, the causal link $A_3 \rightarrow^p A_1$ or the causal link $A_3 \rightarrow^p A_2$ must be modified.

This is one of the main differences of BP-POP with respect to a standard POP. In a POP, the existence of symmetrical threats means that the current plan is inconsistent and it is pruned. In contrast, in BP-POP, it implies the existence of incorrect causal links that need to be restored. This happens when the POP deals with an incomplete graph.

The existence of symmetrical threats represents a severe conflict in the current plan, as they represent an incorrect relationship between two actions. Therefore, symmetrical threats must be solved as soon as they appear in a partial plan.

The technique for solving symmetrical threats is based on the *white-knight* concept developed by Chapman in [4]. Our white-knight technique consists of deleting the threatened causal link and inserting the precondition associated to that causal link into the agenda (which in turn will be satisfied by another existing or new action). The following process is used to solve symmetrical threats:

1. Select one of the symmetrical threats and solve it by promotion and/or demotion
2. Select the other threat and use the white-knight technique (it cannot be solved by using promotion nor demotion)
3. Repeat this process inverting the order in which the threats have been selected

It is clear that all the choices available for restoring the non-correct causal link are taken into account in this process. Whether the correct action to satisfy a literal is an action in the basic plan or a new action, this will be discovered in step 2 when BP-POP generates all of the possibilities available for achieving a precondition (as a standard POP). Thus, the lack of completeness due to the preprocessing technique is recovered during the planning process carried out by the POP by using a white-knight-based technique. Figure 2 shows two plans that would be generated when solving the symmetrical threat in Figure 1 (inverting the threat order selection).

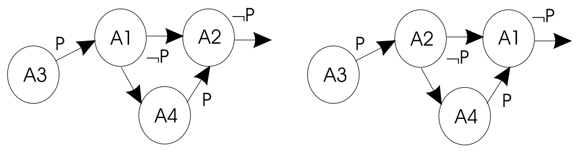

Figure 2. Solving symmetrical threats

In both plans in Figure 2, an extra ordering constraint has been added. Let's focus on the plan in figure 2(left). If we follow the process given above step by step, after step 1, we will have an ordering constraint between A_1 and A_2 ($A_1 < A_2$). As a result of the application of the white-knight technique to restore the causal link between A_2 and A_3, A_4 is added. It is obvious that at that moment, there is a conflict between A_1 and the new causal link between A_4 and A_2. An extra ordering constraint between A_1 and A_4 is added in order to avoid this conflict.

4.1.2 Selection order

We have observed that the system performance depends on the order in which threats are selected to be solved. If we use a logical order to select threats, we will build the correct plan more easily. More specifically, let $\gamma(A_n, A_a, A_t)$ be a threat in Γ. The threats are ordered according to the place that A_n ocuppies in the current plan. That is, the order for solving threats follows the execution order of the actions in the plan. In this way, we assure that the plan is being built correctly from the beginning. In other words, if the system selects γ, it means that there is no threat over predecessor actions of A_n.

4.1.3 Threat resolution

Let $\Pi_b(\Lambda_b, \Sigma_b, \Theta_b, \Phi_b, \Gamma_b)$ be a basic plan and Λ_i be the set of actions of the current plan. We can distinguish two types of actions:

- An action $A_j \in \Lambda_i$ is a *basic action* if $A_j \in \Lambda_b$.
- An action $A_j \in \Lambda_i$ is an *added action* if $A_j \notin \Lambda_b$.

If a threat $\gamma(A_n, A_a, A_t)$ cannot be solved by promotion or demotion, we have to take into account the nature of A_n:

- If it is a basic action and the causal link involved in the threat has not been previously restored, the white-knight technique can be applied. This is a case of modification of a wrong causal link.
- If it is a basic action and the causal link involved in the threat has been restored, the plan is pruned because all the possibilities that exist to satisfy the precondition were taken into account at the moment of the application of the white-knight technique.
- If it is an added action, all the possibilities were considered when solving the corresponding precondition. Therefore, in the case of unresolvable conflict, the plan is pruned.

4.2 Establishing preconditions

As we defined above, a basic plan has no preconditions in the agenda because the preconditions of all of the actions in the basic graph have been solved. However, new preconditions can be added to the agenda as the solution plan is being built. There are two ways of adding preconditions:

- as a result from the application of the white-knight technique, because the precondition associated to the threatened link is added to the agenda to be satisfied.
- once we have preconditions in the agenda, we can choose a new action to solve it. Therefore, its preconditions will be added to the agenda as well.

Preconditions in the agenda have priority over threats, so, if there is a precondition in the agenda, the system will select it in detriment of the threats.

