

# Combining Knowledge Compilation and Search for Conformant Probabilistic Planning

Jinbo Huang

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

## Abstract

We present a new algorithm for conformant probabilistic planning, which for a given horizon produces a plan that maximizes the probability of success under quantified uncertainty about the initial state and action effects, and absence of sensory information. Recent work has studied systematic search in the space of all candidate plans as a feasible approach to conformant probabilistic planning, but the algorithms proposed require caching of intermediate computations in such a way that memory is often exhausted quickly except for small planning horizons. On the other hand, planning problems in typical formulations generally have treewidths that do not grow with the horizon, as connections between variables are local to the neighborhood of each time step. These existing planners, however, are unable to directly benefit from the bounded treewidth owing to a constraint on the variable ordering which is necessary for correct computation of the optimal plan. We show that lifting such constraint allows one to obtain a compact compilation of the planning problem, from which an upper bound can be efficiently computed on the value of any partial plan generated during search. Coupled with several optimizations, this results in a depth-first branch-and-bound algorithm which on the tested domains runs an order of magnitude faster than its predecessors, and at the same time is able to solve problems for significantly larger horizons thanks to its minimal memory requirements.

## Introduction

Probabilistic planning is concerned with domains where an agent seeks to achieve a goal given uncertain knowledge, in the form of probability distributions, about the initial world state and the effects of its actions. In this work we focus on *conformant* probabilistic planning, where the agent has no sensing capabilities and therefore must compute a straight-line plan which can be executed without the need to observe the actual effects of its actions. In particular, we are interested in finding a plan that will reach a goal state in a given number of steps with maximum probability. Existing algorithms for this task (and its variant where one is interested in finding a plan whose success probability exceeds a given threshold) include BURIDAN (Kushmerick, Hanks, & Weld 1995), MAXPLAN (Majercik & Littman 1998a;

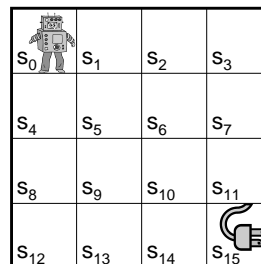


Figure 1: A blind robot with faulty legs on low battery seeks a charge station.

1998b), and *CPplan* (Hyafil & Bacchus 2003; 2004). Algorithms have also been proposed for a special case of the problem, often referred to simply as *conformant planning*, where the uncertainty is not quantified and the agent requires a plan that succeeds with probability 1 (Smith & Weld 1998; Cimatti & Roveri 2000; Bonet & Geffner 2000; 2001; Brafman & Hoffmann 2004; Palacios *et al.* 2005).

Consider the scenario in Figure 1, where a blind robot needs to walk to the charge station in the southeast corner of the room to recharge his battery. For this example we assume that the robot has been told its initial position, which is square  $s_0$ , but will not know where he is once he starts moving. At each step, the robot can choose to move to the adjacent square to his North, South, West, or East, but any move will only proceed in the intended direction with probability 0.8, and with probability 0.2 will proceed to his right (assuming he is facing the intended direction), the circuitry controlling his legs being slightly faulty; in either case he stays put if hitting the wall. For instance, on attempting to move East from  $s_0$ , the robot could end up moving South, to the square  $s_4$  below him, with probability 0.2. Now, to make matters worse, the robot's battery only has enough power to sustain 8 moves, and the question is, what sequence of moves will maximize his probability of arriving at the charge station before power runs out?

This example serves to illustrate the importance, as well as nontriviality, of careful planning in the face of uncertainty: An optimal 8-step plan, such as S-S-S-E-E-E-E-E, will succeed with probability 0.738, while a less carefully chosen plan, such as E-E-E-S-S-S-S-S, only has a success

probability of 0.168, even less than that of an optimal 6-step plan (such as E-E-E-S-S-S), which is 0.262.

In principle, conformant probabilistic planning can be solved by searching in the space of all possible sequences of actions of a given length: At each leaf of the search tree a value is computed equal to the success probability of the corresponding action sequence, and any of the leaves with the highest value is returned as representing an optimal plan for the given horizon.

For such a search to be practical, one needs (i) an efficient method for *plan assessment* (Kushmerick, Hanks, & Weld 1995), which computes the success probability of a given plan, and (ii) some provision for pruning, given the potentially exponential time complexity of the search.

For dealing with the first of these two tasks, a formalism known as *sequential effects tree* (ST) (Littman 1997) has proved useful. In the ST formulation, probabilities associated with action effects are represented by a special set of propositional variables, known as *chance* variables, which can then be conveniently incorporated into a satisfiability-based encoding (Kautz & Selman 1992) of the planning problem. The resulting encoding, in the form of a propositional formula, has the property that any complete satisfying assignment encodes exactly one possible outcome of executing a particular plan which satisfies the goal, and the product of the probabilities represented by literals in the assignment equals the probability of this particular outcome occurring. Plan assessment then reduces to a sum-of-products operation, explicit or in a factored form, over all satisfying assignments consistent with a given plan.

Both MAXPLAN (Majercik & Littman 1998a; 1998b) and CPplan (Hyafil & Bacchus 2003; 2004) have adopted the ST formulation as the basis for plan assessment, and run different versions of depth-first search in the space of variable assignments until an optimal plan is identified. To deal with the second task listed above, these planners also employ caching mechanisms, allowing a branch of the search to be pruned if the partial variable assignment leads to a subproblem that has been previously solved.

