Backward Plan Construction for Planning with Partial Observability

Jussi Rintanen
Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Georges-Köhler-Allee, 79110 Freiburg im Breisgau
Germany

Abstract

We present algorithms for partially observable planning that iteratively compute belief states with an increasing distance to the goal states. The algorithms handle nondeterministic operators, but restrict to problem instances with a finite upper bound on execution length, that is plans without loops. We discuss an implementation of the algorithms which uses binary decision diagrams for representing belief states. It turns out that generation of new belief states from existing ones, corresponding to the use of conditional branches in the plans, can very naturally be represented as standard operations on binary decision diagrams. We also give a preliminary experimental evaluation of the algorithms.

Introduction

For solving planning problems in which the exact sequence of states encountered during plan execution cannot be predicted, for example because of nondeterminism, it is necessary to produce plans that apply different actions depending on how the plan execution has proceeded so far. Such plans are called conditional plans.

Construction of conditional plans is particularly tricky when there is no full observability; that is, when during plan execution it is not possible to uniquely determine what the current state of the world is. Planning problems having this property are said to be partially observable, and their solution requires that the sets of possible current world states – the belief states – are (implicitly) maintained during plan execution and (implicitly) represented by a plan.

In this paper we address the partially observable conditional planning problem. Partial observability is a notoriously difficult problem in AI planning and in policy construction for partially observable Markov decision processes (POMDPs) (Sondik 1978; Kaelbling *et al.* 1998). Recent works for planning not directly based on traditional techniques for solving POMDPs include (Weld *et al.* 1998; Rintanen 1999; Bonet and Geffner 2000; Bertoli *et al.* 2001). Bonet and Geffner handle numeric probabilities, like the POMDP model, the other works essentially handle three qualitative probabilities, $p = 1.0$, $p = 0.0$ and $1.0 > p > 0.0$. There is also some work on an easier special case, the unobservable (or conformant) planning problem, which does not allow observations at all (Smith and Weld 1998; Cimatti and Roveri 2000; Bonet and Geffner 2000).

We present two new algorithms for partially observable planning that use backward search in the belief space, producing belief states with an increasing distance to the goal states, each corresponding to a plan with which the goals can be reached from the belief state in question. The backward steps in the construction correspond to operator applications (the maximal possible predecessor belief state for reaching a given belief state with an operator) and branching (belief states obtained as combination of a number of belief states that can be observationally distinguished from each other.)

An effective implementation of the algorithms is obtained by using BDDs for representing the belief states. The size of a BDD often does not grow proportionally to the number of states in the belief state that is represented. Also, the backward steps corresponding to branching allow a very effective handling of large number of potential ways of branching as simple BDD operations, sometimes leading to big reductions in the amount of work needed.

In two special cases, planning with full observability and with no observability, the algorithms function like some existing specialized algorithms for the problems in question.

If the problem instance is fully observable, the algorithms essentially do breadth-first search backwards from the goal states with the whole search tree traversed up to a certain level represented as a single BDD, corresponding to one belief state. This backward traversal of the state space is like in BDD-based algorithms for model-checking in computer-aided verification and in earlier algorithms for fully observable planning that use BDDs.

If the problem instance is unobservable, the computational problem is to find a path from the initial belief state to the goal belief state, like in classical planning, but at the level of belief states instead of states. The symbolic breadth-first traversal of the state space used in the fully observable case is not possible, and the algorithms just produce belief states with an increasing distance to the goal states.

Both of these two forms of plan search are realized within the planning algorithms we propose. We generate belief states backwards starting from the goal belief state. The belief states can be combined (except in the unobservable special case) to obtain bigger belief states (the behavior used in the fully observable case without restrictions), and new belief states can be obtained by computing the possible predecessor belief states of a belief state with respect to an op-

erator application (which is the only behavior used in the unobservable special case.)

The structure of the paper is as follows. The next section describes the exact planning problem we are addressing, including the type of observations we handle and the form of plans. Then we describe the combination operator for belief states which is the basis of the backward traversal algorithms, and how how binary decision diagrams lead to a natural and efficient implementation of the combination operation. Two BDD-based algorithms based on the combination operation are described, the first uses exhaustive generation of belief states and the second uses a simple heuristic for selecting which belief states to produce. In the implementation section we describe the main insights obtained from implementing the algorithms, and make runtime comparisons to other planners. Finally, we conclude the paper by discussing connections to earlier work.

Problem Definition

Observation Model

During plan execution, before deciding which action to take next, it is often useful or necessary to try to determine what is the current state. Of course, given that the last action was taken in a certain belief state, it is possible to compute the set of possible successor states, but in addition, it may be possible to obtain new information about the world that allows distinguishing between the possible new current states.

The observation model we use is based on partitioning the state space S into non-empty pairwise-disjoint observational classes $P = \langle C_1, \ldots, C_n \rangle$ such that $S = \bigcup_{i \in \{1,\ldots,n\}} C_i$. Given the actual current state s (which we in general do not know), we observe C_i for $i \in \{1, \ldots, n\}$ such that $s \in C_i$, meaning that the actual current state is one of the states in C_i, but no further distinctions between the states in C_i can be made. If we already knew that the current state is one of the states in B, then we know that s is in $B \cap C_i$.

This can easily be generalized to the case in which the possible observations depend on the last action taken; that is, we have different partitions for different operators, allowing for example the expression of special observation actions.

We have problem representations with state variables and implementation techniques that use BDDs in mind, so it is most convenient to assume that partitions $C_1, \ldots, C_n$ are induced by sets O of state variables that are observable. Then, a component C_i of the partition is a set $C_i \subseteq S$ of states that assign the same values to all of the variables in O. Notice that when there are n observable Boolean state variables, there can be 2^n components in the partition.

There are several extensions of the observation model that can be implemented by reduction to the basic model we described above. For example compound observations can be reduced to atomic observations, assuming a sufficiently expressive language for describing operators. Within the BDD framework, handling observability of compound formulae ϕ directly is easy by having a state variable o_ϕ observable, and requiring that $o_\phi \leftrightarrow \phi$ holds in every state. This can be directly encoded to the BDD representation of transition relations of operators.

Plans

The plans our algorithms produce are directed acyclic graphs. Terminal nodes correspond to goal belief states. Nodes with exactly one successor node are labeled with an operator, and they correspond to operator applications. Nodes with more than one successor are branches: one of the successor nodes is chosen based on what is observed. The edges to the successor nodes are labeled with corresponding sets of observations.

