Optimal Symbolic PDDL3 Planning with MIPS-BDD

Stefan Edelkamp [*]
Computer Science Department
University of Dortmund, Dortmund, Germany

Introduction

State trajectory and plan preference constraints are the two language features introduced in PDDL3 (Gerevini & Long 2005) for describing benchmarks of the 5^{th} international planning competition. *State trajectory constraints* provide an important step of the agreed fragment of PDDL towards the description of temporal control knowledge (Bacchus & Kabanza 2000) and temporally extended goals (DeGiacomo & Vardi 1999). They assert conditions that must be met during the execution of a plan and are often expressed using using quantification over domain objects. Annotating goal conditions and state trajectory constraints with *preferences* models *soft constraints*. For planning with preferences, the objective function scales the violation of the constraints.

Symbolic exploration based on BDDs (Bryant 1985) acts on sets of states rather than on singular ones and exploit redundancies in the joint state representation. BDDs are directed acyclic automata for the bitvector representation of a state. The unique representation of a state set as a BDD is much more memory-efficient than an explicit representation for the state set. In MIPS-BDD we make optimal BDD solver technology applicable to planning with PDDL3 domains. We compile state trajectory expressions to PDDL2 (Fox & Long 2003). The grounded representation is annotated with propositions that maintain the truth of preferences and operators that model that the synchronized execution or an associated property automaton. We contribute *Cost-Optimal Breadth-First-Search* and adapt it to the search with preference constraints.

Symbolic Breadth-First Search

Symbolic search is based on satisfiability checking. The idea is to make use of Boolean functions to avoid (or at least lessen) the costs associated with the exponential memory blow-up for the state set involved as problem sizes get bigger. For propositional action planning problems we can encode the atoms that are valid in a given planning state individually by using the binary representation of their ordinal numbers, or via the bit vector of atoms being true and false.

There are many different possibilities to come up with an encoding of states for a problem. The more obvious ones seem to waste a lot of space, which often leads to bad performance of BDD algorithms. We implemented the approach of (Helmert 2004) to infer a minimized finite domain encoding of a propositional planning domain[1]

Given a fixed-length binary encoding for the state vector of a search problem, characteristic functions represent state sets. The function evaluates to true for the binary representation of a given state vector, if and only if, the state is a member of that set. As the mapping is 1-to-1, the characteristic function can be identified with the state set itself. Transitions are formalized as relations, i.e., as sets of tuples of predecessor and successor states, or, alternatively, as the characteristic function of such sets. The *transition relation* has twice as many variables as the encoding of the state. If x is the binary encoding of a state and x' is the binary encoding of a successor state, then $T(x, x')$ evaluates to true. We observe that T is the disjunct of all individual state transitions T_O, with O being an operator in $\mathcal{O}$. What we are really interested in, is to compute the (partitioned) *image* $\bigvee_{O \in \mathcal{O}} \exists x \ (T_O(x, x') \wedge \mathit{Open}(x))$ of a state set represented by *Open* wrt. a transition relation T.

For symbolic breadth-first search, let Open_i be the boolean representation of a set of states reachable from the initial state $\mathcal{I}$ in i steps, initialized with $\mathit{Open}_0 = \mathcal{I}$, and $\mathit{Open}_{i+1}(x') = \bigvee_{O \in \mathcal{O}} \exists x \ (T_O(x, x') \wedge \mathit{Open}_i(x))$. Note that S on the right hand side of the equation depends on x compared to x' on the left hand side. Thus, it is necessary to substitute x' with x in Open_i, written as $\mathit{Open}_i[x \leftrightarrow x']$. To terminate the exploration, we check, whether $\mathit{Open}_i \wedge \mathcal{G}$ is equal to the *false* function $\perp$.

In order to retrieve the solution path we assume that all sets $\mathit{Open}_0, \ldots, \mathit{Open}_i$ are available. We start with a state that is in the intersection of Open_i and the goal $\mathcal{G}$. This state is the last one on the sequential optimal solution path. We take its characteristic function S into the relational product with T to compute its potential predecessors. Next we compute the second last state on the optimal solution path in the intersection of *Pred* and Open_{i-1}, and iterate until the entire solution has been constructed.