We have now come to the major issue we wish to address in this work: While effective in pruning some branches of the search, caching methods employed by these existing planners lead to fast exhaustion of memory except for small planning horizons, and at the same time do not appear to have realized the fullest power of pruning possible. On the SAND-CASTLE-67 domain (Majercik & Littman 1998a), for example, MAXPLAN and CPplan with full caching ran out of memory (3 GB) for horizons 20 and 28, respectively, and required approximately 2000 and 300 seconds (on a 2.4 GHz processor), respectively, to solve the problem for horizons 19 and 27 (Hyafil & Bacchus 2003). As we later show, the new algorithm we propose solved horizon 28 in only 10 seconds on a comparable machine (2.4 GHz with 4 GB of RAM), using less than 0.1% of the available memory. In fact, it never used more than that much memory when we gradually increased the horizon all the way to 40, which it solved in 2306 seconds, finding an optimal plan with a 0.999999 probability of success.

## Overview of Paper

The main idea of this paper is to improve *both* the time and the space efficiency of the depth-first search used for conformant probabilistic planning. Adopting the formulation used by previous planners, we start by encoding probabilistic planning problems as propositional formulas, a subset of whose variables are associated with probabilities. We then discuss the distinctive structure of the class of formulas thus generated—that their degree of connectivity, or *treewidth* in particular, does not grow with the planning horizon.

The bounded treewidth<sup>1</sup> suggests that these problems could be tractable for standard inference methods, such as variable elimination (Dechter 1996) and recursive conditioning (Darwiche 2001b), which have a complexity that is polynomial in the size of the problem and exponential only in its treewidth. Previous algorithms for conformant probabilistic planning were unable to directly benefit from this tractability, because they were obliged to (explicitly or implicitly) respect a constraint on the variable ordering in computing the success probability of an optimal plan, or an optimal completion of a partial plan. Specifically, this computation, in principle, requires a sequence of maximizations over the (uninstantiated) action variables, followed by a sequence of summations over the chance variables. When such constraint is imposed on the variable ordering, efficient computation is no longer guaranteed, because variable orders leading to low treewidth may have been ruled out.

This variable ordering constraint thus stands in our way to tractability, and borrowing a great idea from (Park & Darwiche 2003), what we propose is, simply, ignore it! Doing so we can now perform the (mixed and unsound) sequence of maximizations and summations in polynomial time by choosing a suitable variable ordering. Although the result will generally not be the exact success probability of an optimal solution, it is easy to show that it will be an *upper bound* thereon, and hence an *admissible* heuristic for pruning.

This gives rise naturally to a depth-first branch-and-bound algorithm, which is known to require only linear space, eliminating the memory limitations of previous algorithms. Here it is also worth mentioning that at the leaves of the search tree the computed upper bound will indeed equal the exact value because all action variables have been instantiated and no maximizations are required anymore. This guarantees correct plan assessment and ultimately a correct success probability for the optimal plan found.

We address next the time efficiency of the algorithm. First, to achieve the greatest amount of pruning, an upper bound must be computed at each node of the search tree, or for each partial plan generated during search. It is therefore important that each computation be performed as efficiently as possible—a naive implementation of this computation will likely overwhelm the search with much redundant work. Second, care should be taken to secure tighter bounds whenever possible, again for pruning to wield its full power.

<sup>1</sup>Throughout this paper we use the notion of “bounded treewidth” in the sense that the treewidth does not grow with the planning horizon. It can, of course, grow with the size and connectivity of the planning domain.

We tackle the first issue by compiling the propositional encoding of the planning problem into a logical form known as deterministic DNNF, or d-DNNF) (Darwiche 2001a; Darwiche & Marquis 2002). The compiler we use (Darwiche 2004; 2005) is based on recursive conditioning and since no variable ordering constraints need be imposed, the size of the compilation will only grow at most linearly with the planning horizon thanks to the bounded treewidth. A critical advantage of obtaining a d-DNNF compilation of the problem beforehand is that it represents an explicit, compact factorization of the maximization-summation sequence, which is *provably small* and which can be evaluated in *linear time* to compute the upper bound we need for each partial plan generated during search. This eliminates the need and inefficiency of repeating the factorization process (which is the bulk of the work), or worse, the flat computation based on enumerating models of the original propositional formula, each and every time an upper bound is called for.

We tackle the second issue, regarding further enhancement of the algorithm’s pruning power, by means of intelligent variable ordering, value ordering, and value elimination, all of which are oriented toward obtaining tighter upper bounds during the depth-first search. In particular, we propose and justify a new variable ordering heuristic, which calls for expanding a node the highest upper bound for whose branches is *lowest*.

The resulting algorithm, which we call COMPLAN, is then empirically evaluated on several domains which have been previously used to evaluate MAXPLAN and CPplan. The results indicate a consistent, significant improvement in *both* the time and the space efficiency of solving the problems. In particular, COMPLAN uses practically no memory and is hence able to solve problems for substantially larger horizons than the previous planners. For horizons previous planners are able to handle, it runs an order of magnitude faster.

The rest of the paper is organized as follows: We review conformant probabilistic planning and its propositional encoding; describe the compilation of these encodings into d-DNNF; present the depth-first branch-and-bound algorithm where upper bounds are computed by traversing the d-DNNF compilation in linear time, together with the various techniques for obtaining tighter bounds; demonstrate the empirical time and space efficiency of the new algorithm; and conclude with a few words on related work.

## Conformant Probabilistic Planning

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. For the problem in Figure 1, for example,  $\mathbf{S}$  is the set of the 16 squares,  $\mathbf{I}$  assigns probability 1 to square  $s_0$  and probability 0 to all others,  $\mathbf{A} = \{N, S, W, E\}$ ,  $\mathbf{G} = \{s_{15}\}$ , and  $n = 8$ .