The algorithms in this paper do not produce plans with loops. Loops are needed for finitely representing plans that have no finite upper bound on execution length. Execution length may be unbounded when some of the operators are nondeterministic. For example, the number of times a dice has to be thrown to get 6 is unbounded, and the plan needs a loop (throw the dice until you get 6.) Loops are not needed when all operators are deterministic. Many types of nondeterministic problems have solutions as plans without loops.

Distance Computation for Partially Observable Planning

Many of the earlier algorithms for fully observable planning are best understood as computing the distance/cost from every state to a goal state. These distances can be understood as a plan. The plan can be executed by always choosing an operator that reduces the distance of the current state by 1, or in nondeterministic planning, that makes the highest expected reduction in the distance/cost to a goal state.

In this paper we generalize this idea to the more complicated partially observable case. The passage from fully observable to partially observable planning requires employing the notion of belief states, that is sets of states, and computing (an upper bound on) the distances between belief states and the goal belief state. As in the fully observable case, the distance computation yields a plan as a byproduct.

The first complication is that the number of belief states is much higher than the number of states: for n state variables the maximum number of reachable states is 2^n, and this induces a belief space consisting of 2^{2^n} belief states.

The second complication is the definition of distances. We define the distance between a belief state and the goal belief state in terms of a plan implicitly associated with the belief state. The plan determines the distance as the maximum number of actions the plan may need for reaching a goal state. The computation of distances takes place in parallel with the computation of the belief states, and the plans need not be explicitly constructed.

In fully observable planning with deterministic actions, the distance of a state is simply one plus the minimum of the distances of the states that can be reached by one action. The same holds in partially observable planning for belief states, but there is also the additional possibility of determining the distance of a belief states in terms of the distances of other belief states. This is because plans can have branches. If from B we can reach belief states having distance d by a branch (observation) node in the plan, without taking any actions, then the distance of B is also d.

So for the distance computation we need to take into account the branch nodes, and this is a main difference between fully observable and partially observable planning. In fully observable planning, if we know that plans for reaching the goals exist for both B_1 and B_2, then we immediately know that there is also a plan for $B_1 \cup B_2$. In partially observable planning this is not the case: the existence of two plans, one reaching the goals from B_1 and the other from B_2 does not mean that there is a plan for $B_1 \cup B_2$. This is because in $B_1 \cup B_2$ we might not be able to choose the appropriate subplan corresponding to B_1 and B_2, as we might not know whether the state we are in belongs to B_1 or to B_2. In the next section we make this problem explicit, and explain how plan branching is handled in the distance computation.

For making the distance computation more feasible we will restrict to generation of set-inclusion maximal belief states. When we know that belief state B has distance n, we do not need to separately represent belief states $B' \subset B$ with distance n. This is because the same plan that reaches the goals from B will reach the goals also from B'.

Combination Operation $\otimes$ for Belief States

Belief states B_1 and B_2 are *observationally distinguishable* if every $s_1 \in B_1$ and $s_2 \in B_2$ assign a different truth-value to at least one observable state variable, and therefore belong to different components of the partition representing observability (for all C_i in the partition, either $C_i \cap B_1 = \emptyset$ or $C_i \cap B_2 = \emptyset$.) This means that after making the possible observations concerning the current state, one can exclude one of the sets of states B_1 and B_2 as impossible.

Later in Definition 2 we introduce a combination operator for belief states that is based on the following intuitions. If plan existence for belief states B_1 and B_2 has been shown (that is, there is plan Z_1 for B_1 and plan Z_2 for B_2, then for $B_1' \subseteq B_1$ and $B_2' \subseteq B_2$ such that B_1' and B_2' are observationally distinguishable, there must also be a plan for $B' = B_1' \cup B_2'$ that reaches the goal states. This means that in belief state B' we can choose one of Z_1 and Z_2 on the basis of values of the observable state variables, and have a guarantee that the goal states will be reached by the respective subplans. Hence a plan for B' starts with a branch node that selects one of the subplans Z_1 and Z_2.

Identifying sets B_1' and B_2' is easy in two special cases. If nothing is observable (the partition has only one component consisting of all states), the only possibilities are $B_1' = B_1, B_2' = \emptyset$ and $B_1' = \emptyset, B_2' = B_2$; that is, combining belief states does not produce new ones. If everything is observable (the partition consists of singleton sets), we choose $B_1' = B_1, B_2' = B_2$, which means that belief states can always be combined by taking their union.

In these two special cases the number of new belief states (1 or 2) is much smaller than in the general partially observable case, which will be discussed next. This agrees with the computational complexity results that say that the general planning problem with partial observability is much more difficult than either of these special cases, and that planning without observability is more difficult than planning with full observability (Mundhenk *et al.* 2000).

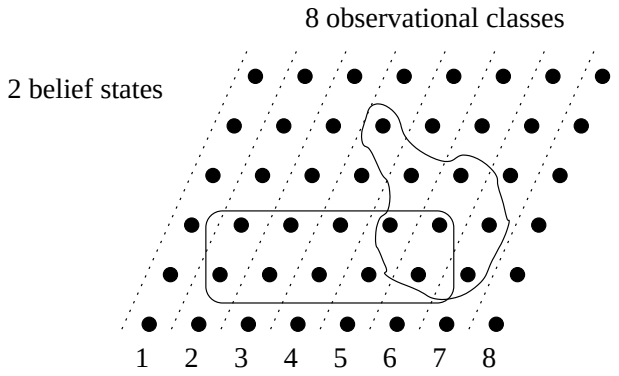

Figure 1: The intersections of two observational classes with the belief states are not in the inclusion relation. The belief states can be combined in four different ways.

Example 1 Consider the two belief states B_1 and B_2 in Figure 1. There are eight observational classes (components of the partition.) The problematic observational classes are those that intersect both B_1 and B_2 and the intersections are not in the inclusion relation (like in the unobservable case with only one observational class; in the fully observable case non-empty intersections coincide because they are singleton sets.) These are the classes C_4 and C_5. For all other classes C_i, $i \in \{1, 2, 3, 6, 7, 8\}$ either $B_1 \cap C_i \subseteq B_2 \cap C_i$ or $B_2 \cap C_i \subseteq B_1 \cap C_i$.

