ComPlan: A Conformant Probabilistic Planner[*]

Jinbo Huang
Logic and Computation Program
National ICT Australia
Canberra, ACT 0200 Australia
jinbo.huang@nicta.com.au

Abstract

ComPlan is a conformant probabilistic planner that finds a plan with maximum probability of success for a given horizon. The core of the planner is a a depth-first branch-and-bound search in the plan space. For each potential search node, an upper bound is computed on the success probability of the best plans under the node, and the node is pruned if this upper bound is not greater than the success probability of the best plan already found. A major source of efficiency for this algorithm is the efficient computation of these upper bounds, which is possible by encoding the original planning problem as a propositional formula and compiling the formula into deterministic decomposable negation normal form.

Conformant Probabilistic Planning

Consider the Slippery-Gripper domain (Kushmerick, Hanks, & Weld 1995), where a robot needs to have a block painted and held in his gripper, while keeping his gripper clean. The gripper starts out clean, but may be wet, which may prevent him from holding the block; painting the block, while certain to succeed, may make his gripper dirty; a dryer is available to dry the gripper. Given probability distributions quantifying these uncertainties and a planning horizon, the robot requires a plan that achieves the goal with maximum probability.

A probabilistic planning problem can be characterized by a tuple $\langle \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{G}, n \rangle$ where $\mathbf{S}$ is the set of possible world states, $\mathbf{I}$ is a probability distribution over $\mathbf{S}$ quantifying the uncertainty about the initial state, $\mathbf{A}$ is the set of actions, all of which are assumed to be applicable in every state, $\mathbf{G}$ is the set of goal states, and n is the planning horizon.

To quantify the uncertainty in action effects, each action $a \in \mathbf{A}$ is a function that maps each state $s \in \mathbf{S}$ to a probability distribution Pr_s^a over all states $\mathbf{S}$. Starting from our initial *belief state* B_0, which is equal to $\mathbf{I}$, each action $a \in \mathbf{A}$ taken will bring us to a new belief state with updated probabilities for the world states $\mathbf{S}$:

$$B_n(s') = \sum_{s \in \mathbf{S}} B_{n-1}(s) \cdot Pr_s^a(s').$$

A solution to the *conformant* probabilistic planning problem is then a sequence of n actions, or an n-step plan, leading to belief state B_n, such that the sum of the probabilities assigned by B_n to the goal states is maximized.

[*]This document is based on (Huang 2006).

Propositional Encoding

A probabilistic planning problem can be encoded by a propositional formula, where a subset of the propositional variables are labeled with probabilities (Littman 1997). As an example we consider the encoding of Slippery-Gripper. The state space $\mathbf{S}$ of Slippery-Gripper can be encoded by four propositional variables: BP (block-painted), BH (block-held), GC (gripper-clean), GD (gripper-dry). Suppose initially the block is not painted and not held, and the gripper is clean but dry only with probability 0.7.

To encode this initial belief state, we introduce a *chance* variable p, and label it with the number 0.7. We then write:

$$\neg\mathsf{BP}, \neg\mathsf{BH}, \mathsf{GC}, \neg p \vee \mathsf{GD}, p \vee \neg\mathsf{GD}.$$

This five-clause formula has the property that each setting of variable p will simplify the formula, resulting in a single world state, whose probability is given by the label of p (if p is set to $true$), or 1 minus that (if p is set to $false$).

We now consider the encoding of uncertain action effects. First we introduce a new set of state variables—$\mathsf{BP}', \mathsf{BH}', \mathsf{GD}', \mathsf{GC}'$—to encode the states reached after executing an action. There are three actions available: $\mathbf{A} = \{\mathsf{dry}, \mathsf{paint}, \mathsf{pickup}\}$. Suppose action dry dries a wet gripper with probability 0.8, and does not affect a dry gripper. We introduce a chance variable q, label it with 0.8, and write:

$$\neg\mathsf{dry} \vee \mathsf{GD} \vee \neg q \vee \mathsf{GD}', \neg\mathsf{dry} \vee \mathsf{GD} \vee q \vee \neg\mathsf{GD}', \neg\mathsf{dry} \vee \neg\mathsf{GD} \vee \mathsf{GD}'.$$