[*]The author is supported by the German Research Foundation (DFG) project *Heuristic Search* Ed 74/3

[1]We found an application for further improvement of the encoding through a specialized BDD exploration. A set of atoms $a_1 \vee \ldots \vee a_n$ can be merged to a fact/SAS$^+$ group if the planning goal $\sum_{1 \leq i \neq j \leq n} a_i \wedge a_j$ cannot be reached from the initial state. A BDD backward search terminates usually fast.

We employ BDDs for symbolic exploration. A BDD is a data structure for a concise and unique representation of Boolean functions in form of a DAG with a single root node and two sinks, labeled "1" and "0", respectively. For evaluating the represented function for a given input, a path is traced from the root node to one of the sinks. The variable ordering has a large influence on the size of a reduced and ordered BDD. In an interleaved representation, that we employ for the transition relation, we alternate between x and x' variables. Moreover, we have experimented that preference variables are better to be queried at the top of the BDD.

BDDs for Bounded Arithmetic Constraints

The computation of a BDD $F(x)$ for a linear objective function $f(x) = \sum_{i=1}^{n} a_i x_i$, we first compute the minimal and maximal value that f can take. This defines the range that has to be encoded in binary. For the ease of presentation we assume that we consider $x_i \in \{0,1\}$.

The work of (Bartzis & Bultan 2006) shows that the BDD for representing f has at most $O(n \sum_{i=1}^{n} a_i)$ nodes and can be constructed with matching time performance. Even wile taking the most basic representation, this result improves on alternative, more expressive structures like ADDs. Moreover, the result generalizes to variables $x_i \in \{0,\ldots,2^b\}$ and the conjunction/disjunction of several linear arithmetic formulas. This implies that Metric Planning for bounded linear arithmetic expressions in the preconditions and effects is actually efficient for BDDs.

The BDD construction algorithm in MIPS-BDD for the objective function differs from the specialized construction in (Bartzis & Bultan 2006) but computes the same result.

Symbolic Cost-Optimal Breadth-First Search

We build the binary representation for the objective function as follows. For goal preferences of type ($\texttt{preference}$ p ϕ_p) we associate a Boolean variable v_p (denoting the violation of p) and construct the following indicator function: $X_p(v,x) = (v_p \wedge \phi_p(x)) \vee (\neg v_p \wedge \phi_p(x))$.

Figure 1 displays the pseudo-code for a symbolic BFS-exploration incrementally improving an upper bound U on the solution length. The state sets that are used are represented in form of BDDs. The search frontier denoting the current BFS layer is tested for an intersection with the goal, and this intersection is further reduced according to the already established bound.

Theorem The latest plan stored by the algorithm *Cost-Optimal-Symbolic-BFS* has minimal cost.

Proof The algorithm eliminates duplicates and traverses the entire planning state space. It generates each possible planning state exactly once. Only inferior states are pruned.

State Trajectory Constraints

State trajectory constraints can be interpreted Linear Temporal Logic (LTL) (Gerevini & Long 2005) and translated into automata that run concurrent to the search and accept when the constraint is satisfied (Gastin & Oddoux 2001). LTL includes temporal modalities like **A** for *always*, **F** for