To quantify the uncertainty of 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  $\mathbf{S}$ . For example, action E

above maps state  $s_0$  to a probability distribution  $Pr_{s_0}^E$  such that  $Pr_{s_0}^E(s_1) = 0.8$ ,  $Pr_{s_0}^E(s_4) = 0.2$ , and  $Pr_{s_0}^E(s') = 0$  for all other states  $s'$ .

Starting from our initial *belief state* (Bonet & Geffner 2000)  $B_0$ , which is equal to  $\mathbf{I}$ , each action taken will bring us to a new belief state with updated probabilities for the world states  $\mathbf{S}$ . Let us continue to use the example in Figure 1, and suppose action E is taken at step 0. Our new belief state  $B_1$  will then be:

$$B_1(s') = \begin{cases} 0.8, & \text{if } s' = s_1; \\ 0.2, & \text{if } s' = s_4; \\ 0, & \text{otherwise.} \end{cases}$$

Now let action E be taken again at step 1. Our new belief state  $B_2$  will be:

$$B_2(s') = \begin{cases} 0.64, & \text{if } s' = s_2; \\ 0.32, & \text{if } s' = s_5; \\ 0.04, & \text{if } s' = s_8; \\ 0, & \text{otherwise.} \end{cases}$$

In general, after the execution of action  $a \in \mathbf{A}$ , we have:

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

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. In our example, the 8-step plan S-S-S-E-E-E-E-E is one solution, leading to  $B_8(s_{15}) = 0.738$  (in this case there is only one goal state).

## Propositional Encoding

A probabilistic planning problem can be encoded by a propositional formula, where a subset of the propositional variables are labeled with probabilities. A formal procedure for doing so can be found in (Kautz & Selman 1992; Littman 1997). Here we explain it using an example.

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.

The state space  $\mathbf{S}$  of this domain 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:

$$\neg BP, \neg BH, GC, \neg p \vee GD, p \vee \neg GD.$$

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*). In

general, when there are multiple chance variables, the probability of the resulting world state is the product of these numbers, one from each chance variable.

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

$$\begin{aligned} &\neg\text{dry} \vee \text{GD} \vee \neg q \vee \text{GD}', \\ &\neg\text{dry} \vee \text{GD} \vee q \vee \neg\text{GD}', \\ &\neg\text{dry} \vee \neg\text{GD} \vee \text{GD}'. \end{aligned}$$

These clauses have the property that each setting of variable  $q$  will make *dry* a deterministic action with respect to variable  $\text{GD}$ . For example, under  $q = \text{true}$ , the clauses simplify to  $\neg\text{dry} \vee \text{GD} \vee \text{GD}'$ ,  $\neg\text{dry} \vee \neg\text{GD} \vee \text{GD}'$ , which says that after *dry* is executed ( $\text{dry} = 1$ ), the gripper will be dry regardless of its condition before the action ( $\text{GD} \vee \text{GD}'$ ,  $\neg\text{GD} \vee \text{GD}'$ ). The probability of this effect occurring is then given by the the label of  $q$  (0.8), and the probability of the alternative effect occurring (that the action will not make a wet gripper dry) is given by 1 minus that number.

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

$$\begin{aligned} &\neg\text{dry} \vee (\text{BP} \Leftrightarrow \text{BP}'), \\ &\neg\text{dry} \vee (\text{BH} \Leftrightarrow \text{BH}'), \\ &\neg\text{dry} \vee (\text{GC} \Leftrightarrow \text{GC}').^2 \end{aligned}$$

In general, these two sets of clauses will encode the uncertain effects of an action so that each setting of the chance variables forces exactly one deterministic effect, whose probability of occurring is given by the product of numbers contributed by the chance variables, each number being the label of the variable if that variable is set to *true*, and 1 minus that if it's set to *false*.

The remaining two actions can be encoded in a similar way. Suppose action *paint* paints the block with probability 1, but makes the gripper dirty with probability 1 if it holds the block and probability 0.1 if it doesn't; and action *pickup* picks up the block with probability 0.95 if the gripper is dry and probability 0.5 if it is wet. Introducing chance variables  $r, s, t$  to represent the probabilities 0.1, 0.95, 0.5, and omitting the frame axioms for brevity, we have:

$$\begin{aligned} &\neg\text{paint} \vee \text{BP}', \\ &\neg\text{paint} \vee \neg\text{BH} \vee \neg\text{GC}', \\ &\neg\text{paint} \vee \text{BH} \vee (r \Leftrightarrow \neg\text{GC}'), \\ &\neg\text{pickup} \vee \neg\text{GD} \vee (s \Leftrightarrow \text{BH}'), \\ &\neg\text{pickup} \vee \text{GD} \vee (t \Leftrightarrow \text{BH}'). \end{aligned}$$

<sup>2</sup>We are using the equivalence operator  $\Leftrightarrow$  for brevity. Recall that an equivalence  $a \Leftrightarrow b$  translates into two clauses  $\neg a \vee b, a \vee \neg b$ .

