Utilizing Structured Representations and CSPs in Conformant Probabilistic Planning

Nathanael Hyafil and **Fahiem Bacchus**[1]

1 Introduction[2]

Conformant planning problems are planning problems under incomplete knowledge, non-deterministic actions, and no sensing. Since the plan is oblivious to the system state, it cannot be conditional. Instead conformant plans are linear sequences of actions that must achieve the goal no matter what the under-determined initial state actually is, and no matter what happens during its non-determinstic execution. Conformant plans are useful in situations where sensing is too expensive or when a failure has occurred and sensing is no longer available. Probabilistic conformant plans are a useful generalization in the case when we can quantify the different possible initial states, and the different possible execution paths with probabilities. A probabilistic conformant plan is one that has a probability of success, and the conformant probabilistic planning problem, CPP, is the problem of finding a plan with maximum probability of success.

In previous work we presented a planning algorithm for solving CPPs [1]. This algorithm employed a CSP encoding of the problem, and a backtracking tree search similar in spirit to the SAT based encoding used in the MAXPLAN planner [3] (which also solved CPPs). Although the CSP approach yielded consistently superior performance to the SAT based MAXPLAN, its performance was generally inferior to state-of-the-art algorithms for Partially Observable Markov Decision Processes (POMDPs). Such POMDP algorithms can solve CPP as a special case. One of the main problems with our previous approach was that the algorithm's worst case exponential space complexity was often realized in practice. Nevertheless, the core ideas embodied in the algorithm captured a number of useful intuitions for solving CPPs more effectively [1].

In this paper we recast our algorithm so that it can utilize decision diagrams; specifically algebraic decision diagrams (ADDs) [4]. Decision diagrams have been used before for various types of planning, but not previously for CPP and not previously in the manner we utilize them here. Decision diagrams still have worst case exponential space complexity, but in practice can often be much more compact. Our empirical results demonstrate that this is generally the case in our application as well.

2 Background

States in a CPP problem are represented by a set of state variables each of which can take on a finite number of different values. A complete set of assignments to these variables specifies a particular state; $\mathcal{S}$ contains all possible complete sets of assignments. A CPP includes a specifed initial belief state $\mathcal{B}$, which is a probability distribution over $\mathcal{S}$: for any $s \in \mathcal{S}$, $\mathcal{B}[s]$ is the probability of s. The problem is to achieve a goal $G \subseteq \mathcal{S}$, using *probabilistic* actions from the set A. For each pair of states s, s' and action a, $Pr(s, a, s')$ denotes the probability that a yields s' when executed in s. A plan $\pi = \langle a_1, \ldots, a_k \rangle$ generates a set of *execution paths*. Each execution path is a length $k+1$ sequence of states $s_0, s_1, \ldots, s_k$ where $\mathcal{B}[s_0] > 0$, and $Pr(s_{i-1}, a_i, s_i) > 0$ for all $0 < i \leq k$. The first state s_0 is a possible initial state, and each subsequent state is a possible successor of the previous state under the action specified in π. The probability of an execution path is $\mathcal{B}[s_0] \prod_{i=1}^{k} Pr(s_{i-1}, a_i, s_i)$. This is the probability that executing π will yield this particular sequence of states. The *value* of π is equal to the sum of the probabilities of all of its execution paths whose final state is in G, i.e., the probability that π achieves the goal.

To encode a length n CPP as a CSP, $n+1$ copies of the state variables are used, indexed by the plan step; each copy is used to represent one of the states in a possible execution path of a n-length plan. There are also n action variables, each one with domain equal to the set of actions. n additional sets of "random" variables are used to capture the randomness of the action effects: the legal settings of the i-th random variable correspond to the possible probabilistic effects of the i-th action on the i-th state. Thus each setting of these variables has a particular probability—the probability the i-th action will have this particular effect–and determines the i-th action's effects. There are three types of constraints: initial and goal state constraints (which force the initial state to be one where $\mathcal{B}[s] > 0$ and the goal state to be in G), and action constraints. These constraints force the transitions between s_i and s_{i+1} to be compatible with a_i given the random effects determined by the i-th random variables. Each solution to the CSP is a particular execution path of a particular length n plan that reaches the goal.