A belief state containing both $B_1 \cap C_j$ and $B_2 \cap C_j$ for $j \in \{4, 5\}$ is not (directly) backward reachable from B_1 and B_2, because for none of the states in $(B_1 \cap C_j) \cup (B_2 \cap C_j)$ can we say whether we are in B_1 or B_2, and hence the appropriate plan associated with B_1 or B_2 cannot be chosen.

Therefore, any backward reachable belief state B must fulfill either $B \cap C_j \subseteq B_1 \cap C_j$ or $B \cap C_j \subseteq B_2 \cap C_j$. Because we are interested in maximal belief states, this is either $B \cap C_j = B_1 \cap C_j$ or $B \cap C_j = B_2 \cap C_j$.

Hence we reach backwards the following belief states.

$$\begin{aligned} B_{11} &= B_s \cup (B_1 \cap C_4) \cup (B_1 \cap C_5) \\ B_{12} &= B_s \cup (B_1 \cap C_4) \cup (B_2 \cap C_5) \\ B_{21} &= B_s \cup (B_2 \cap C_4) \cup (B_1 \cap C_5) \\ B_{22} &= B_s \cup (B_2 \cap C_4) \cup (B_2 \cap C_5) \end{aligned}$$

Here $B_s = (B_1 \cup B_2) \cap (C_1 \cup C_2 \cup C_3 \cup C_6 \cup C_7 \cup C_8)$ is the union of the intersections of B_1 and B_2 with the unproblematic classes (intersections are in the inclusion relation). In these classes we can always safely choose the belief state with the bigger intersection.

Clearly, the number of combined belief states is 2^n when the number of non-inclusion observational classes is n. ■

Notice that the number of observational classes may be exponential on the number of state variables, which may make the number of combinations extremely high.

Now we are ready to give the definition of the combination operator $\otimes$ for belief states B_1 and B_2 which identifies maximal subsets of $B_1 \cup B_2$ from which either B_1 and B_2 can be chosen based on the observations that can be made.

Definition 2 *Let $P = \langle C_1, \ldots, C_n \rangle$ be a partition of the set of all states. Let B_1 and B_2 be two sets of states. Then $B_1 \otimes B_2$ is defined as*

$$\begin{aligned} B_1 \otimes B_2 \quad = \quad & \{I_1 \cup \cdots \cup I_n | \\ & I_i \in \{B_1 \cap C_i, B_2 \cap C_i\}, \\ & I_i \not\subset B_1 \cap C_i, I_i \not\subset B_2 \cap C_i, \\ & \text{for all } i \in \{1, \ldots, n\}\}. \end{aligned}$$

The combination operator has many algebraic properties that may be taken advantage of in improving algorithms. In this section we point out the most important ones. We consider the operator $\otimes$ with an arbitrary (but fixed) partition of the state space to observational classes.

Theorem 3 (Monotonicity) *Let B, B_1 and B_2 be belief states so that $B_1 \subseteq B_2$. For all $B_1' \in B \otimes B_1$, there is $B_2' \in B \otimes B_2$ such that $B_1' \subseteq B_2'$.*

Proof: Let B_1' be any member of $B \otimes B_1$. Hence $B_1' = \bigcup_{1 \leq i \leq n} I_i$ for I_i such that $I_i = B \cap C_i$ or $I_i = B_1 \cap C_i$.

Now we can construct B_2' as follows. The construction for every component in the partition (for every observational class) is independent of the other components. Choose any $i \in \{1, \ldots, n\}$. If $B_1' \cap C_i = B \cap C_i$ and $B \cap C_i \not\subset B_2 \cap C_i$, we choose $B \cap C_i$ for B_2'. Otherwise we choose $B_2 \cap C_i$ for B_2'. In both cases the intersection $B_1' \cap C_i$ is included in $B_2' \cap C_i$. Because this holds for every component C_i of the partition, we have $B_1' \subseteq B_2'$. □

Theorem 4 (Commutativity) *For all belief states B_1 and B_2, $B_1 \otimes B_2 = B_2 \otimes B_1$.*

Proof: Directly by the symmetry of the definition. □

We define for belief states B and sets S of belief states

$$S \otimes B = B \otimes S = \bigcup \{B \otimes B' | B' \in S\}.$$

Theorem 5 (Associativity) *For all belief states B_1, B_2 and B_3, $(B_1 \otimes B_2) \otimes B_3 = B_1 \otimes (B_2 \otimes B_3)$.*

Proof: We show that every belief state in $(B_1 \otimes B_2) \otimes B_3$ is also in $B_1 \otimes (B_2 \otimes B_3)$. Because of the commutativity of $\otimes$, showing that belief states in $B_1 \otimes (B_2 \otimes B_3) = (B_3 \otimes B_2) \otimes B_1$ are also in $(B_1 \otimes B_2) \otimes B_3 = B_3 \otimes (B_2 \otimes B_1)$ is by exactly the same argument.

Take any belief state B in $(B_1 \otimes B_2) \otimes B_3$. Take any component C_i of the partition (with $1 \leq i \leq n$.) Now $B \cap C_i$ equals either $B_3 \cap C_i$ or $(B_1 \otimes B_2) \cap C_i$. In the latter case $B \cap C_i$ equals either $B_1 \cap C_i$ or $B_2 \cap C_i$.

Because B was generated by cartesian products of intersections of B_1, B_2 and B_3 with the observational classes, the intersections of B with the observational classes can be considered independently one at a time.

If $B \cap C_i$ equals $B_1 \cap C_i$, then we have a belief state B' in $B_1 \otimes (B_2 \otimes B_3)$ with $B' \cap C_i = B \cap C_i$.

If $B \cap C_i$ equals $B_2 \cap C_i$ (the case $B_3 \cap C_i$ is the same because of commutativity), then there is a belief state B'' in $B_2 \otimes B_3$ with $B'' \cap C_i = B \cap C_i$, and further, a belief state B' in $B_1 \otimes (B_2 \otimes B_3)$ with $B' \cap C_i = B \cap C_i$. □

Because the operator is associative and commutative, it unambiguously generalizes to sets of belief states as

$$\otimes\{B_1, B_2, \ldots, B_n\} = B_1 \otimes B_2 \otimes \cdots \otimes B_n.$$

This can be more directly expressed as

$$\begin{aligned} \otimes\{B_1, \ldots, B_m\} \quad = \quad & \{I_1 \cup \cdots \cup I_n | \\ & I_i \in \{B_1 \cap C_i, \ldots, B_m \cap C_i\}, \\ & I_i \not\subset B_j \cap C_i \text{ for } j \in \{1, \ldots, m\} \\ & \text{for all } i \in \{1, \ldots, n\}\} \end{aligned}$$

for a partition $P = \langle C_1, \ldots, C_n \rangle$ of the state space.