We now write the following clauses saying that exactly one of the three actions will be taken:

$$\begin{aligned} &\text{dry} \vee \text{paint} \vee \text{pickup}, \\ &\neg\text{dry} \vee \neg\text{paint}, \\ &\neg\text{dry} \vee \neg\text{pickup}, \\ &\neg\text{paint} \vee \neg\text{pickup}. \end{aligned}$$

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

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

This completes our propositional encoding of the planning problem, for horizon 1. The resulting set of clauses  $\Delta_1$  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  $\Delta_n$  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:

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

## Semantics of the Encoding

The semantics of this propositional encoding becomes clear when we look at the models (satisfying assignments) of the propositional formula. The formula  $\Delta_1$  above, for example, has no models. This means that no 1-step plan satisfies the goal with positive probability (indeed, the block cannot become painted and held after any single action). If we consider the 2-step version of the problem, the following is one of the models of the formula  $\Delta_2$  that encodes it (the overbar denotes an assignment of *false* and its absence *true*):

$$\begin{aligned} &p, \overline{\text{BP}}, \overline{\text{BH}}, \text{GC}, \overline{\text{GD}}, \overline{\text{dry}}, \overline{\text{paint}}, \overline{\text{pickup}}, \overline{q}, \overline{r}, s, t, \\ &\text{BP}', \overline{\text{BH}'}, \text{GC}', \overline{\text{GD}'}, \overline{\text{dry}'}, \overline{\text{paint}'}, \overline{\text{pickup}'}, \overline{q'}, \overline{r'}, s', t', \\ &\text{BP}'', \text{BH}'', \text{GC}'', \text{GD}''. \end{aligned}$$

This model encodes the fact that if the action sequence *paint-pickup'* is executed, then under a particular setting ( $p, \overline{q}, \overline{r}, s, t, \overline{q'}, \overline{r'}, s', t'$ ) of all the chance variables, it will lead to the goal. All models of  $\Delta_2$  consistent with the partial assignment (*paint, pickup'*) then encode all the *eventualities* where this particular plan *paint-pickup'* will succeed.

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 corresponding model. For example, the above model gives a probability of  $0.7 \cdot (1 - 0.8) \cdot (1 - 0.1) \cdot 0.95 \cdot 0.5 \cdot (1 - 0.8) \cdot (1 - 0.1) \cdot 0.95 \cdot 0.5 = 0.005117175$ .

The probability of success for the 2-step plan paint-pickup' is then the sum of the probabilities given by all models of  $\Delta_2$  consistent with (paint, pickup').

In general, an  $n$ -step plan is an instantiation  $\pi$  of the action variables  $A = A_0 \cup A_1 \cup \dots \cup A_{n-1}$  in  $\Delta_n$ , an eventuality is an instantiation  $\epsilon$  of the chance variables  $P = P_{-1} \cup P_0 \cup \dots \cup P_{n-1}$  in  $\Delta_n$ , and a solution to the conformant probabilistic planning problem is a plan  $\pi^*$  such that the sum of the probabilities  $Pr(\epsilon)$  of all eventualities  $\epsilon$  consistent with  $\pi$  is maximized:

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

## Compilation to Deterministic DNNF

In this section we show how the particular structure of probabilistic planning problems, as characterized by Equation 1, can be exploited by compiling the propositional encoding  $\Delta_n$  into deterministic decomposable negation normal form (deterministic DNNF, or d-DNNF) (Darwiche 2001a; Darwiche & Marquis 2002). Based on the compilation, we then show in the next section how an upper bound can be efficiently computed on the value of any partial plan, leading to a time- and space-efficient depth-first branch-and-bound algorithm for computing Equation 2.

### Deterministic DNNF

A propositional formula is in d-DNNF if it (i) only uses conjunction (and,  $\wedge$ ), disjunction (or,  $\vee$ ), and negation (not,  $\neg$ ), and negation only appears next to variables; and (ii) satisfies *decomposability* and *determinism*. Decomposability requires that conjuncts of any conjunction share no variables, and determinism requires that disjuncts of any disjunction be pairwise logically inconsistent.

The formula shown as a graph in Figure 2, for example, is in d-DNNF, and is equivalent to the 2-step encoding  $\Delta_2$  of SLIPPERY-GRIPPER described earlier, after all the state variables  $S = S_0 \cup S_1 \cup S_2$  have been existentially quantified.

Recall that existential quantification of a variable is the removal of references to that variable from all models of the propositional theory. Formally,  $\exists x. \Delta \equiv \Delta|_x \vee \Delta|_{\bar{x}}$ , where  $\Delta|_x$  ( $\Delta|_{\bar{x}}$ ) denotes setting variable  $x$  to *true* (*false*) in the propositional formula  $\Delta$  (multiple variables can be existentially quantified in any order using this definition). 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. Indeed, any plan  $\pi$  (instantiation of action variables) and eventuality  $\epsilon$  (instantiation of chance variables) uniquely determine the values of all the state variables, and hence one may just as well ignore their presence. The d-DNNF compiler we use, described later in the section, allows these variables to be existentially quantified during the compilation process with no extra cost.

We note that although the structure shown in Figure 2 is a tree, d-DNNF representations are graphs in general, allowing the sharing of subformulas for compactness. Also, there is only one candidate plan in this example (that is the only way to possibly have the block painted and held in two steps

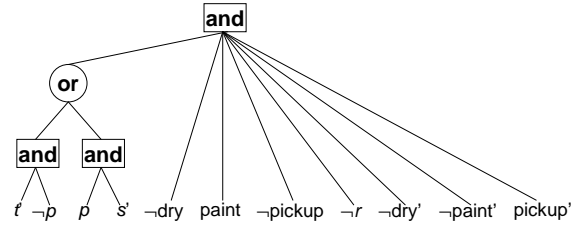


Figure 2: The 2-step SLIPPERY-GRIPPER in d-DNNF.

while keeping the gripper clean), but in general the d-DNNF graph will encode multiple candidate plans, among which we later show how to efficiently search for an optimal one.