As we said above, BP-POP is building a correct plan starting from the first action. If the system is working on action A_i, any action previous to action A_i has no conflicts, that is, we have a correct subplan. Therefore, when choosing an existing action to solve a precondition of A_i, we can use this situation in the following way:

1. Each action in the plan previous to A_i is assigned a value, depending on the distance between that action and A_i.
2. This value is used to give priority according to the nearness of that action to A_i.
3. It is obvious that the nearest actions have a greater probability of solving a precondition correctly.

For example, in Figure 3, action A_4 attemps to satisfy its precondition p. Both A_1 and A_3 produce p. If we use the distance concept, it is clear than A_3 is better than A_1, since there is a greater probability that there is an action between A_1 and A_4 that negates p. If A_2 negates p, A_1 could not be the producer for A_4.

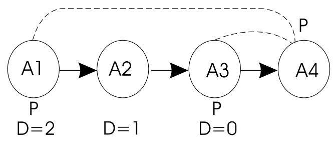

Figure 3. Example. Establishing preconditions

In order to assure completeness, the system will create all possible plans, but it will use the distance between actions to give more priority to the plan that uses the nearest action.

4.3 An example

In this section, we show an example of how GPOP works. Suppose that we have to solve the following problem:

Operators	Preconditions	Effects
O1	*a, b*	*d, ¬a*
O2	*b, c, d*	*e, f, ¬c*
O3	*f*	*g, ¬f*
O4	*f*	*c, ¬f*
O5	*g*	*f*
Initial Situation		*a, b, c*
Goal		*c, g*

After applying our preprocessing technique, the basic plan shown in Figure 4 is obtained. A_0 and A_N stand for initial situation and goal, respectively, A_i is an instantiation of the operator O_i. The following threats will be found:

- $\gamma_1(A_3, A_2, A_4)$
- $\gamma_2(A_4, A_2, A_3)$

As they are symmetrical threats, we use the technique explained above to eliminate that conflict. By solving γ_1 and then γ_2, the plan in Figure 5 is obtained. If γ_2 was solved before, the same plan would be obtained but inverting A_4 and A_3.

The precondition f of A_4 is achieved by inserting a new action A_5, because there is no other action in the plan that can satisfy f for

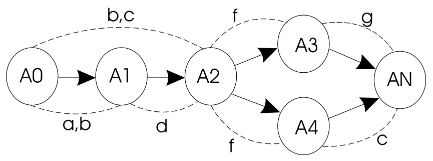

Figure 4. Basic plan

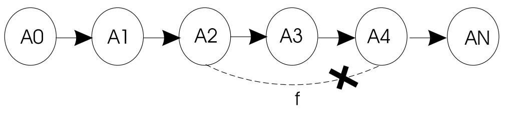

Figure 5. Symmetrical threats solved

A_4. The plan obtained is shown in Figure 6. As A_5 is a new action, its precondition g is inserted into the agenda.

The extra ordering constraint between A_3 and A_5 is added in order to avoid the conflict between them, as we explained above.

At this moment, the precondition g of A_5 is selected. Two plans will be created: one using A_3 and another one inserting a new action. The first one is shown in Figure 7. This plan is the solution to the problem given, because there are no preconditions or threats to be solved.

5 RESULTS

All the planners tested were run on a Sun Ultra 10 machine. We run GPOP, BlackBox v3.6 [2] and IPP v4.0 [7]. The results [4] are classified into two groups: when the obtained basic graph in GPOP was complete (table 1) and when it was incomplete (table 2).

Table 1. Performance (in secs.) of GPOP (complete graph), IPP and BlackBox on different problems

Problem	**GPOP**	**IPP**	**BlackBox**
Sussman	0.006	0.02	0.02
Reversal Tower4	0.011	0.03	0.03
Reversal Tower5	0.033	0.03	0.06
Tower4	0.013	0.03	0.07
Tower5	0.025	0.05	0.21
Tower6	0.046	0.12	0.6
Tower9	0.221	3.7	111
T_LargeA	0.07	0.5	0.82
T_LargeB	0.276	2.21	4.34
T_LargeC	1.446	37.68	–
T_LargeD	3.954	–	–
Flat-tire1	0.006	0.02	–
Flat-tire2	0.006	0.02	0.03
Flat-tire4	0.007	0.01	–

In most of the problems where GPOP was able to obtain a complete graph, the CPU time was reduced by more than 50% compared

[4] We have used a blocksworld domain with 3 operators.

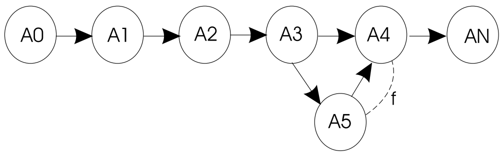

Figure 6. New action added

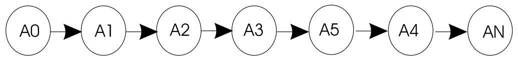

Figure 7. Solution plan

to IPP and BlackBox. For example, in the *blocksworld* domain, as the number of blocks increased, this difference was greater. This is specially noteworthy in TowerLarge problems: GPOP solved the TowerLargeD problem that neither IPP nor BlackBox were able to solve. For those problems with an incomplete graph, GPOP behaved slightly worse than IPP and BlackBox, although this difference was not as significant as in the previous case. As the average and standard deviation results show (Table 3) GPOP behaviour was much more stable.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented our system GPOP. It uses a preprocessing technique based on graph analysis that obtains a basic graph composed of actions that belong to a solution for a given problem. This basic graph is used to build a basic plan, which is the input of a modified POCL planner. This planner will complete that plan (if necessary) in order to obtain a solution plan.