We use the generalized definition of the $\otimes$ operation in implementing the planning algorithms. This is how we avoid the generation of a very high number of intermediate belief states that would otherwise be obtained by pairwise combinations of belief states.

Theorem 6 (Inclusion) *For all $B \in T$ there is $B' \in \otimes T$ such that $B \subseteq B'$.*

Proof: We construct $B' = \bigcup_{i=1}^{n} I_i$ by constructing I_i and showing that $B \cap C_i \subseteq I_i$ for every i.

Consider C_i. If $B \cap C_i \subset B^* \cap C_i$ for no $B^* \in T$, then we choose $I_i = B \cap C_i$, and clearly $B \cap C_i \subseteq I_i$. If $B \cap C_i \subset B^* \cap C_i$ for some $B^* \in T$, let B^* be such a belief state in T so that $B^* \cap C_i$ is set-inclusion maximal. Now we choose $I_i = B^* \cap C_i$, and clearly $B \cap C_i \subseteq I_i$.

Because the inclusion $B \cap C_i \subseteq I_i$ holds for all components of the partition, $B \subseteq \bigcup_{i=1}^{n} I_i = B'$. □

Implementation of $\otimes$ with BDDs

Instead of representing individual states and belief states as lists of and lists of lists of atomic propositions, for many types of planning it is more efficient to represents sets of states implicitly as formulae. A computationally effective representation of propositional formulae and Boolean functions is binary decision diagrams (BDDs) (Bryant 1992). BDDs allow equivalence testing in constant time and many operations on Boolean functions in polynomial time. They have been widely applied in model-checking in computer-aided verification, and in the last years also in AI planning, especially in planning under uncertainty.

In this section we show how any two belief states B_1 and B_2 represented as BDDs can be combined to $B_1 \otimes B_2$ by using standard operations on BDDs. Especially interesting is the use of the existential quantification operation of BDDs in handling partial observability. This BDD implementation of $\otimes$ is the basis of our planning algorithm implementation.

The implementation of $\otimes$ with BDDs is often much more efficient than a more direct implementation for two reasons. First, not all of the states in the belief states have to be represented separately. This is in general the main benefit of BDDs. Second, the $\otimes$ operation can be performed without iterating over every observational class (component of the partition), because BDDs very effectively allow handling those observational classes that intersect the belief states so that the intersections are in the set-inclusion relation.

The computation proceeds as follows.

1. Compute the observational classes that intersect both belief states so that the intersections are not in inclusion relation. This is by the following BDD computation.

$$X = \exists U(B_1 \wedge \neg B_2) \wedge \exists U(B_2 \wedge \neg B_1)$$

Here U is the set of unobservable state variables, and $\exists$ denotes the existential abstraction operation on BDDs that is defined for one variable x as

$$\exists x \Phi \stackrel{\text{def}}{\equiv} \Phi[0/x] \vee \Phi[1/x].$$

In the fully observable case X is the empty set, and in the unobservable case X is the universal set (if neither of the belief states subsumes the other.)

2. For those observational classes that intersect the belief states and the intersections are in the inclusion relation we can always choose the bigger intersection. The set of all those observational classes can directly be identified by simple BDD operations, and the explicit generation of these observational classes is avoided. The part of the state space that intersects both belief states so that the intersections are in the inclusion relation is simply

$$B = (B_1 \cup B_2) \backslash X.$$

3. For identifying the no-inclusion observational classes we iterate over the *cubes* of X, which are the disjuncts of a DNF of X. For every cube we assign the observable variables not occurring in the cube truth-values in all possible ways, in each case obtaining one observational class. The benefit of iterating over the cubes of X instead of using all the valuations of the observable variables is that X may include only a fraction of all observational classes. There are procedures for efficiently doing iteration over the cubes for example in the CUDD BDD package.
4. For every observational class C_i in X compute $B_1^i = B_1 \cap C_i$ and $B_2^i = B_2 \cap C_i$.
5. Produce the 2^n belief states as

$$B \cup B_{i_1}^1 \cup B_{i_2}^2 \cup \cdots B_{i_n}^n$$

where the i_j are assigned 1 or 2 in all 2^n possible ways.

Planning Algorithms

In this section we propose two algorithms for planning with partial observability that use the $\otimes$ operation. The first algorithm – given in Figure 2 – exhaustively computes sets D_i for $i \geq 0$ that contain all maximal belief states at distance i to the goal belief state.

In the algorithm description $\text{preimg}_o(B)$ computes the strong preimage of a set B of states with respect to an operator o (Cimatti *et al.* 1998). The strong preimage is the maximal set of states from which a state in B is always reached by applying o. For deterministic operators this coincides with the standard (weak) preimage computation used in model-checking and other applications of BDDs (Burch *et al.* 1994). Both preimage computations can easily be implemented with the standard operations on BDDs.

The variables I and G are respectively the initial and the goal belief states, and the set O consists of the operators in the problem. This algorithm description assumes a uniform observability at all points of time so that only one operator $\otimes$ that uses one partition of the state space to observational classes is needed. For the case in which the current observational classes depend on the operator last applied, we have to consider different functions $\otimes$, and to compute the preimage of a belief state only with respect to an operator that corresponds to the observability assumption with which the belief state was obtained by $\otimes$.