### Efficient Plan Assessment

The first motivation for compiling the planning problem into d-DNNF is an efficient plan assessment algorithm it provides: The computation of the success probability of any complete plan  $\pi$  for the  $n$ -step planning problem  $\Delta_n$ ,

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

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 (intuitively, decomposability ensures that the multiplications are well defined and determinism ensures that no eventuality is doubly counted). Specifically, given the label  $Pr(p)$  of each chance variable  $p$ , and a plan  $\pi$ ,  $Pr(\pi)$  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

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

The value assigned to the root of the d-DNNF graph is then the success probability  $Pr(\pi)$  of the complete plan  $\pi$ , as stated formally in the following theorem:

**Theorem 1 (Plan Assessment)** Let  $G$  be the root node of a d-DNNF graph equivalent to  $\exists S.\Delta_n$ , where  $S = S_0 \cup S_1 \cup \dots \cup S_n$  are the state variables of the  $n$ -step planning problem  $\Delta_n$ . We have  $val(G, \pi) = Pr(\pi)$  for any complete instantiation  $\pi$  of the action variables  $A_0 \cup A_1 \cup \dots \cup A_{n-1}$ .

For example, given the 2-step plan paint-pickup' for SLIPPERY-GRIPPER, which is actually a 6-variable instantiation ( $\neg dry, paint, \neg pickup, \neg dry', \neg paint', pickup'$ ), a single traversal of the graph of Figure 2 computes its probability of success as 0.7335.

### Exploiting Bounded Treewidth Using C2D

Compilation of propositional formulas into d-DNNF can be done by invoking the publicly available C2D compiler (Darwiche 2004; 2005). We will not delve into the algorithm used by C2D, but one of the features of this compiler is that it can base the compilation process on a given variable order, and produce a d-DNNF graph whose size is guaranteed, *in the worst case*, to be linear in the number of variables, and exponential only in the *width* of the variable order.

The width of a variable order is defined with respect to the *connectivity graph* of the propositional formula, where each variable becomes a node and there is an edge between each pair of variables that appear together in some clause. To compute the width, one *eliminates* the variables one at a time in the given order, by deleting the corresponding node with all its incident edges from the graph. All the neighbors of a node are pairwise connected (where they are not connected already) before the node is deleted, and the maximum number of neighbors any node has right before its deletion is then the width of the variable order. The *treewidth* (Robertson & Seymour 1986) of the graph is the minimum width among all possible variable orders.

The abovementioned complexity guarantee offered by the C2D compiler brings us to the second motivation for compiling probabilistic planning problems into d-DNNF: These problems have treewidths that do not grow with the planning horizon, which implies that there exist variable orders of bounded width. In fact, any variable order consistent with the chronological order of the time steps, or the reverse thereof, has a bounded width, because variables in any time step are connected only to those in the immediately preceding and succeeding time steps (see Equation 1), and therefore the neighbors of an eliminated variable can only be in its own time step and one of the two neighboring time steps depending on the direction of the elimination.

Having variable orders of bounded width implies that the size of the d-DNNF compilation will grow at most linearly with the horizon, which will allow us to ultimately scale the proposed approach to large horizons. To give a concrete idea of this linear growth and the scalability of this part of the computation, we report in Table 1 the results of compiling the SLIPPERY-GRIPPER domain (an extra action clean has been added as in (Hyafil & Bacchus 2003) ) using C2D for horizons 1 to 20 and some selected large horizons. The state variables have been existentially quantified from all results using the “-exist” option for C2D. The second column gives the number of variables and clauses in the original encoding.

Table 1: Linear growth of d-DNNF compilations for planning problems with bounded treewidth.

Horizon	Vars	Clauses	Width	d-DNNF		
				Nodes	Edges	Time
1	18	33	6	1	0	0.02
2	31	58	9	17	16	0.02
3	44	83	9	123	264	0.02
4	57	108	9	276	590	0.03
5	70	133	9	435	901	0.05
6	83	158	9	598	1220	0.03
7	96	183	9	761	1539	0.04
8	109	208	9	924	1858	0.07
9	122	233	9	1087	2177	0.05
10	135	258	9	1250	2496	0.05
11	148	283	9	1413	2815	0.05
12	161	308	9	1576	3134	0.09
13	174	333	9	1739	3453	0.10
14	187	358	9	1902	3772	0.06
15	200	383	9	2065	4091	0.06
16	213	408	9	2228	4410	0.07
17	226	433	9	2391	4729	0.12
18	239	458	9	2554	5048	0.13
19	252	483	9	2717	5367	0.08
20	265	508	9	2880	5686	0.09
100	1305	2508	9	15920	31206	0.82
200	2605	5008	9	32220	65908	5.35
500	6505	12508	9	81120	158806	42.43
1000	13005	25008	9	162620	318306	174.00
2000	26005	50008	9	325620	637306	698.18

The third column gives the width of the variable order used, which is the reverse natural order.<sup>3</sup> The last column gives the number of nodes and edges of the d-DNNF compilation, and the time taken, in seconds based on a 2.4 GHz processor. (Note that for horizon 1, the single-node d-DNNF graph is an empty disjunction, equivalent to *false*, signifying that no 1-step plan with positive success probability exists.)

We now proceed to describe the main algorithm of this paper, where these d-DNNF compilations of planning problems will be used to efficiently compute upper bounds to prune a depth-first search for optimal plans.

### Depth-First Branch-and-Bound

Given a method for plan assessment, which we have provided in Theorem 1, an optimal conformant plan can be found by any systematic search in the space of all possible sequences of actions. Previous implementations of this approach include both MAXPLAN (Majercik & Littman 1998a; 1998b) and CPplan (Hyafil & Bacchus 2003; 2004), which, however, suffer from major memory limitations owing to the caching methods used, as we discussed earlier.