Procedure Cost-Optimal-Symbolic-BFS
Input: State space problem with transition relation T
Goal BDD $\mathcal{G}$, and initial BDD $\mathcal{I}$
Output: Optimal solution path is stored

```
$U \leftarrow \infty$
loop
    $Reach(x') \leftarrow \mathcal{I}(x')$; $Open(x') \leftarrow \mathcal{I}(x)$
    $Intersection(x) \leftarrow \mathcal{I}(x) \wedge \mathcal{G}(x)$
    $Bound(v) \leftarrow F(v) \wedge \bigvee_{i=0}^{U} [v = i]$
    $Eval(v,x) \leftarrow Intersection(x) \wedge \bigwedge_p X_p(v,x)$
    $Metric(x) \leftarrow \exists v : Eval(v,x) \wedge Bound(v)$
    while ($Metric(x) \neq \bot$)
        if ($Open = \bot$) return "Exploration completed"
        $Succ(x') = \bigvee_{O \in \mathcal{O}} \exists x\, T_O(x,x') \wedge Open(x)$
        $Open(x) \leftarrow (Succ(x') \wedge \neg Reach(x'))[x' \leftrightarrow x]$
        $Reach(x') \leftarrow Reach(x') \vee Succ(x')$
        $Intersection(x) \leftarrow Open(x) \wedge \mathcal{G}(x)$
        $Eval(v,x) \leftarrow Intersection(x) \wedge \bigwedge_p X_p(v,x)$
        $Metric(x) \leftarrow \exists v : Eval(v,x) \wedge Bound(v)$
    $U \leftarrow ConstructAndStoreSolution(Metric(x)) - 1$
```

Figure 1: Cost-Optimal BFS Planning Algorithm.

eventually, and **U** for *until*. We propose to compile the automata back to PDDL with each transition introducing a new operator (Edelkamp 2006). Each automaton state for each automaton results in an atom. For detecting accepting states we additionally include *accepting* propositions. The initial state of the planning problem includes the start state of the automaton and an additional proposition if it is accepting. For all automata, the goal includes their acceptance.

Including state trajectory constraints in the Cost-Optimal Breadth-First Search algorithm is achieved as follows.

For ($\texttt{hold-after}$ t ϕ) we impose that ϕ is satisfied for the search frontier in all steps $i > t$. For ($\texttt{hold-during}$ t_1 t_2 ϕ) as similar reasoning applies.

For ($\texttt{sometimes}$ ϕ) we apply automata-based model checking to build a (Büchi) automata for the LTL formula $\mathbf{F}\phi$. Let $\mathcal{S}$ be the original planning space and $A_{\mathbf{F}\phi}$ be the constructed (Büchi) automaton for formula $A_{\mathbf{F}\phi}$ and $\otimes$ the cross product between two automata, then $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{F}\phi}$ and $\mathcal{G} \leftarrow \mathcal{G} \cup \{\texttt{accepting}(A_\phi)\}$. The initial state is extended by the initial state of the automaton, which in this case is not accepting.

For ($\texttt{sometimes-before}$ ϕ ψ) the temporal formula is more complicated, but the reasoning remains the same. We compile $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{(\neg\phi\wedge\neg\psi)\mathbf{U}((\neg\phi\wedge\psi)\vee(\mathbf{A}(\neg\phi\wedge\neg\psi)))}$ and adapt the planning goal and the initial state accordingly.

For ($\texttt{always}$ ϕ) we apply automata theory to construct $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{G}\phi}$. Alternatively, for all i we could impose $Open_i \leftarrow Open_i \wedge \phi$ in analogy to $\texttt{hold-during}$ and $\texttt{hold-after}$. For ($\texttt{at-most-once}$ ϕ) we assign the planning problem $\mathcal{P}$ to $\mathcal{P} \otimes A_{\mathbf{A}\phi\rightarrow(\phi\mathbf{U}(\mathbf{G}\neg\phi)))}$. For ($\texttt{within}$ t ϕ) we build the cross product $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{F}\phi}$. Moreover, we set $Open_t \leftarrow Open_t \wedge \{\texttt{accepting}(A_{\mathbf{F}\phi})\}$.

Preferences for State Trajectory Constraints

For state trajectory constraints that are constructed via automata theory, we apply the following construction. Instead of adding the automaton acceptance to the goal state we combine the acceptance with the violation predicate. If the automaton accepts then the preference is not violated; if it is located in a non-accepting state, then it is violated. For example, given ($\texttt{preference}\ p$ ($\texttt{at-most-once}\ \phi$)) we explore the cross product $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{A}\phi \rightarrow (\phi \mathbf{U}(\mathbf{G}\neg\phi))}$. Let $a = \{\texttt{accepting}(A_{\mathbf{A}\phi \rightarrow (\phi \mathbf{U}(\mathbf{G}\neg\phi)))})\}$. If $a \in \mathit{add}(O)$ then $\mathit{del}(O) \leftarrow \mathit{del}(O) \cup \{v_p\}, \mathit{add}(O) \leftarrow \mathit{add}(O) \setminus \{v_p\}$. If $a \in \mathit{del}(O)$ then $\mathit{add}(O) \leftarrow \mathit{add}(O) \cup \{v_p\}, \mathit{del}(O) \leftarrow \mathit{del}(O) \setminus \{v_p\}$. An specialized operator *skip* allows to fail the automata completely. If automaton is ignored once, it remains invalid for the rest of the computation.

