Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations

Héctor Palacios
Universitat Pompeu Fabra
Paseo de Circunvalación, 8
Barcelona, Spain
hector.palacios@upf.edu

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Adnan Darwiche
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
darwiche@cs.ucla.edu

Héctor Geffner
ICREA/Universitat Pompeu Fabra
Paseo de Circunvalación, 8
Barcelona, Spain
hector.geffner@upf.edu

Abstract

Optimal planners in the classical setting are built around two notions: branching and pruning. SAT-based planners for example branch by trying the values of a selected variable, and prune by propagating constraints and checking consistency. In the conformant setting, a similar branching scheme can be used if restricted to action variables, but the pruning scheme must be modified. Indeed, pruning branches that encode inconsistent partial plans is not sufficient since a partial plan may be consistent and complete (covering all the action variables) and still fail to be a conformant plan. This happens indeed when the plan does not conform to some possible initial state or transition. A remedy to this problem is to use a criterion stronger than consistency for pruning. This is actually what we do in this paper where the consistency-based pruning criterion used in classical planning is replaced by a validity-based criterion suitable for conformant planning. Under the assumption that actions are deterministic, a partial plan can be defined as *valid* when it is logically consistent with the theory and *each* possible initial state. A valid partial plan that is complete is guaranteed to encode a conformant plan, and vice versa. Checking validity, however, while useful for pruning can be very expensive. We show then that such validity checks can be performed in *linear time* provided that the theory encoding the problem is transformed into a logically equivalent theory in deterministic decomposable negation normal form (d-DNNF). In d-DNNF, plan validity checks can be reduced to two linear-time operations: projection (finding the strongest consequence of a formula over some of its variables) and model counting (finding the number of satisfying assignments). We then define and evaluate a conformant planner that branches on action variables, and prunes invalid partial plans in linear time. The empirical results are encouraging, showing the potential benefits of stronger forms of inference in planning tasks that are not reducible to SAT.

Introduction

Optimal planners in the classical setting are built around two notions: branching and pruning. In state-based search, classical planners branch by applying actions (forward or backwards) and prune by comparing estimated costs with a given bound. SAT-based planners, on the other hand, branch by trying the values of a selected variable, and prune by propagating constraints and checking consistency.

In principle, the same two notions can and have been used in the conformant setting, where plans must map the initial situation into a goal situation in the presence of uncertainty (Goldman & Boddy 1996; Smith & Weld 1998). Still the results in the conformant setting have not been as strong, in part due to the higher complexity of the problem (Haslum & Jonsson 1999; Rintanen 2004a), in part, due to the lack of sufficiently strong pruning criteria.

In the context of *directional* branching schemes that search for plans by applying actions forward or backwards, the problem of optimal conformant planning becomes a shortest path problem over a graph in which the nodes are sets of states or *belief states*. This is the formulation pursued in (Bonet & Geffner 2000), where the search for the goal belief state from a given initial belief state is carried out by means of the A* algorithm informed by a heuristic function obtained from a suitable relaxation. This relaxation retains the uncertainty in the model but assumes full observability, resulting in a heuristic function that is useful in certain problems, but not in those problems where reasoning by cases is not appropriate. For example, if a robot does not know whether it is at distance one or two from the goal, reasoning by cases, the robot will conclude that it is best to move toward the goal; yet since in conformant planning, it must reach the goal with certainty, a move in a different direction, that would help the robot find its true location may be necessary. In general, the assumption of full observability yields a heuristic that is not well informed for problems that include an information-gathering component, a feature that is present in many conformant problems even if they do not involve explicit observations.

The complexity of the search in belief space grows with two factors: the *size* and the *number* of belief states. The first is exponential in the number of variables; the second in the number of states. The switch to symbolic representations as in (Cimatti, Roveri, & Bertoli 2004), where sets of states are represented by OBDDs (Bryant 1992), provides a handle on the first problem but not on the second that demands more informed admissible heuristic functions. Steps in this direction have been reported recently in (Cimatti, Roveri, & Bertoli 2004) and (Rintanen 2004b).

Conformant planning can also be approached from a logical perspective, working on the theory encoding the problem, and branching on action literals until a valid plan is found. This approach, however, while so successful in the classical setting (Kautz & Selman 1996),[1] has not appeared to have worked well in the conformant setting, where pruning inconsistent partial plans is not sufficient. Indeed, a partial plan may be complete (covering all action variables) and logically consistent with the theory, and still fail to be a conformant plan. This happens when the plan does not conform to some possible initial state or transition. A remedy to this problem is the use of a criterion stronger than consistency for pruning. This is actually what we do in this paper where the consistency-based pruning criterion used in classical planning is replaced by a validity-based criterion suitable for conformant planning. Under the assumption that actions are deterministic, a partial plan can be defined as *valid* when it is logically consistent with the theory and *each* possible initial state. A valid partial plan that is complete is guaranteed to encode a conformant plan, and vice versa. Checking validity, however, while useful for pruning can be very expensive. We show then that such validity checks can be performed in *linear time* provided that the theory encoding the problem is transformed into a logically equivalent theory in deterministic decomposable negation normal form (d-DNNF) (Darwiche 2002). In d-DNNF, validity checks can be reduced to two linear-time operations: projection (finding the strongest consequence of a formula over some of its variables) and model counting (finding the number of satisfying assignments). We then define and evaluate a conformant planner that branches on action variables, and prunes invalid partial plans in linear time. The empirical results are encouraging, showing the potential benefits of stronger forms of inference in planning tasks that are not reducible to SAT.

The proposed contributions of the paper are of three types. From a practical viewpoint, a novel and promising technique for optimal conformant planning; from a theoretical point of view, a framework that matches the polynomial operations offered by d-DNNF representations with those required in conformant planning applications, and from a conceptual point of view, a perspective on conformant planning based on compiled representations that we expect may lead to further exchanges between the two areas.

[1]See (Giunchiglia, Massarotto, & Sebastiani 1998) for a similar approach that only branches on action literals.

The paper is organized as follows: we define first the conformant planning task, the propositional encoding of planning problems, and the validity of partial conformant plans in terms of projection and model counting operations. We then review the target language d-DNNF and the process of compiling formulas in CNF into d-DNNF. We finally present the planner, empirical results, and a closing discussion.