We have shown how the performance of POCL planners can be dramatically improved by using proper preprocessing techniques. We have also shown that GPOP obtains better averages than other planners such as IPP and BlackBox.

This is a first prototype. We are now working on improving the BP-POP performance since the obtained results indicate that our system has a bottleneck in BP-POP. On the other hand, we are also working on adapting the basic graph to problem domains with special features. These domains (logistics-like domains) have several solutions for the same problem, depending on which object is selected to execute an action (for example, which plane is selected to go from city A to city B). We are working on an extended version of the basic graph to arbitrarily select an object. We think that this approach will allow us to obtain nearly all the actions in a valid solution.

As we have explained, our system is not able to detect when a problem is not solvable. Additionally, even though for most of the tested cases GPOP was able to obtain the optimal solution, this cannot be guaranteed. For these reasons, we are working on improving the preprocessing technique in order to obtain basic graphs that will yield optimal solutions or report that the basic graph cannot be obtained when the problem is not solvable.

Table 2. Performance (in secs.) of GPOP (incomplete graph), IPP and BlackBox on different problems

Problem	**GPOP**	**IPP**	**BlackBox**
Hanoi3d	0.176	0.04	0.11
Hanoi4d	1.811	0.13	1.41
Ferry	0.073	0.02	0.04
Monkeyt1	0.079	0.03	0.11
Monkeyt2	0.212	0.04	0.26
Flat-tire3	0.026	0.03	0.1
Fixit	0.405	0.03	–

Table 3. Average and Standard Deviation of GPOP, IPP and BlackBox of solved problems

	GPOP	**IPP**	**BlackBox**
Average	0.454	2.238	7.034
Standard Deviation	1.017	8.393	26.812

REFERENCES

[1] A. Barret, D. Christianson, M. Friedman, K. Golden, S. Penberthy, Y. Sun and D. Weld. *UCPOP v4.0 user's manual*, Technical Report TR 93-09-06d, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA, (1996).

[2] BlackBox Group, *The BlackBox planner*, http://http://www.research.att.com/ kautz/blackbox/, (1999).

[3] A. Blum and M. Furst, *Fast planning through planning graph analysis*, Artificial Intelligence, **90(1-2)**, 281–300, (1997).

[4] D. Chapman, *Planning for Conjuntive goals*, Artificial Intelligence, **32-3**, 333–377, (1987).

[5] M. Fox and D. Long, *The automatic inference of state invariants in TIM*, JAIR, **9**, 367–421, (1998).

[6] M. Fox and D. Long, *The Detection and Exploitation of Symmetry in Planning Domains*, Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 556–560, (1999).

[7] IPP Group, *The IPP planner*, http://www.informatik.uni-freiburg.de/ koehler/ipp.html, (1999).

[8] H. Kautz and B. Selman, *Pushing the envelope: Planning, propositional logic and stochastic search*, Proc. 13th AAAI, **2**, 1194–1201, (1996).

[9] H. Kautz and B. Selman, *BlackBox: A new approach to the application of theorem proving to problem solving*, Working Notes of the AIPS98 Workshop on Planning as Combinatorial Search, Pittsburg, PA, (1998).

[10] J. Koehler, B. Nebel, J. Hoffmann and Y. Dimopoulos, *Extending planning graphs to an ADL subset*, Fourth European Conference in Planning, 273–285, (1997).

[11] A. Howe, E. Dahlman, C. Hansen, M. Scheetz and A. von Mayrhauser, *Exploiting Competitive Planner Performance*, Proc. of the European Conference in Planning, 60–72, (1999).

[12] J.S. Penberthy and D.S. Weld. *UCPOP: A Sound, Complete, Partial Order Planner for ADL*, Proc. of the 1992 International Conference on Principles of Knowledge Representation and Reasoning, 103–114. Morgan Kaufmann, Los Altos, CA, (1992).

[13] L. Sebastia, E. Onaindia, E. Marzal, *Improving expressivity and efficiency in Partial-Order Causal Link planners*, Proc. of the 18th Workshop of the UK Planning and Scheduling Special Interest Group, **1**, 124–136, (1999).

[14] D. Weld, C. Anderson and D. Smith, *Extending graphplan to handle uncertainty and sensing actions*, Proc. of the 15th National Conference on AI, 897–904, (1998).