PROCEDURE exhaustive()
$i := 0$;
$D_0 = \{G\}$;
WHILE $I \subseteq B$ for no $B \in D_i$ and $D_i \neq D_{i-1}$
$D_{i+1} := \otimes(\{\text{preimg}_o(B) | B \in D_i, o \in O\} \cup D_i)$;
$i := i + 1$;
END;
IF $I \subseteq B$ for some $B \in D_i$ *THEN* plan has been found;

Figure 2: Algorithm that systematically generates the belief space

PROCEDURE heuristic()
$i := 0$;
$D_0 = \{G\}$;
$A = D_0$;
WHILE $I \subseteq B$ for no $B \in D_i$ and $A \neq \emptyset$;
$B :=$ an element of A of maximum cardinality;
$A := A \backslash \{B\}$;
$D_{i+1} := \otimes(\{\text{preimg}_o(B) | o \in O\} \cup D_i)$;
$A := A \cup (D_{i+1} \backslash D_i)$;
$i := i + 1$;
END;
IF $I \subseteq B$ for some $B \in D_i$ *THEN* plan has been found;

Figure 3: Algorithm that heuristically selects which belief states to expand

The second algorithm – given in Figure 3 – uses a heuristic for selecting one belief state at a time for preimage computation. Because the belief space is not traversed systematically, the correspondence between the sets D_i and the belief states at distance i is lost, and therefore distance information for the generated belief states has to be maintained separately. The heuristic tried with this second algorithm simply uses the cardinality of a belief state as the usefulness measure. On more complicated problems this is likely not to yield as good results as on the problems considered in the experiments described later in the paper.

After the initial belief state has been reached by using this backward computation, a branching plan can be extracted, as will be discussed in the next section.

We show the correctness of the exhaustive algorithm next. It is obvious that the heuristic algorithm is also correct, because all belief states are used at some point of time: the most promising ones are used early, and others later.

Theorem 7 *Whenever there exists a finite acyclic plan for a problem instance, the algorithm in Figure 2 returns it.*

Proof: So assume there is an acyclic plan for a problem instance. Let all nodes N of the plan be annotated with sets S_N of states possible during execution of the plan.

We show by induction on i that if the distance from node N to a terminal node of the plan is i (counting all edges on the path), then at the ith iteration of the algorithm a belief state B such that $S_N \subseteq B$ has been produced, i.e. $S_N \subseteq B$ for a belief state $B \in D_i$. Notice that the steps in constructing the sets D_i often correspond to two edges in the plan: one operator edge and one edge from a branch node.

Base case $i = 0$: The goal belief state is reached from a belief state by 0 steps only if the belief state is a subset of the goal belief state.

Inductive case $i \geq 1$: Now N is a node with distance i.

If it is an operator node, then a superset B' of the belief state B obtained from S_N by applying o (and associated with the successor node) is by the induction hypothesis in D_{i-1}. The preimage of B' with respect to o is one of the belief states the are combined to obtain D_i from D_{i-1}. Hence by Theorem 6 a superset of S_N is in D_i.

Otherwise N is a branch node. Then by the induction hypothesis supersets of the belief states corresponding to the successor nodes are in D_{i-1}. We show that a superset of the belief state for the node itself is in D_i. Let the belief states corresponding to the successor nodes be $B_1, \ldots, B_m$. Let the labels of edges to the successor state be $\phi_1, \ldots, \phi_m$. These can be understood equivalently as propositional formulae or as sets of states.

For the plan to be executable under the observability $P = \langle C_1, \ldots, C_n \rangle$, it must be the case that for no $k \in \{1, \ldots, n\}$ and $\{\phi, \phi'\} \subseteq \{\phi_1, \ldots, \phi_m\}$ such that $\phi \neq \phi'$, $\phi \cap C_k \neq \emptyset$ and $\phi' \cap C_k \neq \emptyset$. Otherwise during plan execution when making observation C_k it would not be possible to decide whether to follow the edge labeled ϕ or the edge labeled ϕ', as the observation would be compatible with both.

The belief states $B_1, \ldots, B_m$ associated with the successor nodes are subsets of the edge labels, so $B_k \subseteq \phi_k$ for all $k \in \{1, \ldots, m\}$. Hence no two of these belief states intersect the same observational class. Let $B'_1, \ldots, B'_m$ be the belief states in D_{i-1} so that $B_i \subseteq B'_i$ for $k \in \{1, \ldots, m\}$.

Finally we show that $S_N \cap C_k$ for every $k \in \{1, \ldots, n\}$ is included in the intersection of C_k with some belief state in D_{i-1}, and therefore S_N is a subset of a belief state in D_i. So take any $k \in \{1, \ldots, n\}$ such that $S_N \cap C_k$ is non-empty. Now $S_N \cap C_k = B_j \cap C_k$ for one $j \in \{1, \ldots, m\}$ by the considerations above, and further $S_N \cap C_k \subseteq B'_j \cap C_k$. Now either $B' \cap C_k = B''$ is set-inclusion maximal among intersections of belief states in D_{i-1}, or there is a belief state B'' in D_{i-1} with an even bigger intersection with C_k. In either case, $S_N \cap C_k \subseteq B'' \cap C_k$.

Because this holds for all $k \in \{1, \ldots, n\}$, one of the belief states in D_i includes S_N by the definition of $\otimes$. □

The algorithm is more efficient when keeping only set-inclusion maximal sets in D_i. That this resriction does not reduce the number of belief states that are computed is shown by Theorem 3. Under the maximality condition, a belief state B has distance at most i if there is a belief state B' such that $B \subseteq B$ and B' has distance at most i.

Extraction of Plans

The backward computation of belief states with increasing distances to the goal belief state is finished when a belief state that includes the initial belief state has been found. The assignment of distances to the belief states can be understood as a plan, but it may be useful to extract a graph-like plan that makes explicit the process of making observations and determining the next belief states of the plan execution, and identifies which belief states really are relevant parts of the plan. So, plan construction proceeds as follows.

1. Let the current belief state B be the initial belief state and d the distance computed for it.
2. If the distance of the current belief state is $d = 0$, then stop ($B \subseteq G$ for the goal belief state G.)
3. Identify operators $o_1, \ldots, o_n$ and a partition $B_1, \ldots, B_n$ of B ($n \geq 1$) so that the belief states in the partition are observationally distinguishable from each other, and from B_i a belief state with distance $\leq d - 1$ is reached with o_i.
4. Recursively construct a plan for the distance $d - 1$ belief states B_i by choosing $B = B_i$ and going to step 2.

Because the distance is reduced by at least one when going to a successor node, all execution paths in the plan are finite, ending in a terminal node with distance 0 and associated with a subset of the goal states, and hence the process terminates after a finite time.

Implementation of the Algorithms

We have implemented the algorithms in C. The planner takes its input in an extension of PDDL (Ghallab *et al.* 1998) that allows expressing observability restrictions and initial states as belief states. We used the CUDD BDD package by Fabio Somenzi of the University of Colorado.