Conceptually CPP can be solved by finding all solutions to its CSP encoding and for each plan summing the probability of the solutions corresponding to its execution paths. In this context, the CSP constraints serve to eliminate zero probability execution paths, and constraint propagation serves to prune some of the path prefixes that have probability zero of reaching the goal. Goal reachability analysis through constraint propagation is one of the main features of the CSP approach. Summing all solutions is not practical, but dynamic programming techniques can be employed to cache and reuse intermediate results as follows. During the search we will visit nodes ν corresponding to a particular state s_i at the i-th step of an execution path. The path to ν corresponds to some execution path prefix $s_0, s_1, \ldots, s_i$ for some plan prefix $a_1, \ldots, a_i$, and the subtree under ν will contain all successful execution path suffixes $s_{i+1}, \ldots, s_n$ ($s_n \in G$) for all possible plan suffixes $\tau = a_{i+1}, \ldots, a_n$. If for all

[1] Department of Computer Science, University of Toronto, Toronto, Canada. email: nhyafil@cs.utoronto.ca, fbacchus@cs.utoronto.ca.

[2] Due to space considerations a number of important details and citations of related work have been elided. See [2] for a more detailed presentation including citations of related work.

Name	p2-2-2		p3-2-2		p4-2-2		p2-2-4	
l-t-p-a	4-2-2-1		6-2-2-2		8-2-2-1		3-2-4-1	
$\|S\|$	392		1458		3872		19208	
$\|A\|$	30		46		66		54	
n	CPplan	Pomdp	CPplan	Pomdp	CPplan	Pomdp	CPplan	Pomdp
1	0	0.2	0	5.8	0	73.7	0	648
2	0	1.8	0	48.8	0	1123	0	13974
3	0	6.5	0	116	0	3184	0	–
4	0	21.5	0	230	0.1	9119	0	–
5	0.1	66.6	0	451	0.5	–	0.3	–
6	0.6	212	0.3	877	2.4	–	2.1	–
7	1.7	737	1b.9	1796	10.9	–	11.4	–
8	3.7	3495	5.7	3871	34.9	–	39.9	–
9	7.6	–	15.6	9378	83.0	–	185	–
10	14.8	–	28	–	208	–	391	–
11	42.6	–	45.8	–	304	–	722	–
12	281	–	68.8	–	648	–	1221	–
13	oom	–	122	–	6398	–	1445	–
14		–	414	–	oom	–	3725	–
15		–	oom	–	–	–	oom	–

Table 1. Solution Time on 4 Logistics problems in CPU sec. using Caching with ADDs. l-t-p-a: number of locations, trucks, packages and airplanes. $|S|$: number of states, $|A|$: number of actions. n (plan length). "–": $>$ 15000 seconds.

	Deterministic		Full Uncertainty	
n	CPplan	Pomdp	CPplan	Pomdp
1	0	5.0	0	6.4
2	0	49.7	0	53.6
3	0	104.0	0	127.7
4	0	170.4	0	254.1
5	0.1	279.7	0	498.3
6	0.5	427.7	0.3	967.8
7	2.5	674.8	1.5	1934
8	8.9	1054	5.3	4031
9	23.6	1614	13.6	9163
10	31.4	2686	26.0	–
11	40.7	3907	32.7	
12	35.0	5776	40.7	
13	41.9	8536	51.7	
14	69.9	11777	128.8	
15	68.2	–	1124	
16	91.7		oom	
17	99.0			
18	103.6			
19	91.6			
20	126.6			
21	137.1			
22	oom			

Table 2. Solution Time on P-3-2-2 with "extreme" probability settings (CPU sec.) using Caching with ADDs. n: plan length. "–": $>$ 15000 seconds.

such plan suffixes we store the probability of their success from s_i, we can reuse this information whenever another execution path prefix for any other plan prefix reaches the same state s_i. That is, we never need search the subtree below the same state at the same step of the plan twice.

