GAMER: Fully-Observable Non-Deterministic Planning via PDDL-Translation into a Game

Peter Kissmann and Stefan Edelkamp
Faculty of Computer Science
TU Dortmund, Germany
{peter.kissmann,stefan.edelkamp}@cs.uni-dortmund.de

Abstract

This paper presents an optimal planner for the international probabilistic planning competition at ICAPS-08, IPPC-2008 for short. The planner solves non-deterministic action planning problems with binary decision diagrams.

The efficiency of the planning approach is based on a translation of the non-deterministic planning problems into a two-player turn-taking game, with a set of actions selected by the solver and a set of actions taken by the environment.

The formalism we use is a PDDL-like planning domain description language that has been derived to parse and instantiate general games. This conversion allows to derive a concise description of planning domains with a minimized state vector, thereby exploiting existing static analysis tools for deterministic planning.

Subsequently, we apply strong and strong cyclic planning algorithms as found in the literature. We additionally observe that the policy can be extracted naturally.

Introduction

Non-deterministic planning may be roughly characterized as planning in an *oily world*, where the outcome of actions is uncertain. More formally, non-deterministic planning with full observability refers to compute a conditional plan that achieves the goal for a non-deterministic shortest-path planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{I}, \mathcal{A}, \mathcal{T}, \mathcal{G})$ with a finite state space (set of states) $\mathcal{S}$, an initial state $\mathcal{I} \in \mathcal{S}$, a set $\mathcal{G} \subseteq \mathcal{S}$ of goal states, sets $\mathcal{A}(s)$ of applicable actions for each $s \in \mathcal{S}$, and a non-deterministic transition function $\mathcal{T}(s, a) \subseteq \mathcal{S}$.

Solutions are policies (partial functions) mapping states into actions. Let $\pi : \mathcal{S} \rightarrow \bigcup_{s \in \mathcal{S}} \mathcal{A}(s)$ be a policy, $\mathcal{S}_\pi$ the domain of definition of π, and $\mathcal{S}_\pi(s)$ the set of states reachable from s using π, then we say that π is closed with respect to s if and only if $\mathcal{S}_\pi(s) \subseteq \mathcal{S}_\pi$, π is proper with respect to s iff a goal state can be reached using π from all $s \in \mathcal{S}_\pi$(s), π is acyclic with respect to s iff there is no trajectory $s = (s_0, \ldots, s_n)$ with i and j such that $0 \leq i < j \leq n$, and $s_i = s_j$. We also say that π is closed (resp. proper or acyclic) with respect to $\mathcal{S}' \subseteq \mathcal{S}$ if it is closed (resp. proper or acyclic) with respect to all $s \in S'$.

A policy π is a *valid solution* for the non-deterministic model iff π is closed and proper with respect to the initial state $\mathcal{I}$. A valid policy π is assigned a (worst-case scenario) cost V_π equal to the longest trajectory starting at $\mathcal{I}$ and ending at a goal state. For acyclic policies with respect to $\mathcal{I}$ we have $V_\pi < +\infty$.

A policy π is *optimal* if it is a valid solution of minimum V_π value. The competition judges the cost of plans in non-deterministic domains that admit acyclic solutions, where optimal solutions always have finite cost. In non-deterministic domains with cyclic solutions, the solutions are judged solely by the time taken to generate a solution.

In this paper, we discuss the design of the planner GAMER that solves non-deterministic planning problems optimally, which means it computes strong and strong-cyclic plans that reflect optimal policies. The name GAMER refers to the fact that the planner was implemented as a side-product of our research in general game play, where we use binary decision diagrams (BDDs) to find and represent optimal playing strategies. Non-deterministic planning and game playing have much in common as in the former the environment plays the role of the adversary. We bridged the gap between general game play and planning by using a PDDL-like input for the former, which enables using existing static analysis tools. The distinctive advantage we aimed at is, therefore, the efficient state encoding by using multi-variate state encodings as inferred by many current deterministic planners. As a consequence, we compile each non-deterministic action into two, one representing the actor's desired move, and one for the response of the environment.

Transformation

Recall that the formal definition of the probabilistic version of PDDL has been enriched with an additional non-deterministic statement of the form:

(*oneof* e_1 e_2 $\ldots$ e_n)

where each e_i is a PDDL effect. The semantics is that, when executing such effect, one of the e_i, $i \in \{1, \ldots, n\}$, is chosen and applied to the current state.