We also need a set of clauses, known as a *frame axiom*, saying that the other variables are not affected by the action:

$$\neg\mathsf{dry} \vee (\mathsf{BP} \Leftrightarrow \mathsf{BP}'), \neg\mathsf{dry} \vee (\mathsf{BH} \Leftrightarrow \mathsf{BH}'), \neg\mathsf{dry} \vee (\mathsf{GC} \Leftrightarrow \mathsf{GC}').$$

After all actions are encoded, we write the following saying that exactly one of the three actions will be taken:

$$\mathsf{dry} \vee \mathsf{paint} \vee \mathsf{pickup},$$

$$\neg\mathsf{dry} \vee \neg\mathsf{paint}, \neg\mathsf{dry} \vee \neg\mathsf{pickup}, \neg\mathsf{paint} \vee \neg\mathsf{pickup}.$$

Finally, our goal that the block be painted and held and the gripper be clean translates into three unit clauses:

$$\mathsf{BP}', \mathsf{BH}', \mathsf{GC}'.$$

This completes our propositional encoding of the planning problem, for horizon 1. The resulting set of clauses Δ_1 can be characterized as the conjunction of three components:

$$\Delta_1 \equiv \mathcal{I}(P_{-1}, S_0) \wedge \mathcal{A}(S_0, A_0, P_0, S_1) \wedge \mathcal{G}(S_1),$$

where $\mathcal{I}(P_{-1}, S_0)$ is a set of clauses over the initial chance variables P_{-1}, and the state variables S_0 at time 0, encoding the initial belief state; $\mathcal{A}(S_0, A_0, P_0, S_1)$ is a set of clauses over the state variables S_0, action variables A_0, and chance variables P_0 at time 0, and state variables S_1 at time 1, encoding the action effects; and $\mathcal{G}(S_1)$ is a set of clauses over the state variables S_1 at time 1, encoding the goal condition.

From this characterization, an encoding Δ_n for n-step planning can be produced in a mechanical way by repeating the middle component with a new set of variables for each additional time step, and updating the goal condition:

$$\Delta_n \equiv \mathcal{I}(P_{-1}, S_0) \wedge \mathcal{A}(S_0, A_0, P_0, S_1) \wedge \mathcal{A}(S_1, A_1, P_1, S_2) \wedge \ldots \wedge \mathcal{A}(S_{n-1}, A_{n-1}, P_{n-1}, S_n) \wedge \mathcal{G}(S_n). \quad (1)$$

In this encoding, an n-step plan is an instantiation π of the action variables $A = A_0 \cup A_1 \cup \ldots \cup A_{n-1}$, an *eventuality* is an instantiation of the chance variables $P = P_{-1} \cup P_0 \cup \ldots \cup P_{n-1}$, and the probability of an eventuality is given by multiplying together the label of each chance variable, or 1 minus that, depending on its sign in the instantiation. A solution to the conformant probabilistic planning problem is then a plan π^* such that the sum of the probabilities $Pr(\epsilon)$ of all eventualities ϵ consistent with π is maximized:

$$\pi^* = \arg\max_{\pi} \sum_{\pi \wedge \epsilon \wedge \Delta_n \text{is consistent}} Pr(\epsilon). \quad (2)$$

Compilation to Deterministic DNNF

COMPLAN exploits the particular structure of probabilistic planning problems, as characterized by Equation 1, by compiling Δ_n into deterministic decomposable negation normal form (deterministic DNNF, or d-DNNF) (Darwiche & Marquis 2002) using the publicly available C2D compiler (Darwiche 2004; 2005), before the search starts.

Deterministic DNNF A propositional formula is in d-DNNF if it (i) only uses conjunction, disjunction, and negation, and negation only appears next to variables; and (ii) satisfies *decomposability* and *determinism*. Decomposability requires that conjuncts of any conjunction share no variables; determinism requires that disjuncts of any disjunction be pairwise inconsistent. The formula shown in Figure 1, for example, is in d-DNNF, and is equivalent to the 2-step encoding Δ_2 of SLIPPERY-GRIPPER, after the state variables $S = S_0 \cup S_1 \cup S_2$ have been existentially quantified.