Conformant Planning

We consider conformant planning problems P given by tuples of the form $P = \langle F, O, I, G \rangle$ where F stands for the fluent symbols f in the problem, O stands for a set of deterministic actions a, and I and G are sets of clauses over the fluents in F encoding the initial and goal situations. In addition, every action a has a precondition $pre(a)$, given by a set of fluent literals, and a list of conditional effects $c^k(a) \rightarrow e^k(a)$, $k = 1, \ldots, n_a$, where $c^k(a)$ and $e^k(a)$ are conjunctions of fluent literals (as mentioned above, we assume that actions are deterministic and hence that all uncertainty lies in the initial situation).[2]

The semantics of a conformant planning P can be given in terms of a state model $S(P)$ in which the states are the possible truth assignments to the fluent symbols, and belief states are collections of states that are deemed possible. The initial belief state b_I in $S(P)$ is the set of states satisfying I, while a final belief state b_G is one made up of states satisfying G. An action a maps a belief state b into a belief state b_a if a is applicable in every state s in b, and b_a is the collection of states s_a obtained from applying a to each state s in b. Finally, a is applicable in s if $pre(a)$ is true in s, and a maps state s into s_a if s_a satisfies all the conditional effects $e^k(a)$ of a whose conditions $c^k(a)$ are true in s, and all other literals in s that are not affected by a in s.

We will take a *conformant plan* for the problem $P = \langle F, O, I, G \rangle$ to be a sequence of *sets* of actions $A^0, A^1, \ldots, A^{n-1}$ that maps the initial belief state $b_0 = b_I$ into a final belief state $b_n = b_G$. Every pair of actions in each set A^i must be *compatible.* In the sequential setting, no pair of distinct actions is compatible, while in the parallel setting, two actions are deemed compatible when the sets of boolean variables in their effects are disjoint. A conformant plan is *optimal* if the length n of the sequence is minimal. In the sequential setting, optimal plans thus minimize the number of actions, while in the parallel setting, the duration or makespan of the plan.

We assume throughout that the planning problem is consistent, and hence that the initial belief state b_I and all the belief states that are reachable from it are non-empty.

[2]As mentioned in (Smith & Weld 1998), uncertainty in action effects can actually be translated into uncertainty in the initial state by introducing a polynomial number of extra fluents in the problem; e.g. if actions have at most two effects, then it is sufficient to add a ‘hidden’ fluent for each action and time point, making the action effects deterministic but conditional on the status of the hidden fluent.

Propositional Encodings

The propositional encodings for conformant planning problems that we use is a slight variation of the propositional encodings used in the SAT approach to classical planning (Kautz & Selman 1996) where boolean variables x_i are created for fluents and actions x in the problem, and i is a temporal index that ranges from $i = 0$ up to the planning horizon $i = N$ (actually no action variables x_i are created for the last time slice $i = N$). For a formula B, we use the notation B_i to stand for the formula obtained by replacing each variable x in B by its time-stamped counterpart x_i. The encoding $T(P)$ of a conformant planning problem $P = \langle F, I, O, G\rangle$ with horizon N can then be described as follows:

1. **Init:** a clause C_0 for each init clause $C \in I$
2. **Goal:** a clause C_N for each goal clause $C \in G$.
3. **Actions:** For $i = 0, 1, \ldots, N-1$ and $a \in O$:

$$a_i \supset pre(a)_i \quad \text{(preconditions)}$$

$$c^k(a)_i \wedge a_i \supset e^k(a)_{i+1},\ k = 1, \ldots, k_a \quad \text{(effects)}$$

4. **Frame:** for $i = 0, 1, \ldots, N-1$, each fluent literal $l \in F$

$$l_i \wedge \bigwedge_{c^k(a)} \neg[c^k(a)_i \wedge a_i] \supset l_{i+1}$$

   where the conjunction ranges over the conditions $c^k(a)$ associated with effects $e^k(a)$ that 'delete' l.
5. **Exclusion:** $\neg a_i \vee \neg a'_i$ for $i = 0, \ldots, N-1$ if a and a' are incompatible.

In classical planning the relation between the encoding $T(P)$ and the planning problem P is such that the models of $T(P)$ are in one-to-one correspondence with the plans that solve P. The situation in conformant planning has to be different as conformant planning cannot be reduced to checking the consistency of propositional encodings. We thus make use of these encodings in a different way.

Validity

Given the encoding $T(P)$, we will refer to collections of *action literals* as *partial plans*, and denote them as T_A. We assume that no partial plan contains complementary or incompatible literals, and say that a partial plan is *complete* when it mentions all action variables in the theory. We will refer to a partial plan that is complete, as a *complete plan*. Clearly, a complete plan T_A may be logically consistent with the theory $T(P)$ and fail to represent a conformant plan for P if it fails to conform to some possible initial state (we are assuming that actions are all deterministic).

Let $T_0(P)$ refer to the slice of the theory $T(P)$ that represents the initial situation, let s_0 represent a state satisfying $T_0(P)$, and let $Lits(s_0)$ refer to the set of literals true in s_0. Then we define the notion of *validity* of partial plans in the conformant setting as follows:

Definition 1 (Validity) *A partial plan T_A is* valid *in the context of a domain theory $T(P)$, if and only if for* every *possible state s_0 satisfying $T_0(P)$, the set of formulas given by T_A, $T(P)$ and $Lits(s_0)$ is logically consistent.*

This definition has two desirable properties that we will exploit in the branching scheme used to search for conformant plans. The first is that a complete plan that is valid represents a conformant plan and vice versa: a conformant plan represents a valid complete plan. The second, and not least important, is that an incomplete partial plan that is not valid cannot lead to a conformant plan. We state these properties as follows:

Theorem 1 *1) A complete plan T_A that is valid in the context of $T(P)$ encodes a candidate plan $A^0, \ldots, A^{N-1}$ that is conformant with respect to P where $a \in A^i$ iff $a_i \in T_A$. 2) A conformant plan $A^0, \ldots, A^{N-1}$ for P implies that the complete plan T_A is valid with respect to $T(P)$, where $a_i \in T_A$ iff $a \in A^i$, and $\neg a_i \in T_A$ iff $a \notin A^i$. 3) A partial plan T_A that is not valid in the context of $T(P)$ cannot be extended into a valid complete plan, and hence cannot lead to a conformant plan.*

These properties ensure the soundness and completeness of a simple branch and prune algorithm that branches on action literals, prunes partial plans that are not valid, and terminates when a non-pruned complete plan is found. Of course, this simple algorithm would not be necessarily efficient as it involves an expensive validity check in every node of the search tree, which if done naively, would involve a number of satisfiability tests linear in the number of possible initial states.