In this section we present a depth-first branch-and-bound algorithm for optimal conformant probabilistic planning, where upper bounds on values of partial plans are efficiently computed by traversing the d-DNNF compilation of the

<sup>3</sup>This causes C2D to instantiate variables in roughly increasing order of the time steps, which is helpful as the values of earlier action and chance variables determine the subsequent state.

planning problem. A depth-first search requires only linear space, and the d-DNNF graph, as discussed in the previous section, also has a linear size. Therefore this new algorithm eliminates the memory limitations of the previous planners. As we shall later see, it also exhibits a significantly better time efficiency on the tested domains, thanks to the effectiveness of the upper bounds and several optimizations we employ to further improve their tightness.

## Computing Upper Bounds

Recall that our goal is to compute Equation 2, which can be decomposed into a sequence of maximizations over the action variables followed by a sequence of summations over the chance variables. Specifically, the definition of  $val(\Delta)$ , which denotes the success probability of an optimal conformant plan given the propositional encoding  $\Delta$  of the probabilistic planning problem, can be written as follows:

$$val(\Delta) = \begin{cases} \max\{val(\Delta|_a), val(\Delta|_{\bar{a}})\}, & \text{if } \Delta \text{ mentions some action variable } a; \\ val(\Delta|_p) \cdot Pr(p) + val(\Delta|_{\bar{p}}) \cdot (1 - Pr(p)), & \text{if } \Delta \text{ mentions no action variables, but} \\ & \text{some chance variable } p; \\ 1, & \text{if } \Delta \text{ mentions no action or chance} \\ & \text{variables, and is consistent;} \\ 0, & \text{if } \Delta \text{ mentions no action or chance} \\ & \text{variables, and is inconsistent.} \end{cases}$$

Intuitively, the first case of the definition says that the actions should be chosen to give the maximum probability of success; the second case says that for a complete sequence of actions chosen, the success probability is the weighted average between the success probabilities under the two complementary scenarios  $p$  and  $\bar{p}$ , for each chance variable  $p$ ; the other two cases ensure that all and only the scenarios satisfying the goal are counted.

Note that this definition applies as well to the value  $Pr(\pi)$  of a partial plan  $\pi$ , which is defined as the success probability of its optimal completion. Specifically, we have  $Pr(\pi) = val(\Delta|_\pi)$ , where  $\Delta|_\pi$  denotes setting the relevant action variables to constants in  $\Delta$  as they are set in  $\pi$ .

Since maximization commutes with maximization, in the above definition 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*  $val(\Delta)$  (Park & Darwiche 2003). From here on we will denote this result by  $val'(\Delta)$ . Accordingly,  $val'(\Delta|_\pi)$  cannot be lower than  $val(\Delta|_\pi)$  for partial plan  $\pi$ .

The motivation for lifting the variable ordering constraint is that we can now take advantage of the bounded treewidth of these planning problems by computing  $val'(\Delta|_\pi)$  on the compact d-DNNF compilation of  $\Delta$  produced by the C2D compiler, and using the results as upper bounds to prune a

search. As discussed earlier, for these compilations to have the guaranteed linear size, it is critical for the compiler to freely choose suitable variable orderings. Indeed, our experience indicates that imposing the ordering constraint on these problems, in order to directly compute  $val(\Delta)$ , generally makes the compilation infeasible except for small horizons, as would be expected given the inherent complexity of the planning problem.

Before describing the detailed procedure for computing  $val'(\Delta|_\pi)$  on the d-DNNF compilation of  $\Delta$ , we need to first point out that results produced by the C2D compiler have a special structure. Specifically, when no existential quantification has been requested, disjunction nodes in the d-DNNF graph produced by C2D are *decision nodes*, which have the

form of  $\beta \text{ or } \neg x \text{ and } x \text{ and } \gamma$ , where  $x$  is a variable and  $\beta$  and  $\gamma$  are two subgraphs. Such a structure facilitates the computation of both maximization and summation: When  $x$  is an action (chance) variable one takes the max (sum) at the or-node.

When existential quantification of the state variables  $S$  has been requested, which we would like to do as discussed earlier, what happens is that any would-be decision node

$\beta \text{ or } \neg x \text{ and } x \text{ and } \gamma$  collapses into a simpler disjunction  $\beta \text{ or } \gamma$  wherever  $x$  is a state variable. On these or-nodes we will perform a summation as well since the two disjuncts  $\beta$  and  $\gamma$  represent two disjoint sets of scenarios in which the partial plan can be successfully completed.

We now present Algorithm 2, which computes the upper bound  $val'(\Delta|_\pi)$  by a single bottom-up traversal of the d-DNNF graph for  $\exists S.\Delta$ .

---

### Algorithm 2 Upper Bound

---

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


---

Note that we are using the same name for the value function as in Algorithm 1. The reason is that Algorithm 1 can now be regarded as a special case of Algorithm 2 where  $\pi$  is

a complete plan. It is precisely in this case that the computed value is guaranteed to be exact; in other cases they are upper bounds as formally stated in the following theorem:

**Theorem 2 (Upper Bound)** *Let  $G$  be the root node of a d-DNNF graph equivalent to  $\exists S.\Delta$  produced by C2D, where  $S$  are the state variables of the planning problem  $\Delta$ . We have  $val(G, \pi) \geq val(\Delta|_{\pi}) = Pr(\pi)$  for any partial instantiation  $\pi$  of the action variables.*

At this point our depth-first branch-and-bound algorithm is ready to run, keeping the d-DNNF graph  $G$  on the side. 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  $\pi$ , 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

Good variable orderings are a source of efficiency for almost any search. In this algorithm we have to implicitly examine all candidate plans, and our goal is to use orderings that will lead to greater amounts of pruning.