Recall that existential quantification of a variable is defined as: $\exists x.\Delta \equiv \Delta|_x \vee \Delta|_{\overline{x}}$, where $\Delta|_x$ ($\Delta|_{\overline{x}}$) denotes setting variable x to $true$ ($false$) in Δ. In these planning problems, existential quantification of the state variables is useful because these variables do not appear in Equation 2 and their absence can reduce the size of the problem.

Efficient Plan Assessment Compilation of the planning problem into d-DNNF provides an efficient method for plan assessment: The computation of the success probability of any complete plan π for the n-step planning problem Δ_n,

$$Pr(\pi) = \sum_{\pi \wedge \epsilon \wedge \Delta_n \text{is consistent}} Pr(\epsilon), \quad (3)$$

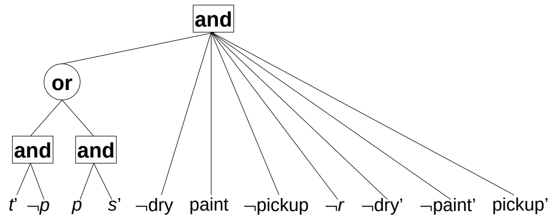

Figure 1: The 2-step SLIPPERY-GRIPPER in d-DNNF.

which is necessary for computing Equation 2, can be done in time linear in the size of a d-DNNF compilation of $\exists S.\Delta_n$, where S is the set of the state variables.

The source of this efficiency lies in that the d-DNNF graph can be viewed as a *factorization* of Equation 3, by regarding the conjunction nodes as multiplications and disjunction nodes as summations. Specifically, given the label $Pr(p)$ of each chance variable p, and a plan π, $Pr(\pi)$ can be obtained by a single bottom-up traversal of the d-DNNF graph, where a value is assigned to each node N of the graph as described in Algorithm 1.

Algorithm 1 Plan Assessment

$$val(N, \pi) = \begin{cases} 1, & \text{if } N \text{ is a leaf node that mentions an action variable and is consistent with } \pi; \\ 0, & \text{if } N \text{ is a leaf node that mentions an action variable and is inconsistent with } \pi; \\ Pr(p), & \text{if } N \text{ is a leaf node } p \text{ where } p \text{ is a chance variable;} \\ 1 - Pr(p), & \text{if } N \text{ is a leaf node } \neg p \text{ where } p \text{ is a chance variable;} \\ \prod_i val(N_i, \pi), & \text{if } N = \bigwedge_i N_i; \\ \sum_i val(N_i, \pi), & \text{if } N = \bigvee_i N_i. \end{cases}$$

The value assigned to the root is then $Pr(\pi)$. For example, given the 2-step plan $\mathsf{paint\text{-}pickup'}$ for SLIPPERY-GRIPPER, which is actually a 6-variable instantiation $(\neg\mathsf{dry}, \mathsf{paint}, \neg\mathsf{pickup}, \neg\mathsf{dry'}, \neg\mathsf{paint'}, \mathsf{pickup'})$, Algorithm 1 run on Figure 1 computes its success probability as 0.7335.

Depth-First Branch-and-Bound

Given a method for plan assessment (Algorithm 1), an optimal conformant plan can be found by a systematic search in the space of all possible plans. COMPLAN uses a depth-first branch-and-bound for this purpose, where upper bounds on values of partial plans are efficiently computed by traversing the d-DNNF compilation of the planning problem.

Computing Upper Bounds Recall that our goal is to compute Equation 2, which is a sequence of maximizations over the action variables followed by a sequence of summations over the chance variables. Note that maximizations commute, and therefore the maximizations over action variables can be performed in any order. Similarly, the summations over chance variables can be performed in any order. These two sequences cannot be swapped or mixed, though, as maximization does not commute with summation.

However, if we disregard this constraint and allow the maximizations and summations to be mixed in any order, it is not difficult to see that we will get a result that *cannot be lower* than the correct value. The motivation for lifting the variable ordering constraint is that we can now take advantage of the bounded treewidth of these planning problems by performing these maximizations and summations on the compact d-DNNF compilation, and using the results as upper bounds to prune a search.