A main goal of the paper is to provide a formulation in which this branch and prune algorithm can be run efficiently. The key idea will involve the compilation of the planning theory $T(P)$ into a suitable target language in which some normally intractable operations on boolean formulae become tractable and fast. The two main boolean operations that we will need are *projection* and *model counting*:

- the *projection* of a formula Δ over a subset V of its variables, denoted as $Project(\Delta, V)$, stands for the strongest formula over the V variables implied by Δ. Such formula is unique up to logical equivalence (the projection operation is dual to 'forgetting' (Lin & Reiter 1994)).
- the *model count* of a formula Δ, denoted as $MC(\Delta)$, stands for the number of truth assignments that satisfy the formula.

With these two operations, and if we let F_0 refer to the set of fluent variables f_0 at time $i = 0$ and $T_0(P)$ refer to the slice of $T(P)$ encoding the initial situation, the validity check from Definition 1 can be rephrased as follows:

Theorem 2 (Validity by Projection and MC) *A (partial) plan T_A is* valid in the context of a domain theory $T(P)$ iff

$$MC(T_0(P)) = MC(Project(T(P) + T_A, F_0)) \,. \quad (1)$$

The theorem reduces the validity check of a partial plan T_A to the comparison of two numbers: the number of possible initial states, and the number of initial states compatible with the domain theory and the commitments made in T_A. Clearly, the second number cannot be greater than the first, as $T(P)$ alone entails $T_0(P)$ ($T_0(P)$ is part of $T(P)$). Yet the second number can be smaller: this would happen precisely when some possible initial state s_0 is not compatible with $T(P)$ and T_A, which according to Definition 1, is exactly the situation in which T_A is not a valid partial plan.

Having discussed a branch and bound algorithm for conformant planning based on validity checks which can be computed by means of projection and model count operations, we turn to a target representation language that renders these two operations tractable and efficient.

Deterministic DNNF

Knowledge compilation is the area in AI concerned with the problem of mapping logical theories into suitable target languages that make certain desired operations tractable (Selman & Kautz 1996; Cadoli & Donini 1997). For example, propositional theories can be mapped into their set of Prime Implicates making the entailment test of clauses tractable (Reiter & de Kleer 1987). Similarly, the compilation into Ordered Binary Decision Diagrams (OBDDs) renders a large number of operations tractable including model counting (Bryant 1992). While in all these cases, the compilation itself is intractable, its expense may be justified if these operations are to be used a sufficiently large number of times in the target application. Moreover, while the compilation will run in exponential time and space in the worst case, it will not necessarily do so on average. Indeed, the compilation of theories into OBDDs has been found useful in formal verification (Clarke, Grumberg, & Peled 1999) and more recently in planning (Giunchiglia & Traverso 1999). A more recent compilation language is Decomposable Negation Normal Form (DNNF) (Darwiche 2001). DNNFs support a rich set of polynomial–time operations, some of which are particularly suited to our application, like *projection* on an arbitrary set of variables, which can be performed simply and efficiently. A subset of DNNF, known as deterministic DNNF, also supports *model counting,* which is critical to our application as shown later.

Decomposability and determinism of NNF

A propositional sentence is in negation normal form (NNF) if it is constructed from literals using only conjunctions and disjunctions (Barwise 1977). A practical representation of NNF sentences is in terms of rooted directed acyclic graphs (DAGs), where each leaf node in the DAG is labeled with a literal, true or false; and each non-leaf (internal) node is labeled with a conjunction $\wedge$ or a disjunction $\vee$; see Figure 1. Decomposable NNFs are defined as follows:

Definition 2 *(Darwiche 2001) A* decomposable negation normal form (DNNF) *is a negation normal form satisfying the* decomposability property: *for any conjunction $\wedge_i \alpha_i$ in the form, no variable appears in more than one conjunct α_i.*

The NNF in Figure 1 is decomposable. It has ten conjunctions and the conjuncts of each share no variables. Decomposability is the property which makes DNNF tractable: a decomposable NNF formula $\wedge_i \alpha_i$ is indeed satisfiable iff every conjunct α_i is satisfiable, while $\vee_i \alpha_i$ is satisfiable (always) iff some disjunct α_i is. The satisfiability of a DNNF can thus be tested in linear time by means of a single bottom up pass over its DAG.

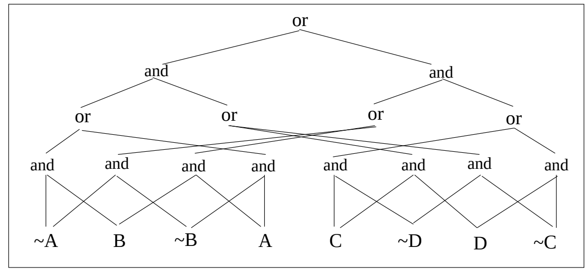

Figure 1: A negation normal form (NNF) represented as a rooted DAG.

The NNF $(A \vee B) \wedge (\neg A \vee C)$ is not decomposable since variable A is shared by the two conjuncts. Any such form, however, can be converted into DNNF. The main technique to use here is that of performing *case analysis* over the variables that violate the decomposition property, in this case A. Assuming that A is true, the NNF reduces to C, while if A is false, the NNF reduces to B. The result is the NNF $(A \wedge C) \vee (\neg A \wedge B)$ which is decomposable, and hence, a DNNF.[3]

The above principle can be formulated more precisely using the notion of *conditioning,* which reduces an NNF Δ given a literal α. Specifically, the conditioning of Δ on literal α, written $\Delta|\alpha$, is obtained by simply replacing each leaf α in the NNF DAG by true and each leaf $\sim\alpha$ by false—from now on, $\sim\alpha$ denotes the complement of literal α. If $\Delta = (A \vee B) \wedge (\neg A \vee C)$, then $\Delta|A$ is $(\mathsf{true} \vee B) \wedge (\mathsf{false} \vee C)$ which simplifies to C. Similarly, $\Delta|\neg A$ is $(\mathsf{false} \vee B) \wedge (\mathsf{true} \vee C)$ which simplifies to B. The case analysis principle can now be phrased formally as follows:[4]

$$\Delta \equiv (\Delta|A \wedge A) \vee (\Delta|\neg A \wedge \neg A) \tag{2}$$

This will actually be the pattern for decomposing propositional theories as we shall see later. For now though, we point out that the split exhibited by (2) above, leads to a second useful property called *determinism*, giving rise to the special class of *Deterministic DNNFs*:

Definition 3 *(Darwiche 2002) A* deterministic DNNF (d-DNNF) *is a DNNF satisfying the* determinism property: *for any disjunction $\vee_i \alpha_i$ in the form, every pair of disjuncts α_i is mutually exclusive.*

Determinism is the property which makes *model counting* over DNNFs tractable: the number of models of a DNNF

[3]Disjunctive Normal Form (DNF), with no literal sharing in terms, is a subset of DNNF. The DNF language, however, is *flat* as the height of corresponding NNF DAG is no greater than 2. This restriction is significant as it reduces the succinctness of DNF as compared to DNNF. For example, it is known that the DNF language is incomparable to the OBDD language from a succinctness viewpoint, even though DNF and OBDD are both strictly less succinct than DNNF (Darwiche & Marquis 2002).

[4]This principle is also known as Boole's expansion and Shannon's expansion.

$\wedge_i \alpha_i$ is the *product* of the number of models of each conjunct α_i, while the number of models of a DNNF $\vee_i \alpha_i$ that satisfies determinism is the *sum* of the number of models of each disjunct.[5]

The final key operation on DNNFs that we need is *projection.* The projection of a theory Δ on a set of variables V is the strongest sentence implied by Δ over those variables. This sentences is unique up to logical equivalence and we will denote it by $Project(\Delta, V)$. Projection is dual to *elimination* or *forgetting* (Lin & Reiter 1994): that is, projecting Δ on V is equivalent to eliminating (existentially quantifying) all variables that are *not* in V from Δ. Like satisfiability on DNNFs, and model counting on d-DNNFs, projection on DNNFs can be done in linear time (Darwiche 2001). Specifically, to project a DNNF on a set of variables V, all we have to do is replace every literal in Δ by true if that literal mentions a variable outside V. For example, the projection of DNNF $(A \wedge \neg B) \vee C$ on variables B and C is the DNNF $(\mathsf{true} \wedge \neg B) \vee C$, which simplifies to $\neg B \vee C$. Moreover, the projection on variable C only is the DNNF $(\mathsf{true} \wedge \mathsf{true}) \vee C$, which simplifies to true.

Testing plan validity using d-DNNF

The use of d-DNNFs has been motivated by the desire to make the validity test for partial plans tractable and efficient. The test, captured by (1), involves computing the model count of a projection. We have seen that we can count the models of d-DNNF and take the projection of a DNNF in linear time. This may suggest that we can render the validity test for partial plans linear in the size of the d-DNNF representation. This however is not true in general; the problem is that the linear–time projection operation given above is guaranteed to preserve decomposability but not necessarily determinism. This means that while we can model count and project a deterministic DNNF in linear time, we cannot always model count the projection of a deterministic DNNF in linear time, which is precisely what we want. There are however two conditions under which the projection $Project(\Delta, V)$ of a deterministic DNNF Δ on variables V can be guaranteed to remain deterministic and allow for model counting in linear time. The first condition is that variables $\overline{V}$ which are projected away are *determined* in Δ by variables V that are kept (i.e., the values of variables $\overline{V}$ in any model of Δ are determined by the values of variables V). This condition holds in our setting for the fluent variables f_i for $i > 0$ which are determined by the initial fluent variables f_0 and the action variables a_i, $i = 1, \ldots, N{-}1$. We actually use this result to project the compiled d-DNNF on the initial state variables and on action variables, leaving out all other variables at the outset. The second condition relates to an ordering restriction that we can impose on the splits given by (2) above; we discuss this restriction in the following section.

[5] Actually, in order to get a normalized count it must be made sure that the same set of variables appear in the different disjuncts of the d-DNNF. However, this property called 'smoothness' is easily enforced (Darwiche 2002).

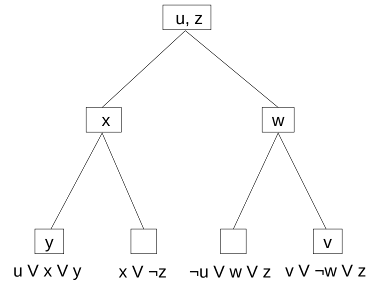

Figure 2: A decomposition tree for a CNF.

Compiling planning theories into d-DNNF

A propositional theory Δ can be compiled into d-DNNF by simply ordering the variables appearing in Δ in a sequence $x_1, \ldots, x_n$, and then splitting Δ first on x_1 leading to $(\Delta|x_1 \wedge x_1) \vee (\Delta|\neg x_1 \wedge \neg x_1)$, and then compiling recursively each of the conditioned theories $\Delta|x_1$ and $\Delta|\neg x_1$ using the suborder $x_2, \ldots, x_n$. Coupled with a caching scheme to avoid compiling the same theory multiple times, the above technique will lead to d-DNNFs that are isomorphic to OBDDs. In fact, this particular method for compiling OBDDs, which deviates from the vast tradition on this subject, was explored recently in (Huang & Darwiche 2004). Deterministic DNNFs, however, are known to be strictly more space efficient than OBDDs (Darwiche & Marquis 2002), and indeed a more efficient compilation scheme is possible (Darwiche 2004). In particular, if during this top–down compilation process one gets to an instantiated theory Δ' of the form $\Delta'_1 \wedge \Delta'_2$ such that Δ'_1 and Δ'_2 share no variables, then the compilation of Δ' can be *decomposed* into the conjunction of the compilation of Δ'_1 and the compilation of Δ'_2. Moreover, one does not need to use a fixed variable order as required by OBDDs, but can choose variables dynamically to split on, typically, to try to maximize the opportunities for decomposition.

Although dynamic variable ordering and decomposition appear to be the reasonable strategy to adopt in this context, experience has shown that it may incur an unjustifiable overhead. The d-DNNF compiler we use is instead based on semi–dynamic variable orderings, which are obtained by a pre–processing step to reduce the overhead during compilation (Darwiche 2004). In particular, before the compilation process starts, one constructs a *decomposition tree (dtree)* as shown in Figure 2 (Darwiche 2001). This is simply a binary tree whose leaves are tagged with the clauses appearing in the CNF to be compiled. Each internal node in the dtree corresponds to a subset of the original CNF and is also tagged with a *cutset:* a set of variables whose instantiation is guaranteed to decompose the CNF corresponding to that node. The compiler will then start by picking up variables from the root cutset to split on until the CNF corresponding to the root is decomposed. It will then recurse on the left and right children of the root, repeating the same process again. Within each cutset (which can be large), the compiler chooses variable order dynamically. Note also that the dtree imposes only a partial order on cutsets, not a total order.