Let  $a_k^1, a_k^2, \dots, a_k^{|\mathbf{A}|}$  be the propositional action variables for step  $k$ , where  $\mathbf{A}$  is the set of actions. (Our “variable ordering” does not refer to these variables, but to the implicit variables representing the action to be taken at each step  $k$ .) At each node of the search tree, let  $lb$  be the current lower bound on the success probability of an optimal plan, let  $\pi$  be the partial plan committed to so far, and let  $k$  be some time step that has not been used in  $\pi$ . We are to select a  $k$  and branch on the possible actions to be taken at step  $k$ . To facilitate our selection, we consider the following quantity:

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

where  $\langle \pi, a_k^i \rangle$  denotes the partial plan  $\pi$  extended with one more action  $a_k^i$  (and  $a_k^j$  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  $\pi$ , and by selecting step  $k$  as the next branching point, upper bounds subsequently computed are likely to improve as well. Our experiments confirm this intuition, although we will not include those results here. Interestingly, this variable ordering heuristic is in contrast to that proposed in (Park & Darwiche 2003)—where a similar search is performed on Bayesian networks—which calls for the maximization of the ratio between  $hb_k$  and the sum of the upper bounds for the unpruned branches. Our next two techniques, however, are common in this class of algorithms, and are in line with the corresponding proposals in (Park & Darwiche 2003).

### 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. (This corresponds to exploring the values of the implicit multi-valued variable representing the action at step  $k$ ; hence the term *value ordering*.) 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 (i.e., as Equation 4 is being computed). This can be done by asserting  $\overline{a_k^i}$  (and adding it to  $\pi$  implicitly) in the d-DNNF graph  $G$  for all  $k$  and  $i$  such that  $val(G, \langle \pi, a_k^i \rangle) \leq lb$ , which is a constant-time operation. 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.

### Variable Sharing

Our final optimization is not specific to the search algorithm, but has to do with reducing the number of propositional variables required for the encoding of the planning problem. Specifically, probabilities that have equal values can be represented by the same chance variable, as long as they apply to different actions in the same time step. This is sound as these actions are mutually exclusive: For any action taken, the chance variables for other actions will only appear in clauses that have been subsumed.

## Experimental Results

We now report the empirical performance of this algorithm, which we call COMPLAN, on three domains that have been used to evaluate previous planners. The domains are SAND-CASTLE-67 (Majercik & Littman 1998a), SLIPPERY-GRIPPER (Kushnerick, Hanks, & Weld 1995) as extended in (Hyafil & Bacchus 2003), and GRID-10X10 (where the robot starts at the origin) (Hyafil & Bacchus 2003). The planners for comparison are MAXPLAN (Majercik & Littman 1998a; 1998b) and *CPplan* (Hyafil & Bacchus 2003; 2004). The machine we use has a 2.4GHz processor with 4GB of RAM. In all the experiments, however, COMPLAN used less than 0.1% of the memory.

Results of running COMPLAN on these domains are given in Tables 2, 3, and 4. For each domain and horizon used, we report the success probability of the optimal plan found, the number of nodes (in the search tree) visited during search, and the time taken (in seconds). The time to compile problems into d-DNNF is relatively insignificant and has not been included here (for the largest horizon the compilation time was 0.05, 0.09, and 14.01 seconds, respectively, for the three domains). Note that the horizons start at 2 in Table 3 and 18 in Table 4 because the smaller horizons admit no valid plans, which was discovered by the compilation alone.

It is also worth noting that these three domains have 2, 4, and 4 actions, respectively, and hence a brute-force search would have to visit up to  $2^{40}$ ,  $4^{20}$ , and  $4^{32}$  nodes, respectively, if it were to solve the largest horizon for each domain.

We observe that these results go far beyond the scale of problems that have been handled by the two previous planners. Specifically, (Hyafil & Bacchus 2003; 2004) reported the following based on a 2.4 GHz processor with 3 GB of RAM: On SAND-CASTLE-67, MAXPLAN ran out of memory for horizon 20, and took about 2000 seconds to solve horizon 19; CPplan ran out of memory for horizon 28, and took about 300 seconds to solve horizon 27. On SLIPPERY-GRIPPER, MAXPLAN ran out of memory for horizon 12, and took about 2000 seconds to solve horizon 11; CPplan ran out of memory for horizon 14, and took about 100 seconds to solve horizon 13. On GRID-10X10, no results were reported for MAXPLAN, and the largest horizon reported for CPplan is 19, which it took 129.12 seconds to solve.

### Conclusions and Related Work

We have presented COMPLAN, a new algorithm for conformant probabilistic planning, which significantly improves on both the time and the space efficiency of previous search algorithms for the same task. The source of this improved efficiency lies in the exploitation, via knowledge compilation, of the bounded treewidth of the planning problems, combined with linear-time methods for plan assessment and the computation of upper bounds, in the framework of a depth-first branch-and-bound search.

The work most closely related to ours is (Palacios *et al.* 2005), which uses d-DNNF compilation and search for conformant planning given deterministic actions and *unquantified* uncertainty about the initial state. The goal of this task is to find a plan that succeeds for every possible initial state the agent may have started in. Since no probabilities are involved, the d-DNNF compilation of the planning problem is used to count the number of initial states that are consistent with the partial plan committed to at each step of the search. The corresponding search path is pruned if this number is less than the total number of possible initial states.

This algorithm can be regarded as a specialized version of COMPLAN for the class of problems it has been designed to solve. As we alluded to earlier, algorithms for the more general *conformant probabilistic planning*, including COMPLAN, can be used to solve these special-case problems by assuming, for example, a uniform probability distribution over the possible initial states. The goal is then to find a plan that succeeds with probability 1. Under this setting, and with the same variable ordering constraint imposed as in (Palacios *et al.* 2005) (there C2D was instructed to instantiate the initial state variables first, so that the model counting procedure would produce an exact result rather than an upper bound), Algorithm 2 will return a number that is equal to the percentage of possible initial states for which an optimal completion of the given partial plan will be successful. This number multiplied by the total number of possible initial states is precisely the number computed by the model counting procedure used in (Palacios *et al.* 2005).