Specifically, Algorithm 2 computes an upper bound on the success probability of the best completions of a partial plan π, by a single bottom-up traversal of the d-DNNF graph for $\exists S.\Delta$ (note that we are using the same name for the value function as in Algorithm 1, which can now be regarded as a special case of Algorithm 2 where π is a complete plan).

Algorithm 2 Upper Bound

$$val(N,\pi) = \begin{cases} 1, & \text{if } N \text{ is a leaf node that mentions an action variable and is consistent with } \pi; \\ 0, & \text{if } N \text{ is a leaf node that mentions an action variable and is inconsistent with } \pi; \\ Pr(p), & \text{if } N \text{ is a leaf node } p \text{ where } p \text{ is a chance variable;} \\ 1 - Pr(p), & \text{if } N \text{ is a leaf node } \neg p \text{ where } p \text{ is a chance variable;} \\ \prod_i val(N_i,\pi), & \text{if } N = \bigwedge_i N_i; \\ \max_i val(N_i,\pi), & \text{if } N = \bigvee_i N_i \text{ and } N \text{ is a decision node over an action variable not set by } \pi; \\ \sum_i val(N_i,\pi), & \text{if } N = \bigvee_i N_i \text{ and } N \text{ is not of the above type.} \end{cases}$$

COMPLAN keeps the d-DNNF graph G on the side during search. For an n-step planning problem, the maximum depth of the search will be n. At each node of the search tree, an unused time step k, $0 \leq k < n$, is chosen (this need not be in chronological order), and branches are created corresponding to the different actions that can be taken at step k. The path from the root to the current node is hence a partial plan π, and can be pruned if $val(G,\pi)$ computed by Algorithm 2 is less than or equal to the *lower bound* on the value of an optimal plan. This lower bound is initialized to 0 and updated whenever a leaf is reached and the corresponding complete plan has a greater value. When search terminates, the best plan found together with its value is returned.

Variable Ordering Let $a_k^1, a_k^2, \ldots, a_k^{|\mathbf{A}|}$ be the propositional action variables for step k, where $\mathbf{A}$ is the set of actions. At each node of the search tree, let lb be the current lower bound on the success probability of an optimal plan, let π be the partial plan committed to so far, and let k be some time step that has not been used in π. We are to select a k and branch on the possible actions to be taken at step k. We consider the following:

$$hb_k = \max\{b_i : b_i = val(G, \langle \pi, a_k^i \rangle), b_i > lb\}, \quad (4)$$

where $\langle \pi, a_k^i \rangle$ denotes the partial plan π extended with one more action a_k^i (and $\overline{a_k^j}$ for all $j \neq i$, implicitly). This quantity hb_k gives the highest value among the upper bounds for the prospective branches that will not be pruned, and we propose to select a k such that hb_k is *minimized*.

The intuition is that the minimization of hb_k gives the *tightest* upper bound on the value of the partial plan π, and by selecting step k as the next branching point, upper bounds subsequently computed are likely to improve as well.

Value Ordering After a time step k is selected for branching, we will explore the unpruned branches a_k^i in *decreasing* order of their upper bounds. The intuition here is that a branch with a higher upper bound is more likely to contain an optimal solution. Discovery of an optimal solution immediately gives the tightest lower bound possible for subsequent pruning, and hence its early occurrence is desirable.

Value Elimination In the process of computing Equation 4 to select k, some branches a_k^i may have been found to be prunable. Although only one k is ultimately selected, all such branches can be pruned as they are discovered. This can be done by asserting $\overline{a_k^i}$ (and adding it to π implicitly) in the d-DNNF graph G for all k and i such that $val(G, \langle \pi, a_k^i \rangle) \leq lb$. Upper bounds computed after these assertions will generally improve, because there is now a smaller chance for the first case of Algorithm 2 to execute.

Acknowledgments

National ICT Australia is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Darwiche, A. 2004. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 328–332.

Darwiche, A. 2005. The C2D compiler user manual. Technical Report D-147, Computer Science Department, UCLA. http://reasoning.cs.ucla.edu/c2d/.

Huang, J. 2006. Combining knowledge compilation and search for conformant probabilistic planning. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.

Littman, M. L. 1997. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, 748–754.