A number of further features are incorporated into the d-DNNF compiler we use (Darwiche 2004). Examples include the use of caching to avoid compiling identical theories twice; unit propagation for simplifying theories; dependency directed backtracking; and clause learning.

The key benefit of d-DNNF compilers in relation to OBDD compilers is that the former make use of *decomposition.* Hence, for example, the complexity of d-DNNF compilations are known to be exponential only in the treewidth of the theory, while OBDD compilations are exponential in the pathwidth, which is no less than the treewidth and usually much bigger (McMillan 1994; Darwiche 2004; Dechter 2004). Another advantage of using d-DNNFs over OBDDs is that d-DNNFs employ a more general form of determinism allowing us to use the linear–time projection operation discussed earlier, while still preserving both decomposability and determinism in some cases. This feature allowed us to project compiled planning theories on the initial state fluents and the actions in linear time. OBDDs do not support a linear–time operation for projection under the same conditions (Darwiche & Marquis 2002).

Finally, we note that we had to use specific decomposition trees to allow us to generate d-DNNFs which can be projected on the initial state fluents *only* while preserving determinism—this is needed to implement the plan validity test in (1) which is critical for pruning during search. In particular, we had to construct dtrees in which initial state fluents are split on before any other variables are split on during case analysis. This guarantees that determinism would be preserved when projecting the d-DNNF on initial state fluents as it guarantees that every remaining disjunctions will be of the form $(f_0 \wedge \alpha) \vee (\neg f_0 \wedge \beta)$ where f_0 is an initial state fluent.[6]

The Conformant Planner

We have implemented a validity-based optimal conformant planner called vplan that accepts the description P of a conformant planning problem and a planning horizon N, and then produces a valid conformant plan with N time steps at most if one exists, else reports failure. If the horizon is incremented by 1 starting from $N = 0$, the first plan found is guaranteed to be optimal. The planner can be run in sequential or parallel mode according to the sets of concurrent actions allowed. vplan translates P into a domain theory T in d-DNNF and then performs a backtrack search for a valid conformant plan by performing operations on the theory: branching on action literals, pruning invalid sets of action literals (partial plans), and terminating when a non-pruned complete plan is found. More precisely, the planner can be characterized by the following aspects:

- **Preprocessing:** the problem P with a given horizon N is translated into a CNF theory $T(P)$, which is compiled into a d-DNNF theory T which is associated with the root node n_r of the search tree; i.e. $T(n_r) = T$.[7]
- **Branching:** at a node n in the search tree, the planner branches by selecting an undetermined action variable a_i and trying each of its possible values; namely, two d-DNNF theories T_{n_1} and T_{n_2} are created for the children nodes n_1 and n_2 of n that correspond to $T_n|a_i$ and $T_n|\neg a_i$. This process continues depth-first until a node is pruned, resulting in a backtrack, or all action variables are determined, resulting in a valid conformant plan.
- **Pruning:** a node n is pruned when the d-DNNF theory T_n associated with n fails the *validity test*:
$$MC(T_0) = MC(Project(T_n, F_0)) \qquad (3)$$
where T_0 stands for the slice of the theory T encoding the initial situation, MC stands for the model count operator, and F_0 stands for the fluent variables in the initial situation. The model count and the projection are done in linear time by means of a single bottom-up pass over the DAG representation. The model count over T_0 is done once, and measures the number of possible initial states.
- **Selection Heuristics and Propagation:** The undetermined action variable a_i for branching in node n is selected as the positive action literal a_i that occurs in the greatest number of models of T_n; this ranking being obtained by means of a single model count $MC(T_n)$ implemented so that with just two passes over the DAG-representation of T_n (one bottom up, another top-down; see (Darwiche 2002)), it yields model counts $MC(T_n \wedge l)$ for all literals l in the theory. Moreover, when for a yet undetermined action literal l this model count yields a number which is smaller than the number of initial states $MC(T_0)$, then its complement $\sim l$ is set to true by conditioning T_n on $\sim l$. This process is iterated until no more literals can be so conditioned on in T_n. This inference cannot be captured by performing deductions on T_n as this could only set a literal $\sim l$ to true when the model count $MC(T_n \wedge l)$ is exactly 0. The inference, however, follows from the *QBF formula* associated with T_n encoding not only the planning *domain* but the planning *task* (Rintanen 1999).[8]

---

[6]A similar technique can be used if one compiles into OBDDs, by having initial state fluents first in the OBDD order.

[7]A subtlety in the translation from P to the CNF theory $T(P)$ are the frame axioms which may generate an exponential number of clauses. In order to avoid such explosion, each conjunction $c^k(a)_i \wedge a_i$ of a condition $c^k(a)_i$ and the corresponding action a_i, is replaced by a new auxiliary variable z_i and the equivalence $z_i \equiv c^k(a)_i \wedge a_i$ is added to the theory. Such auxiliary variables do not affect the compilation into d-DNNF as they are 'implied variables' that are safely projected away when the CNF theories are compiled into d-DNNF.

[8]The QBF formulation of conformant planning reflects that we are not looking for models of T_n but for interpretations over the action variables that can be extended into models of T_n for any choice of the initial fluent variables compatible with T_0. The QBF formula encoding the planning task over the theory $T(P)$ will imply that an action literal a_i that does not participate in any conformant plan that solves P needs to be false. This, however, does not mean that $\neg a_i$ is a deductive consequence of the planning theory $T(P)$; it is rather a deductive consequence of the QBF formula encoding the planning *task*. This distinction does not appear in the classical setting, where the same formula encodes the planning theory and the planning task; it is however important in the conformant setting where this is no longer true.