Usually, PDDL domain input is schematic (parameterized), i. e. the finite number of domain objects for substituting parameters in predicates and actions are specified in the problem specific file. A PDDL domain in which all predicates are atoms and in which all actions have zero parameters is instantiated or grounded. There are many existing tools like ADL2STRIPS by Hoffmann, TRANSLATE by

Helmert, PDDLCAT by Haslum, STAN by Fox and Long, and GROUND by Edelkamp that infer a grounded PDDL representation given a schematic one. Some of them additionally provide a partitioning of atoms into mutually exclusive fact groups, which enable a concise state encoding to multi-variate variables (sometimes called SAS$^+$ encoding of a planning problem). The inference of a minimal state encoding (Edelkamp and Helmert 1999) is essential for the effectiveness of symbolic BDD-based search methods. Moreover, such state encoding leads to improved heuristics and is also referred to as SAS$^+$ planning (Helmert 2004).

We wrote a small compiler that parses a non-deterministic domain and returns the PDDL description of a two-person turn-taking game, where the active role is played by the solver, while the non-deterministic effects in the environment are determined by the opponent. We added an additional variable *action0-player* to determine the player's turn and further variables *action-<actionName>* to determine the action that was chosen. These variables are all mutually exclusive. This way, the player chooses an action and the corresponding action-variable is set. This gives control to the environment, which decides which of the effects will be taken. It also ensures all the effects not in $e_1, \ldots, e_n$. Afterwards, control is returned to the player.

After the translation of the non-deterministic domain into PDDL representing the two-player games of player and environment, any PDDL static analyzer can be used to instantiate the domain.

Translation Complexity

Translation is linear in the parameterized domain, which is very small, and certainly dominated by the action instantiation process in the static analysis tool, which in turn should be smaller than the planning time for generating the conditional plan.

We now closely look at the state vector. Suppose that the minimized state representation results in a state vector $(v_1, \ldots, v_l)$ with each v_i in $D(v_i)$ for $i \in \{1, \ldots, l\}$. Then the vector has $\sum_{i=1}^{l} \lceil \log(|D(v_i)|) \rceil$ bits.

As every action-variable represents the chosing of one of the actions (and thus the environment's turn) and the player's control-variable represents the player's turn, they form a mutually-exclusive group, such that this enlarges the state vector by the logarithm of the number of actions plus one: $\lceil \log(|\mathcal{A}| + 1) \rceil$, resulting in the total number of bits being

$$\lceil \log(|\mathcal{A}| + 1) \rceil + \sum_{i=1}^{l} \lceil \log(|D(v_i)|) \rceil.$$

Computing Conditional Plans

As universal plans are expected to be rather large, a compact description with Binary Decision Diagrams (BDDs) (Bryant 1985) is promising. BDDs encode state sets rather space efficiently, exploiting the sharing of state vectors in a decision diagram with respect to a fixed ordering of the state variables.

In our implementation we adapted the strong and strong-cyclic planning algorithms of Cimatti, Roveri, and Traverso (2003) to please this input. The algorithms are defined in terms of state-action tables that map states to their according actions. The ultimate output is a state-action table in form of a BDD representing $\pi(x, a)$. While strong planning grows an initially empty plan, strong cyclic planning truncates an initially universal plan.

Procedure *Strong-Plan*
$\pi' \leftarrow \top$
$\pi \leftarrow \bot$
$S \leftarrow \mathcal{G} \vee S(\pi)$
while $(\pi \neq \pi') \wedge (\mathcal{I} \not\subseteq S)$
$\pi' \leftarrow \pi$
$\Pi \leftarrow$ *StrongPreImage*(S)
$\pi \leftarrow \pi \vee (\Pi \wedge \neg S)$
$S \leftarrow \mathcal{G} \vee S(\pi)$

Figure 1: Strong Planning Algorithm.

For a planning problem $\mathcal{P}$ we call a plan π *weak*, if for $\mathcal{I}$ a terminal state in $\mathcal{G}$ is reachable; *strong*, if the induced execution structure is acyclic and all its terminal states are contained in $\mathcal{G}$; *strong cyclic*, if from every state in the plan a final state is reachable and every terminal state in the induced execution structure is contained in $\mathcal{G}$.

The intuition for weak plans is that the goal can be reached, but not necessarily so for all possible paths. For strong plans, the goal has to be satisfied despite all non-determinism, and for strong-cyclic plans all execution paths at least have the chance to reach the goal. We have

$$\textit{WeakPreImage}(S', A) = \{S \mid \mathcal{T}(S, A, S'), S \in \mathcal{S}\}$$

as the set of all states S that can reach a state in S by performing A; and

$$\textit{SrongPreImage}(S', A) = \{S \mid \emptyset \neq \textit{Exec}(S, A) \subseteq \mathcal{S}\}$$

as the set of all states S from which A is applicable and application of that reaches S'.