During early experimentation with the idea of performing backward search in the belief space we obtained many insights that led to the implementation discussed in this paper. These insights were vital in achieving the level of performance our planner has.

1. The generalization of $\otimes$ to sets of belief states should be used instead of the binary version of $\otimes$.

   Application of $\otimes$ to all pairs of belief states produces most belief states several times and directly leads to very high runtimes. The only redundancy in a good implementation of the generalized $\otimes$ operator is that already existing belief states (or belief states subsuming them) are generated once more.

2. Sometimes only a fraction of the preimages of the belief states differ, because not all states in a compound belief state can be reached by any single operator.

   We have experimented with techniques that take this into account. First we compute the intersections of the observational classes with all belief states, but we do not blindly take their cartesian product. Instead, choosing which combinations of the intersections to use in producing new belief states is performed separately for every operator. Only such intersections from each observational class are chosen that have a non-empty (weak)

preimage with respect to the operator under consideration. Intersections having an empty preimage may be ignored. On some problems this leads to astronomic reductions in the number of belief states that are produced. For example in the partially observable 6 block blocks world the reduction at some application of the operation is from $217606647530633251447111 68 \sim 2.18 \cdot 10^{25}$ to 10000, from which only 180 belief states we did not have already.

Under the above improvement, we must preserve the correctness of the planning algorithms by explicitly testing whether the initial belief state can be obtained from the intersections[1]. This is so if for every observational class its intersection with the initial belief state is subsumed by one of the intersections of the observational class with an existing belief state. This test also makes it possible to avoid the last phase of combinations of belief states which would produce a belief state subsuming the initial belief state (among with many useless belief states.)

On many problems the number of belief states is still extremely high, and we believe that the application of the combination operator should be controlled far more, for example through heuristics that select which belief states to generate. Typically only a very small fraction of the combinations of the belief states are needed in solving any given problem instance. Currently, the heuristics just control which belief states are used in preimage computations, and even when only a small number of preimages are used, the number of new belief states may be very high.

Experimentation with the Implementation

This section describes results of experiments on a first implementation of our algorithms. These results do not constitute a proof of the general applicability of the algorithmic framework presented in the paper, but act as evidence that the approach is on some kinds of problems very competitive with other types of algorithms. Other planners used in the comparisons were MBP by Bertoli et al. (2001) and GPT by Bonet and Geffner (2000). Based on the runtime statistics in the respective papers, these two planners in general seem to be much faster than earlier planners for partially observable non-probabilistic planning problems.

Our research hypothesis was that when the goal states can be reached from "big" belief states, backward search with belief state combination is more efficient than algorithms that do forward search in the belief space, like the algorithm used in MBP. Big belief states in this context means that despite a high degree of uncertainty, sufficiently many observations can be made to choose the right actions to reach the goals. Forward search algorithms in this case face the problem of choosing between branching, which would reduce the uncertainty but simultaneously lead to very big plans, and going forward by performing an action, thereby potentially losing a useful branching point in favor of keeping the plan size down. Backward search algorithms, on the other hand, could effectively compute these big belief states by the combination operation.

[1]It could be that the initial belief state itself cannot be obtained by any single operator application from any set of states, and therefore the generation of a belief state subsuming the initial belief state would not take place.

symbol	observable state variables
uo	-
po	CLEAR(X), ONTABLE(X)
pfo	ON(X,Y)
fo	ON(X,Y), CLEAR(X), ONTABLE(X)

Table 1: Degrees of observability for the blocks world

It seems that our results do not confirm this hypothesis. For example, the emptyroom problem, which is discussed later, has goal reachability from big initial belief states, but our algorithm implementation is still very much slower than MBP. However, there is some evidence for supporting this hypothesis, obtained from a partially observable variant of the blocks world problem, but in general the question of the strengths of the algorithms remains very much open.

To carry out experiments and make a preliminary comparison between the planners, we identified benchmarks that could be used under different degrees of observability. The most famous benchmark problem in AI planning, the blocks world, serves this purpose well. This problem can be considered with different degrees of observability by having different state variables observable. The choices we used are listed in Table 1. The fully observable case is denoted by *fo* and the unobservable by *uo*. An interesting choice is to have only the ON relation observable (*pfo*). This is not fully observable in the sense that not all state variables are observable, but it is still possible to differentiate between any two states unambiguously. Another one is to have CLEAR(x) and ONTABLE(x) observable, which would best correspond to partially observable planning. Up to three blocks this makes it possible to distinguish between any two states, but for a higher number of blocks this is not so.

We formalized the actions so that problem instances with all of the states as the initial belief state and a stack with all blocks in it as the goal belief state is solvable. A sufficient condition for this is that the operator for moving a block onto the table is parameterized only with the block to be moved and works for irrespective of on which block the block in question is, so it suffices to know that the block is clear when moving it onto the table. For the unobservable case we formalized moving a block onto the table so that the operator has not precondition, and the move succeeds if the block was clear and on something, and otherwise nothing happens. To keep the initial state description small, we took as initial states all the states with stacks of height at most 4. On a higher number of blocks this was still not a sufficient restriction, and starting from 8 blocks neither our planner nor MBP could transform the initial state descriptions into BDDs within 1 GB of memory. Also on the 6 and 7 block problems the BDD size starts being a problem. A more compact encoding of the problems with a smaller number of state variables would speed up the planners considerably.