problem	N^*	CNF theory vars	clauses	d-DNNF theory nodes	edges	time/acc
blocks-2	2	34	105	61	97	0.03/0.06
blocks-3	9	444	2913	4672	20010	0.25/1.13
blocks-4	26	3036	40732	225396	913621	77.5/752.65
sq-center-2	8	200	674	1000	2216	0.1/0.39
sq-center-3	20	976	3642	9170	19555	0.7/6.7
sq-center-4	44	4256	16586	79039	164191	31.17/512.54
ring-3	8	209	669	2753	6161	0.11/0.48
ring-4	11	364	1196	13239	29295	0.62/2.52
ring-5	14	561	1874	60338	132045	3.68/16.4
ring-6	17	800	2703	254379	551641	23.77/120.58
ring-7	20	1081	3683	1018454	2195393	221.58/1096.7
ring-8	23	1404	4814	3928396	8406323	2018.32/12463.3
sortnet-3	3	51	122	133	230	0.03/0.09
sortnet-4	5	150	409	1048	2325	0.04/0.19
sortnet-5	9	420	1343	7395	17823	0.51/1.4
sortnet-6	12	813	3077	30522	77015	1.28/7.12
sortnet-7	16	1484	6679	116138	294840	8.29/56.61
sortnet-8	19	2316	12364	369375	931097	56.73/427.58
sortnet-9	25	3870	24414	1264508	3075923	780.77/6316.53

Table 1: Compilation data for sequential planning. N^* is the optimal planning horizon. Nodes and edges refer to the DAG-representation of the generated d-DNNF. Time refers to the compilation time for the theory with horizon N^*, and 'acc' to the sum of all compilation times for horizons $N = 0, \dots, N^*$. All times are in seconds.

As mentioned above, in order to perform the pruning test (3) efficiently in every node n, we must ensure that the projection operation preserves determinism. This is ensured by compiling the CNF theory $T(P)$ using a decomposition tree in which the splits on the variables belonging to F_0 (the fluent variables f_0 for the initial situation) are done before any other splits. Also, for having an equivalent but smaller d-DNNF we project away all fluent variables f_i from the theory for $i > 0$ at compilation time. Such fluents are not needed, and their elimination satisfies the other condition above: they are determined by the initial fluent and action variables that are kept.

Experimental Results

We performed the experiments on a Intel/Linux machine running at 2.80GHz with 2Gb of memory. Times for the experiments were limited to 2 hours and memory to 1.Gb. We used the same suite of problems as (Rintanen 2004b). These are challenging problems that emphasize some of the critical aspects that distinguish conformant from classical planning; some of the problems are from (Cimatti, Roveri, & Bertoli 2004):

- **Ring:** There are n rooms arranged in a circle and a robot that can move clockwise or counter-clockwise, one step a a time. The room features windows that can be closed and locked. Initially, the position of the robot and the status of the windows are not known. The goal is to have all windows closed and locked. The number of initial states is $n \times 3^n$ and the optimal plan has $3n - 1$ steps. The parameter n used ranges from 3 to 8.
- **Sorting Networks:** The task is to build a circuit made of compare-and-swap gates that maps an input vector of n boolean variables into the corresponding sorted vector. The compare-and-swap action compares two entries in the input vector and swaps their contents if not ordered. The optimal sequential plans minimize the number of gates, while the optimal parallel plans minimize the 'time delay' of the circuit. Only optimal plans for small n are known (Knuth 1973). The number of initial states is 2^n. The parameter n used ranges from 2 to 7.
- **Square-center:** A robot without sensors moves in a room to north, south, east, and west, and its goal is to get to the middle of the room. The optimal sequential plan for a grid of size 2^n with an unknown initial location is to do $2^n - 1$ moves in one direction, $2^n - 1$ moves in an orthogonal direction, and then from the resulting corner, $2^n - 2$ moves to the center for a total of $3 \times 2^n - 4$ steps. In the parallel setting, pairs of actions that move the robot in orthogonal directions are allowed. There are 2^{2n} initial states. The parameter n used ranges from 2 to 4.
- **Blocks:** Refers to blocksworld domain with move-3 actions but in which the initial state is completely unknown. Actions are always applicable but have an effect only if their normal 'preconditions' are true. The goal is get a fixed ordered stack with n blocks. The parameter n used ranges from 2 to 4, and the number of initial states is 3, 13 and 73 respectively.

None of the problems feature preconditions, and only sorting and square-center admit parallel solutions (recall that we only allow parallel actions whose effects, ignoring their conditions, affect different variables).

The results that we report are collected in four tables. Table 1 reports the data corresponding to the compilation of the theories for sequential planning. The last column shows the time taken for compiling each theory with the optimal horizon N^*, and the accumulated time for compiling each theory with horizon $N = 0, \dots, N^*$. All theories compile: most in a few seconds, some in a few minutes, and only two

problem	N^*	$\#S_0$	search at horizon k time	backtracks	#act	search at horizon $k-1$ time	backtracks
blocks-2	2	3	0	1	2	0	1
blocks-3	9	13	0.02	7	9	144.45	248619
blocks-4	26	73	$>$ 2h	$>$ 76029		$>$ 2h	$>$ 78714
sq-center-2	8	16	0	0	8	0.02	243
sq-center-3	20	64	0.05	0	20	$>$ 2h	$>$ 3741672
sq-center-4	44	256	$>$ 2h	$>$ 188597		$>$ 2h	$>$ 191030
ring-3	8	81	0	0	8	0	5
ring-4	11	324	0.06	1	11	0.02	5
ring-5	14	1215	0.71	2	14	0.16	5
ring-6	17	4374	3.49	4	17	0.69	5
ring-7	20	15309	24.48	5	20	3.35	5
ring-8	23	52488	128.64	7	23	13.08	5
sortnet-3	3	8	0	0	3	0	5
sortnet-4	5	16	0	0	5	0.05	421
sortnet-5	9	32	0.02	0	9	$>$ 2h	$>$ 4845305
sortnet-6	12	64	0.2	1	12	$>$ 2h	$>$ 458912
sortnet-7	16	128	$>$ 2h	$>$ 102300		$>$ 2h	$>$ 104674

Table 2: Search data for sequential planning for optimal horizon N^* (left) and and suboptimal horizon $N^* - 1$ (right). The columns show the optimal horizon, number of possible initial states, search time in seconds, number of backtracks, and number of actions in the plan. Rows with '$> 2h$' mean the search reached the cutoff time of 2 hours. All times are in seconds.

of them – the largest ring and sort instance – take 33 and 13 minutes respectively. The accumulated times are also largest for these two instances, taking a total time of 3.4 and 1.7 hours. It is quite remarkable that all these theories actually compile; they are not trivial theories, with many featuring several thousands of variables and clauses, producing large d-DNNFs with millions of nodes in some cases (e.g., the largest sort and two largest ring instances). The largest instances, except for the ring instances, are probably beyond the reach of most conformant planners, whether optimal or not, with the planner in (Rintanen 2004b) producing apparently the best results and solving most instances, except for the 3 most difficult sorting problems. From the point of view put forward in the paper, this means that the compilation is not the bottleneck for solving these problems, but the search. However, in cases in which the d-DNNFs obtained are very large, the advantage of an informative and linear pruning criterion decreases, as the operations are linear on a structure that is very large (of course, there is no escape from this in the worst case, as checking the validity of a candidate conformant plan is hard). We also note though that some instances are quite challenging for conformant planning even if the size of their corresponding d-DNNFs are not that large; this includes sortnet-6 and sq-center-4.