After compiling and instantiating the planning problem, our approach constructs a BDD representation for the initial and goal state sets as well as a BDD for a partitioned transition relation $T_a(x, x')$ with x (x') being state vectors in the lifted state space and where a is any of the player's or environment's moves. Next, we run the adapted algorithms (shown for the sake of completeness in Figures 1 and 2).

For the strong planning algorithm, the state-action table is extended by the state-action pairs calculated by the strong pre-image of all states in π in each step; the old table is denoted by π'; the new one is denoted by π; the states stored in π are returned by $S(\pi)$. The algorithm terminates if no further change from π to π' can be observed. An alternative stopping condition is that the initial state has been reached.

The strong cyclic planning algorithm starts with the universal state-action table. Iteratively, the state-action pairs whose actions lead to states outside the table are pruned, followed by those that are not reachable anymore. Once a fixpoint is reached, those pairs that do not provide any progress toward the goal are removed from the table as well.

Procedure *Strong-Cyclic-Plan*
$\pi' \leftarrow \bot$
$\pi \leftarrow \top$
while $(\pi \neq \pi')$
$\pi' \leftarrow \pi$
$\Pi \leftarrow$ *PruneOutgoing*$(\pi, \mathcal{G})$
$\pi \leftarrow$ *PruneUnconnected*$(\Pi, \mathcal{G})$
if $(\mathcal{I} \subseteq \mathcal{G} \vee S(\pi))$
$\Pi \leftarrow \bot$
repeat
$\Pi' \leftarrow \Pi$
$S \leftarrow \mathcal{G} \vee S(\Pi)$
preImage $\leftarrow \pi \wedge$ *PreImage*(S)
$\Pi \leftarrow \Pi \vee$ (*preImage* $\wedge \neg S$)
until $(\Pi' = \Pi)$
$\pi \leftarrow \Pi$

Figure 2: Strong Cyclic Planning Algorithm.

For more details, all the algorithms can be found in Cimatti, Roveri, and Traverso (2003).

As in our case the action variables are implicit (a part of the state description). Thus, the BDD for the strong/strong-cyclic plan that we obtain is $\pi(x)$ (a state table compared to the state-action table used usually).

Extraction of the Plan

Once the algorithm terminates, extracting the conditional plan is rather obvious. In case a plan is found (otherwise there is no solution), we just iterate over all elements of $\pi(x)$. Given such an element s, we quantify over the action variables $a_1, \ldots, a_m$ ($\exists a_1 \ldots a_m.\ s$) to get the state's description and over the state variables $x_1, \ldots, x_l$ ($\exists x_1 \ldots x_l.\ s$) to get the corresponding action to take.

As some state variables are detected as being constants and thus compiled away by our static analyzer, in the policy we output we have to insert these again (as being $\top$ in all states).

Initial Tests

We tested our implementation on the the tire-world example that has been published along IPPC-2008. We observed that due to the concise state encoding, we actually compile away some static state variables that were present in the original problem description. To please the validation tool, we reinsert the variables through a monitoring of the omissions that have been done wrt. the original file.

Moreover, we found out that for the tire-world domain neither a strong nor a strong-cyclic plan can be computed. We could compute a weak plan, but according to the rules of the IPPC-2008 we return *no solution*.

Conclusion

We showed how to transform a non-deterministic planning problem into a two-player turn-taking game in PDDL notation, mainly in order to apply deterministic tools to infer a minimized state encoding fully automatically. Additionally, this allows to compute disjunctive pre-images. Using a concise state encoding, we expect smaller BDDs and using partitioned transition relations, we expect faster running times. As a feature, conditional plans can be inferred very naturally, and the extraction of the ASCII representation in competition format given the BDD is straight-forward.

We used a representation of two-player games that we found attractive in solving two-player games. Note that there is also a decent link to CTL model checking (McMillan 1998). It has been observed that searching for a strong plan for goal ϕ can be casted as satisfying the CTL goal **AG** ϕ. A strong cyclic plan corresponds to a formula **AGEF** ϕ.

References

Bryant, R. E. 1985. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, 688–694.

Cimatti, A.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *AI* 147(1-2):35–84.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP*, 135–147.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

McMillan, K. L. 1998. Temporal logic and model checking. In Inan, M. K., and Kurshan, R. P., eds., *Verification of Digital and Hybrid Systems*, 36–54. Springer.