Table 2: SAND-CASTLE-67.

Horizon	Success Probability of Optimal Plan	Nodes Visited	Time
1	0.250000	3	0.00
2	0.460000	4	0.00
3	0.629650	5	0.00
4	0.727955	7	0.00
5	0.815863	8	0.00
6	0.865457	11	0.00
7	0.908290	13	0.00
8	0.933433	31	0.00
9	0.954304	57	0.00
10	0.966887	41	0.00
11	0.977229	38	0.00
12	0.983528	80	0.01
13	0.988652	158	0.01
14	0.991795	145	0.01
15	0.994345	166	0.03
16	0.995913	332	0.04
17	0.997182	346	0.08
18	0.997963	623	0.09
19	0.998596	762	0.12
20	0.998985	1491	0.26
21	0.999300	1706	0.34
22	0.999494	2858	0.59
23	0.999651	3802	0.87
24	0.999748	6544	1.60
25	0.999826	8140	2.20
26	0.999874	14039	3.98
27	0.999913	17988	10.04
28	0.999937	31009	10.26
29	0.999957	40034	14.73
30	0.999969	68772	25.31
31	0.999978	89630	35.22
32	0.999984	169325	69.74
33	0.999989	197422	86.91
34	0.999992	364504	165.94
35	0.999995	611055	301.48
36	0.999996	801568	412.48
37	0.999997	1372480	735.69
38	0.999998	1645560	943.38
39	0.999999	2155772	1312.47
40	0.999999	3656072	2306.30

It has been observed that conformant probabilistic planning can also be solved by a class of algorithms, known as *value iteration*, for POMDP (partially observable Markov decision processes). According to (Majercik & Littman 1998a; Hyafil & Bacchus 2003; 2004), these algorithms can do better than search on certain problems, although worse than search on others. In the former case, the efficiency of these algorithms can be partly ascribed to their use of dynamic programming where much redundant computation is avoided, which is analogous to caching in search. Existing caching methods for search, however, have not been able to achieve the same level of effectiveness. There is hence an opportunity for better caching methods to be devised and used with search, so that the best of the two worlds is attained. This prospect is particularly attractive given that COMPLAN only requires linear space, leaving ample room for caching to come into play.

Table 3: SLIPPERY-GRIPPER.

Horizon	Success Probability of Optimal Plan	Nodes Visited	Time
2	0.733500	6	0.00
3	0.830925	7	0.00
4	0.909401	20	0.00
5	0.967910	27	0.00
6	0.980439	70	0.01
7	0.992292	149	0.03
8	0.996130	316	0.07
9	0.998040	817	0.21
10	0.999238	1749	0.54
11	0.999525	6801	2.24
12	0.999793	14165	5.49
13	0.999913	39370	17.17
14	0.999956	89148	42.17
15	0.999980	280682	143.39
16	0.999989	976041	522.61
17	0.999996	1903248	1188.85
18	0.999998	6090724	3719.99
19	0.999999	13696382	9240.79
20	1.000000	45081357	32737.36

Table 4: GRID-10X10.

Horizon	Success Probability of Optimal Plan	Nodes Visited	Time
18	0.047016	25804	8.77
19	0.103832	4804	4.15
20	0.198188	13280	17.34
21	0.292960	6476	16.74
22	0.388362	22009	72.60
23	0.485887	12851	72.77
24	0.556308	47953	332.52
25	0.634180	28699	337.81
26	0.686256	74118	970.94
27	0.732355	74469	1231.80
28	0.766858	129245	2451.81
29	0.791786	201068	4360.65
30	0.812987	276177	6786.83
31	0.825833	523123	18584.31
32	0.838083	570143	22689.47

## Acknowledgments

The author would like to thank Adnan Darwiche, James Park, Sylvie Thiébaux, and the anonymous reviewers for their helpful comments on an earlier version of this paper. National ICT Australia is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

## References

- Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 52–61.
- Bonet, B., and Geffner, H. 2001. GPT: A tool for planning with uncertainty and partial information. In *Proceedings of the IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, 82–87.
- Brafman, R. I., and Hoffmann, J. 2004. Conformant planning

via heuristic forward search: A new approach. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, 355–364.

Cimatti, A., and Roveri, M. 2000. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research* 13:305–338.

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

Darwiche, A. 2001a. On the tractability of counting theory models and its application to belief revision and truth maintenance. *Journal of Applied Non-Classical Logics* 11(1-2):11–34.

Darwiche, A. 2001b. Recursive conditioning. *Artificial Intelligence* 126(1-2):5–41.

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/>.

Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, 211–219.

Hyafil, N., and Bacchus, F. 2003. Conformant probabilistic planning via CSPs. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 205–214.

Hyafil, N., and Bacchus, F. 2004. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 1033–1034.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, 359–363.

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.

Majercik, S. M., and Littman, M. L. 1998a. MAXPLAN: A new approach to probabilistic planning. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 86–93.

Majercik, S. M., and Littman, M. L. 1998b. Using caching to solve larger probabilistic planning problems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, 954–959.

Palacios, H.; Bonet, B.; Darwiche, A.; and Geffner, H. 2005. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 141–150.

Park, J., and Darwiche, A. 2003. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, 459–468.

Robertson, N., and Seymour, P. D. 1986. Graph minors II: Algorithmic aspects of treewidth. *Journal of Algorithms* 7:309–322.

Smith, D. E., and Weld, D. S. 1998. Conformant Graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, 889–896.