Table 2 reports the data for the search for plans in the sequential setting. On the left we show the results for the optimal horizon N^*, while on the right, for the horizon $N^* - 1$ for which there is no solution. In a sense, the first results shows the difficulty of finding conformant plans; the second, the difficulty of proving them optimal. There are actually several examples in which plans are found for the optimal horizon which cannot be proved optimal in the immediately lower horizon; for example, sq-room-3 and sortnet-6. The problems that are solved most easily are the ring problems that actually are the ones that have the largest d-DNNF representation. The reason is that the pruning criterion enables the solution of such instances with very few backtracks. On the other hand, the hardest block, sq-center, and sortnet problems are not solved. By looking at the table, it appears that problems are solved with a few backtracks or are not solved at all. In principle it may be thought that this is because the pruning criterion is too expensive, and node generation rate is very low. Yet, the number of backtracks in some of the problems suggest otherwise: e.g., sortnet-5 cannot be proved optimal after almost 5 million backtracks, and similarly sq-center-3. The complexity of the pruning operation, that grows linearly with the size of the d-DNNF representation explains however why the large unsolved instances reach the cutoff time with a smaller number of backtracks than the small unsolved instances.

Table 3 reports the data for the search for plans in the parallel setting. Given our simple model of parallelism where the only compatible actions are the ones that involve disjoint sets of variables, it turns out that only the sq-center and sortnet problems admit parallel solutions. In the first, one can take orthogonal directions at the same time; in the second, one can compare disjoint pairs of wires concurrently. The instances that get solved do not change significantly with respect to the sequential setting, yet there are two interesting exceptions. One is sq-center-3 which could not be solved before for the horizon $N^* - 1$ now is solved in less than 11 seconds with a relatively large number of backtracks: 11981 (by solving a problem in the horizon $N^* - 1$ we mean proving failure). This breaks the pattern observed earlier where problems were solved almost backtrack-free or not solved at all. In the same instance, the solution found for the optimal horizon N^* is obtained in a slightly more time, but with many more backtracks: 1517. A possible explanation for this is that parallelism removes some symmetries in the problem, leaving an smaller space with fewer solutions. Thus, the proof for solutions become more difficult but the proofs for non-solutions become simpler. At the

problem	N^*	$\#S_0$	search at horizon k time	backtracks	#act	search at horizon $k-1$ time	backtracks
sq-center-2	4	16	0	54	8	0	9
sq-center-3	10	64	1.26	952	20	39.82	20773
sq-center-4	22	256	$>$ 2h	$>$ 235696		$>$ 2h	$>$ 240089
sortnet-4	3	16	0.01	38	6	0	29
sortnet-5	5	32	0.03	0	10	53.63	61469
sortnet-6	5	64	380.15	23884	14	$>$ 2h	$>$ 634880
sortnet-7	6	128	3.48	0	18	$>$ 2h	$>$ 84881

Table 3: Search data for parallel planning for optimal horizon N^* (left) and and suboptimal horizon N^*-1 (right). The columns show the optimal horizon, number of possible initial states, search time in seconds, number of backtracks, and number of actions in the plan. Rows with '$>2h$' mean the search reached the cutoff time of 2 hours. All times are in seconds.

same time though, the parallel formulation makes another problem solvable in the optimal horizon: sort-7. This is actually a difficult problem that is not solved by any planner that we know, including (Rintanen 2004b). We are not solving it fully either; it is solved for the optimal horizon N^*, but not for N^*-1 (which is always the most difficult horizon for proving the lack of solutions).

From the benchmarks considered and reported on in various papers, it is not simple to assess the performance of the proposed planner in relation to existing optimal and non-optimal ones. It seems that various planners do well for some type of problems but not for others. In particular, our own GPT planner (Bonet & Geffner 2000) does well in problems where the *size* of the belief states that are reachable is small, and the heuristic V^*_{dp} that relaxes the problem assuming full observability, remains well informed (this also requires that the size of the state space not be too large either). The planner MBP reported in (Cimatti, Roveri, & Bertoli 2004) extends the scope of heuristic search planners by representing belief states symbolically as OBDDs; so it is not affected necessarily by the size of the state space nor by the size of the belief states. Still, when running in optimal model, MBP depends on the quality of a heuristic function similar to V^*_{dp}, and then it is also somewhat bound to problems where the assumption of full observability does not simplify the problem too much (for such problems actually a better informed admissible heuristic, although not fully automated is discussed in (Cimatti, Roveri, & Bertoli 2004)). The CFF planner recently introduced in (Brafman & Hoffmann 2004) seems to perform best in problems that add a small amount of uncertainty in otherwise large classical planning problems, where the proposed, novel heuristic appears to work best. The problems that we have selected, which correspond to those considered in (Rintanen 2004b), do not appear to exhibit these features, and gathering from the reported papers, it does not seem that these planners would do well on them, or even as well as vplan (with the exception of the ring problems, that involve large state spaces and large belief states, but where the heuristic V^*_{dp} remains well informed making the symbolic heuristic-search approach particularly suitable). In particular, we considered the 'cube-center' problem, a problem that extends the 'square-center' problem with another dimension. In (Brafman & Hoffmann 2004) cubes of sizes up to $m=3$ are reported solvable by CFF and cubes of sizes up to $m=5$ are reported solvable by MBP. We ran this benchmark for vplan and as for square-center, and obtained better results in the parallel formulation, where some symmetries are broken. In this way, we were able to solve cube-center for $m=7$ in less than a minute for the optimal horizon $N^*=8$, proving the lack of solutions for the horizon N^*-1 in 485 seconds.