Memory Limitation

BDDs already save space for large state sets. For purely propositional domains we additionally apply bidirectional symbolic BFS, which is often much faster as unidirectional search. Symbolic BFS is supposed to have small search frontiers (Jensen *et al.* 2006).

One implemented idea is an extension to *Frontier-Search* (Korf *et al.* 2005), which has been proposed for undirected or directed acyclic graph structures. In more general planning problems we have established that a duplicate detection scope (a.k.a. *locality*) of 4 is sufficient to guarantee termination for *Cost-Optimal-Symbolic-BFS* in the competition domains. Moreover, we do not store any intermediate BDD layer that corresponds to state trajectory automata transitions. Only the layers that correspond to the original unconstrained state space are stored.

Our competition results are either *step-optimal* (*Propositional* domains) or *cost-optimal* (*Simple Preferences* / *Qualitative Preferences* domains). We have not yet implemented support for metric and temporal planning operators. There is 3 restrictions to the optimality in state-trajectory domains.

1. We do not support *preference preconditions*. Actually, we can parse and process the conditions, but as the domain of the `is-violated` variables is in fact unbounded this affects a possible encoding as a BDD. Nonetheless, as these variables are monotone increasing, it is not difficult to design a specialized solution for them.
2. We assume that the automaton that is built does not affect the optimality. An automaton that constructed via the LTL translation in LTL2BA is in fact optimized in the number of states and not for the preserving path lengths. On the other hand, there some LTL converters that preserve optimal paths (Schuppan & Biere 2005).
3. The exploration is terminated by limited time or space resources. In this case the reported plans for preference domains are optimal only wrt. the search depth reached.

For larger problems, we looked at suboptimal solutions. We have tested an in-built support for canceling the exploration if the BDD node count for optimal search exceeds a threshold on BDD nodes that corresponds to the limitations of main memory. Subsequently, the entire memory for all BDD nodes is released. We successfully tested two strategies, *heuristic symbolic search* based on pattern databases and *symbolic beam-search* removing unpromising states. For the competition, we switched this feature off.

Conclusion

We have devised an optimal propositional PDDL3 planning algorithm based on BDDs. Besides using the same LTL2BA converter, the algorithm shares no code with our explicit-state planner MIPS-XXL. As the approach for state trajectory constraints relies on a translation to LTL, it has the potential to deal with much larger temporal constraint language expressiveness than currently under consideration.

After the competition, we will likely extend the above planning approach to general domains with linear expressions in the actions. As a prerequisite to apply (Bartzis & Bultan 2006) numerical state variables have to fit into some finite domains. Most of the metric planning domains around belong to this group. Moreover, we encountered that model checkers like nuSMV and CadenceSMV can already deal with LTL formula. For this cases, the LTL formula is directly encoded into a transition relation without using an intermediate explicit automaton (Schuppan & Biere 2005).

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Bartzis, C., and Bultan, T. 2006. Efficient BDDs for bounded arithmetic constraints. *STTT* 8(1):26–36.

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

DeGiacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 226–238.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *ICAPS*, To Appear.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gastin, P., and Oddoux, D. 2001. Fast LTL to Büchi automata translation. In *CAV*, 53–65.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

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

Jensen, R.; Hansen, E.; Richards, S.; and Zhou, R. 2006. Memory-efficient symbolic heuristic search. In *ICAPS*, To Appear.

Korf, R. E.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.

Schuppan, V., and Biere, A. 2005. Shortest counterexamples for symbolic model checking of LTL with past. In *TACAS*, 493–509.