We generated problem instances with an increasing number of blocks, and tested them on the planners (we decided to christen our planner YKÄ). We ran GPT

problem	$\|S\|$	runtime in seconds			
		GPT	MBP	YKÄ$_e$	YKÄ$_h$
bw2fo	3	2.84	**0.06**	**0.02**	**0.05**
bw3fo	13	4.51	31.00	**0.11**	**0.09**
bw4fo	73	>1200	>1200	1.90	**0.56**
bw5fo	501	>1200	>1200	19.20	**6.22**
bw6fo	4051	>1200	>1200	>1200	**129.57**
bw7fo	37633	>1200	>1200	>1200	>1200
bw2pfo	3	2.81	**0.03**	**0.04**	**0.04**
bw3pfo	13	4.08	0.13	**0.10**	**0.07**
bw4pfo	73	>1200	89.19	2.87	**0.73**
bw5pfo	501	>1200	>1200	44.69	**10.75**
bw6pfo	4051	>1200	>1200	>1200	**365.55**
bw7pfo	37633	>1200	>1200	>1200	>1200
bw2po	3	2.66	**0.02**	**0.04**	**0.06**
bw3po	13	4.06	**0.08**	**0.10**	**0.10**
bw4po	73	>1200	**0.69**	1.67	**0.67**
bw5po	501	>1200	13.37	>1200	**7.11**
bw6po	4051	>1200	**389.05**	>1200	>1200
bw7po	37633	>1200	>1200	>1200	>1200
bw2uo	3	2.70	**0.02**	**0.02**	**0.03**
bw3uo	13	3.77	**0.08**	0.28	0.20
bw4uo	73	>1200	**3.42**	>1200	385.43
bw5uo	501	>1200	**9.86**	>1200	>1200
bw6uo	4051	>1200	**70.85**	>1200	>1200
bw7uo	37633	>1200	>1200	>1200	>1200

Table 2: Runtimes of a number of benchmark problems on a 360 MHz Sun Sparcstation. $|S|$ is the size of the state space. The runtimes that were within 30 per cent or 50 milliseconds from the best runtime are highlighted.

with the default setting except for the "set discretization-levels off" setting as adviced by Bonet. We ran MBP with the `-v 0 -explicit_dfs_forward -no_conformant_at_start` options that were used by Bertoli et al. (2001) on many of the runs reported there. For unobservable problems the specialized algorithm for unobservable planning in MBP was used because MBP refuses to apply the general algorithm to unobservable problems.

The runtimes we report are what GPT outputs for the parsing, compilation and solution phases (which would appear to be real time, not CPU time), what MBP outputs as "searching and printing time" and "preprocessing time", and for our planner the total CPU time, including reading and parsing the input file. GPT takes its input in its extension of the PDDL language, MBP in the AR language, and our planner in an extension of the PDDL language. We produced the input files automatically from common PDDL source files by a program that translates usual schematic PDDL descriptions into AR and into the two versions of PDDL, with all operators grounded (no schematic variables, *for all* and *exists* quantification unfolded.)

The runtimes are shown in Table 2. YKÄ$_e$ refers to the algorithm that exhaustively produces all belief states, and YKÄ$_h$ to the one with heuristic selection of belief states.

On the problems with full and almost full observability our algorithm fares much better than MBP. MBP is sensitive to the number of variables that are observable, even when there is no difference in terms of distinguishing between two states, as indicated by *fo* and *pfo* problems. On the *po* problems MBP fares better, and on the unobservable (conformant) problems the specialized conformant planning algorithm in MBP fares much better. We improved the problem representation for MBP by eliminating the unnecessary state variables ON(X,X). Without this improvement the MBP runtime on the 3 block fully observable problem was 1144 seconds instead of only 31.

Our problem representation is not favorable to GPT. GPT seems to compute the initial states by straightforward iteration over the valuations of the state variables, and chooses those that satisfy the initial state formula. For the 3 block problems this is still very feasible (12 state variables), but with 4 blocks it is not (20 state variables.) We believe GPT would solve at least the 4 and 5 block problems if the initial states were represented in a way that better observes the way GPT works. However, we did not verify this.

The number of belief states our algorithms produce for fully observable problems is small, and the efficiency of the algorithms does not much lag behind specialized algorithms for planning with full observability. The number of belief states that are explicitly produced is linear on the maximum length execution of the plan.

We also ran the planners on problems that earlier were used by Bertoli et al. (2001) and Bonet and Geffner (2000) to demonstrate the capabilities of MBP and GPT. The runtimes, which are rather bad on our planner, are shown in Table 3. The formalizations of these problems are not exactly like those used by Bertoli et al. and Bonet and Geffner, because we produced all the problem instances automatically from a formalization of the problems in a version of PDDL. Unlike the MBP and GPT input languages, our version of PDDL does not directly support multi-valued state variables.

GPT is on some problems better and on some problems worse. The GPT implementation differs much from the other two planners, as it translates the problem instances to C++ and compiles and runs them. This is the reason for the higher runtimes on the easiest problems.

The empty room problems represent rooms of varying sizes (n times n squares). The problem is to get from an arbitrary unknown location to the middle of the room by going north, south, east or west. One can only observe whether one is immediately next to one of the walls.

The ring problems have a number of rooms in which windows have to be closed and then locked. Initially the state of the windows is completely unknown.

The medical problems are about performing two medical tests that together unambiguously determine any of n possible illnesses, and then giving one of n medications to the patient. If wrong medication is used, the patient dies, otherwise she will recover. Again, our planner does not do so well. The exhaustive algorithm has difficulties already solving the problem instance with 5 illnesses. By starting from the goal states it identifies the 5 belief states from which the goals can be reached by taking the appropriate medication, and a couple of others where the illness has already been treated. But from here on the number of belief states

problem	$\|S\|$	runtime in seconds			
		GPT	MBP	YKÄ$_e$	YKÄ$_h$
medical02	20	2.51	**0.02**	0.11	0.09
medical03	32	2.76	**0.02**	8.99	0.15
medical04	36	2.82	**0.03**	36.46	3.33
medical05	48	3.41	**0.08**	>1200	4.19
medical06	52	3.44	**0.05**	>1200	328.38
medical07	64	6.61	**0.06**	>1200	244.03
medical08	68	20.41	**0.09**	>1200	>1200
BTS02	7	2.44	**0.03**	**0.02**	**0.05**
BTS03	10	2.58	**0.02**	**0.06**	**0.05**
BTS04	13	2.86	**0.02**	0.09	**0.06**
BTS05	16	3.40	**0.03**	0.24	**0.08**
BTS06	19	4.50	**0.06**	0.87	0.17
BTS07	22	7.27	**0.06**	3.99	0.32
BTS08	25	16.61	**0.07**	19.47	0.51
BTS09	28	49.17	**0.11**	95.73	0.95
BTS10	31	165.91	**0.13**	518.15	1.75
BTS12	37	>1200	**0.21**	>1200	3.87
BTS14	43	>1200	**0.35**	>1200	8.77
BTS16	49	>1200	**0.44**	>1200	19.67
BTCS02	47	3.91	**0.03**	0.09	0.11
BTCS03	132	3.08	**0.01**	2.92	1.00
BTCS04	341	4.44	**0.05**	128.32	13.06
BTCS05	838	>1200	**0.09**	>1200	167.56
BTCS06	1991	188.78	**0.18**	>1200	>1200
emptyroom05	25	21.55	**0.05**	>1200	2.61
emptyroom06	36	317.72	**0.09**	>1200	26.22
emptyroom07	49	>1200	**0.12**	>1200	31.68
emptyroom08	64	>1200	**0.15**	>1200	77.60
emptyroom10	100	>1200	**0.19**	>1200	395.27
emptyroom15	225	>1200	**0.36**	>1200	>1200
ring03	162	3.88	**0.11**	29.65	29.91
ring04	648	4.61	**0.40**	>1200	>1200