The planner reported in (Rintanen 2004b) does particularly well on the suite of problems considered, solving most of them very fast. The planner is a heuristic-search planner based on OBDD representations of belief states that uses a novel heuristic function obtained from relaxations that do not assume full observability. The relaxations appear to stand for conformant planning problems that differ from the original instance in their initial belief state. Rintanen solves the problem for all possible initial belief states with two states at most, and stores the resulting costs in memory. Then, the heuristic $h(b)$ of a belief state b is set to $\max_{b' \subseteq b} h'(b')$ where h' is the stored cost function, and b' is a belief state with at most two states in b. Unfortunately, Rintanen does not report data on the costs of preprocessing, but the results suggest that the heuristic obtained, while more expensive, is more suited than the simpler heuristic V^*_{dp} for problems that involve some form of epistemic reasoning.

Discussion

We have developed an algorithm for conformant planning with deterministic actions that operates over logical encodings in which action literals are selected for branching, and branches that encode invalid partial plans are pruned. The validity test checks at every node of the search tree whether the accumulated set of commitments (or partial plan) is consistent with each possible initial state and the planning theory. This ensures that the planner is sound and complete. Validity tests, however, are expensive. We showed then how they can be reduced to projection and model count operations that can be carried out efficiently in the d-DNNF representation of the planning theory. The empirical results are encouraging, although there is still a lot of room for improvement. Some goals for the future are: better ways for dealing with symmetries in the search space (there are plenty), better preprocessing (e.g., inference in the style of the planning graph capturing that certain actions literals cannot participate in a conformant plan), better criteria for selecting the action on which to branch (the planner is sensitive to this choice, and we should probably explore the use of one criterion for finding plans, and another one for proving optimality as done often in CSPs), and other ways for using the d-DNNF representation to cut the search for plans even further. In this sense, we have found that it is actually possible to compile the planning theory into d-DNNF in such a way that conformant plans can be extracted with *no search* at all by a single bottom-up pass similar to the one used to perform the model count of the projected formula. For this, however, it is necessary to use a dtree with all action variables on top, which leads to large d-DNNFs that render the compilation unfeasible except for the smallest problems. There is thus a search-inference tradeoff that has to do with the dtree used in the compilation that is worth studying further. There are also other uses of the proposed approach; for example, it is straightforward to adjust the branch-and-prune scheme for computing plans that conform with a fraction of possible initial states, or with the highest such fraction for a given planning horizon. The proposed framework is thus rich in possibilities and uses, while suggesting possible improvements in performance that we are currently exploring. The code and problems will be available in our Web page.

Acknowledgements

We thank the members of the Texas Action Group for the use of CCalc in the initial stages of this work, and A. Frangi for the use of the Hermes Computing Resource at the Aragon Inst. of Engr. Research (I3A), U. of Zaragoza. H. Geffner is partially supported by grant TIC2002-04470-C03-02 from MCyT/Spain, B. Bonet by grant DI-CAI-001-04 USB/Venezuela, and A. Darwiche by MURI grant N00014-00-1-0617 ONR/USA.

References

Barwise, J., ed. 1977. *Handbook of Mathematical Logic*. North-Holland.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. *Proc. 6th Int. Conf. on Artificial Intelligence Planning and Scheduling*, 52–61. AAAI Press.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. *Proc. 14th Int. Conf. on Automated Planning and Scheduling*. AAAI Press.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered bynaru-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

Cadoli, M., and Donini, F. 1997. A survey on knowledge compilation. *AI Communications* 10(3-4):137–150.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159:127–206.

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. MIT Press.

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

Darwiche, A. 2001. Decomposable negation normal form. *J. of the ACM* 48(4):608–647.

Darwiche, A. 2002. On the tractable counting of theory models and its applications to belief revision and truth maintenance. *J. of Applied Non-Classical Logics*.

Darwiche, A. 2004. New advances in compiling cnf into decomposable negation normal form. *Proc. 16th European Conf. on Artificial Intelligence*, 328–332. IOS Press.

Dechter, R. 2004. And/or search spaces for graphical models. Technical report, UC Irvine. ICS Technical Report 115, at `http://www.ics.uci.edu/~csp/r118.pdf`.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. *Proc. 5th European Conf. on Planning*. Springer: LNCS 1809.

Giunchiglia, E.; Massarotto, A.; and Sebastiani, R. 1998. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. *Proc. 15th Nat. Conf. on Artificial Intelligence*, 948–953. AAAI Press.

Goldman, R. P., and Boddy, M. S. 1996. Expressive planning and explicit knowledge. *Proc. 3rd Int. Conf. on Artificial Intelligence Planning Systems*. AAAI Press.

Haslum, P., and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. *Proc. 5th European Conf. on Planning*, 308–318. Springer: LNCS 1809.

Huang, J., and Darwiche, A. 2004. Using DPLL for efficient OBDD construction. *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing*, 127–136.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. *Proc. 13th Nat. Conf. on Artificial Intelligence*, 1194–1201. AAAI Press.

Knuth, D. E. 1973. *The Art of Computer Programming, Vol. III: Sorting and Searching*. Addison-Wesley.

Lin, F., and Reiter, R. 1994. Forget it! *Working Notes, AAAI Fall Symposium on Relevance*, 154–159. American Association for Artificial Intelligence.

McMillan, K. L. 1994. Hiearchical representation of discrete functions with applications to model checking. *Proc. 6th Int. Conf. on Computer Aided Verification*.

Reiter, R., and de Kleer, J. 1987. Foundations of assumption-based truth maintenance systems: a preliminary report. *Proc. 6th Nat. Conf. on Artificial Intelligence*, 183–188. AAAI Press.

Rintanen, J. 1999. Constructing conditional plans by a theorem prover. *J. of Artificial Int. Research* 10:323–352.

Rintanen, J. 2004a. Complexity of planning with partial observability. *Proc. 14th Int. Conf. on Automated Planning and Scheduling*, 345–354. AAAI Press.

Rintanen, J. 2004b. Distance estimates for planning in the discrete belief space. *Proc. 19th Nat. Conf. on Artificial Intelligence*, 525–530. AAAI Press.

Selman, B., and Kautz, H. 1996. Knowledge compilation and theory approximation. *J. of the ACM* 43:193–224.

Smith, D., and Weld, D. 1998. Conformant graphplan. *Proc. 15th Nat. Conf. on Artificial Intelligence*, 889–896. AAAI Press.