Two pieces of information are stored at every step i of the plan: a list of visited states at step i, and, for each one, the value of every length $n - i$ plan suffix in that state. Our previous algorithm used a tabular representation for this, requiring space exponential in the plan length. In our new implementation we store this information in an ADD which builds up a much more compact representation of the information. This allows us to empirically validate the potential of the CSP approach by showing that it can in fact be much more effective than state of the art POMDP algorithms on certain problems.

3 Empirical Results

In order to evaluate our modified CSP algorithm we utilized two test domains, GRID-10X10 and a probabilistic version of the standard AI planning benchmark LOGISTICS. All experiments were performed on a 2.4 GHz Xeon machine with 3GB of RAM.

GRID-10X10: GRID-10X10 has a state space size of 100, but each of its 4 actions can only reach 3 states. Hence, GRID-10X10 has significant deterministic structure in its transition probabilities. The CSP approach, with its constraint propagation, is able to take advantage of this deterministic structure. The results for GRID-10X10 are shown in Table 3. The results compare the previous tabular representation, the ADD representation, and the witness algorithm for solving POMDPs. We see that ADDs offer a significant improvement over tables, and that the CSP approach is significantly better than the POMDP algorithm.

Logistics: The results on four different problems from the Logistics domain are shown in Table 1. In these problems *load* and *unload* are probabilistic. *load* succeeds with probability 0.875 for trucks and 0.9 for airplanes; succeeds with probability *unload* 0.75 and 0.8 respectively. *CPplan* in its tabular representation could not solve problems with so many actions and states due to the memory required. As the data shows *CPplan* with ADDs can generate at least 12 step plans in all of these domains whereas POMDP algorithms can solve at best 9-step problems and no more than 2 steps for the biggest instance within the 15,000 seconds allowed (raising this limit to 100,000 seconds only allows for one additional step to be solved). Also on problems that both techniques can tackle, *CPplan* can be up to 3 orders of magnitude faster.

Init.	n	C+Table	C+ADD	POMDP
(7,6)	5	0.01	0.06	1.21
(6,6)	6	0.01	0.10	1.44
(6,5)	7	0.01	0.16	2.15
(3,3)	12	2.97	1.12	237.0
(3,2)	13	11.72	1.72	508.8
(2,2)	14	oom	2.31	949.5
(1,1)	16	-	4.79	2282.3
(0,0)	18	-	11.44	3098.6
(0,0)	19	-	129.12	-

Table 3. Time on GRID-10X10 in CPU sec. Init. (initial state), n (plan length), C+Table (caching with tables), C+ADD (caching with ADDs), POMDP (witness algorithm), oom (out of memory), - ($>$ 3,600 sec).

Finally in order to evaluate how the amount of uncertainty affects the relative performance of *CPplan* and POMDP we ran the problem suites using entirely 0/1 probabilities (so that the problem becomes non-deterministic rather than probabilistic) and probabilities of 0.5 (maximal uncertainty). We report on problem p-3-2-2.

The results show that the relative performance does not change much: CPplan remains orders of magnitude faster than POMDPs. However, both algorithms find the deterministic problem significantly easier. (This also helps illustrate how much harder CPP is than non-probabilistic conformant planning.) Interestingly, in the full uncertainty case CPplan is a bit faster than the results given in Table 1—with equal probablity values the ADDs tend to be more compact as the functions tend to take on fewer distinct values.

REFERENCES

[1] Nathanael Hyafil and Fahiem Bacchus, 'Conformant probabilistic planning via csps', in *International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pp. 205–214, (2003).

[2] Nathanael Hyafil and Fahiem Bacchus, 'Utilizing Structured Representations and CSPs in Conformant Probabilistic Planning', in *http://www.cs.toronto.edu/∼fbacchus/Papers/HBECAI2004full.pdf*, (2004).

[3] Stephen M. Majercik and Michael L. Littman, 'MAXPLAN: A New Approach to Probabilistic Planning', in *The Fourth International Conference on Artificial Intelligence Planning Systems*, pp. 86–93, (1998).

[4] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, 'Algebraic Decision Diagrams and Their Applications', in *IEEE /ACM International Conference on CAD*, pp. 188–191, Santa Clara, California, (1993). IEEE Computer Society Press.