Table 3: Runtimes of a number of benchmark problems on a 360 MHz Sun Sparcstation. $|S|$ is the size of the state space.

explodes because each of the 5 belief states includes the possibility that the first, or the second, or both, or none of the test results are available. With 5 illnesses this means that the number of combined belief states is a couple of thousands ($2^4 \cdot 3^2 \cdot 4 \cdot 6 = 3456$, to be exact), and only one of them is relevant for producing a plan, the one with the uncured illnesses and results of both tests. When the preimages of this belief state with respect to the two test actions have been computed, we have the initial belief state. For 8 illnesses the number of belief states at distance 1 to the goal states is 58320, and the planner does not solve the problem in a reasonable amount of time. The heuristic version of the algorithm needs for the 8 illness problem about three hours.

BTS (Bertoli *et al.* 2001) and BTCS (Bonet and Geffner 2000) are partially observable versions of the notorious bomb in the toilet problem, in which the goal is to disarm a bomb, contained in one of a number of packages, by throwing the packages into a toilet. In these problems one can detect the bomb by a special sensing action, and take advantage of this to produce smaller plans.

Related Work

Algorithms for computing optimal POMDP policies are related to our algorithms, especially the POMDP value iteration algorithms (Sondik 1978; Smallwood and Sondik 1973; Kaelbling *et al.* 1998). These algorithms implicitly do backward construction of branching plans, and for each such plan compute its value for each state in the state space. For belief states the value of such a plan is obtained as a linear combination of the values for the individual states.

Our algorithms, on the other hand, ignore exact costs and probabilities, and just consider whether a plan is acceptable for a given state or a set of states, but are similarly based on an implicit backward construction of branching plans. In our algorithms each plan is represented by a set of states for which it is guaranteed to reach the goals. And each of the backup steps attempts to find bigger such sets, roughly corresponding to the backup steps in the POMDP value iteration algorithms trying to find increasingly better approximations of the value function. Our algorithms do not find optimal plans for example because of the restriction to set-inclusion maximal belief states.

Extensions of BDDs have recently been used for implementing POMDP policy construction algorithms for POMDPs represented in a factored form. For example, algebraic decision diagrams (ADDs) (Fujita *et al.* 1997; Bahar *et al.* 1997) were used by Feng and Hansen (2000).

Earlier work on partially observable planning with BDDs includes (Bertoli *et al.* 2001). Their algorithm uses BDDs for representing belief states. Plan search is by forward search starting from the initial belief state. The planner has two choices: either take a single action and reach another belief state, or make an observation (one observable state variable or several), which splits the belief state to several smaller ones, and continue plan construction recursively from each. When this algorithm splits a belief state under n observable Boolean state variables, it gets 2^n child nodes. The main difficulty in this approach is making informed decisions on when to split and when not to split.

Our algorithms include the BDD-based backward search algorithms for fully observable and unobservable planning by Cimatti and Bertoli et al. (1998; 2000) as special cases, concerning fully observable planning the strong planning algorithm, not the more powerful strong cyclic algorithm.

Closely related to the POMDP algorithms, but based on heuristic search rather than on the MDP value iteration and policy iteration algorithms, is the GPT system by Bonet and Geffner (2000). Plans are found by general-purpose search algorithms like real time dynamic programming. The system is applicable to a much wider range of problems than MBP and our planner.

Conclusions and Future Work

In this paper we introduced a combination operator for belief states and showed how it can be a basis of algorithms for planning with partial observability. The operator has not been considered before, most likely because algorithms explicitly generating belief states backwards starting from the goal belief state have not been introduced before. This

work acts as first evidence that there are possibilities in making this kind of plan construction feasible, even though the performance of the first implementation of this idea is often inferior to other planners addressing the same non-probabilistic planning problem.

Producing all combinations of even a small number of belief states may be very impractical because of their very high number, as indicated by the experimentation with our algorithms. The combination of a set of belief states has a very regular structure because it is essentially the cartesian product of the intersections of the individual belief states with the observational classes. We are currently experimenting with a new type of algorithm that, instead of explicitly producing reachable belief states, maintains combinations of belief states in the product form; that is, it explicitly represents only the intersections of the observational classes and belief states obtained as preimages of other belief states, and does not take the cartesian product which leads to the immediate explosion in the number of explicitly represented belief states. The main computational problem with this representation is to find an implicitly represented belief state (possibly intersecting several observational classes) from which a new belief state can be obtained as the preimage with respect to an operator. We consider this as one of the more interesting topics for future research.

Also, our algorithms should be generalized to plans with loops, which are needed when there is no finite upper bound on execution length. Instead of just computing strong preimages, we also have to use weak preimages, and handle the larger belief states that can be reached from them.

References

R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design: An International Journal*, 10(2/3):171–206, 1997.

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 473–478. Morgan Kaufmann Publishers, 2001.

B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61. AAAI Press, 2000.

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

J. R. Burch, E. M. Clarke, D. E. Long, K. L. MacMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, 1994.

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

A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) and the Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 875–881. AAAI Press, 1998.

M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. *Formal Methods in System Design: An International Journal*, 10(2/3):149–169, 1997.

M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the Planning Domain Definition Language, draft 1.1. unpublished, April 1998.

E. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 130–139, 2000.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720, 2000.

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

R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

D. E. Smith and D. S. Weld. Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) and the Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 889–896. AAAI Press, 1998.

E. J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: discounted costs. *Operations Research*, 26(2):282–304, 1978.

D. S. Weld, C. R. Anderson, and D. E. Smith. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) and the Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 897–904. AAAI Press, 1998.