Structured Reachability Analysis for Markov Decision Processes

Craig Boutilier, Ronen I. Brafman, and Christopher Geib
Department of Computer Science
University of British Columbia
Vancouver, BC, CANADA, V6T 1Z4
email: {cebly,brafman,geib}@cs.ubc.ca

Abstract

Recent research in decision theoretic planning has focussed on making the solution of Markov decision processes (MDPs) more feasible. We develop a set of algorithms for *structured reachability analysis* of MDPs that are suitable when an initial state (or set of states) is known. Using compact, structured representations of MDPs (e.g., Bayesian networks), our methods---which vary in the tradeoff between complexity and accuracy---produce structured descriptions of (estimated) reachable states that can be used to eliminate variables or variables values from the problem description, reducing the size of the MDP and making it easier to solve. Furthermore, the results of our methods can be used by existing (exact and approximate) abstraction algorithms for MDPs.

1 Introduction

While Markov decision processes (MDPs) have proven to be useful as conceptual and computational models for decision theoretic planning (DTP), there has been considerable effort devoted within the AI community to enhancing the computational power of these models. One of the key drawbacks of classic algorithms such as policy iteration [10] or value iteration [1] is the need to explicitly "sweep through" state space: since state spaces grow exponentially with the number of problem features, such methods are wildly impractical for realistic planning problems.

Recent research on the use of MDPs for DTP has focussed on methods for solving MDPs that avoid explicit state space enumeration while constructing optimal or approximately optimal policies. These techniques can be roughly categorized into two classes: those based on *aggregation* of states; and those based on *reachability analysis*.[1] In aggregation, certain states are clustered and treated as a single state. Recent automatic *abstraction* methods work by detecting the irrelevance of certain variables and eliminating those variables from consideration, effectively clustering states that differ on irrelevant (or approximately irrelevant) variables [3, 5, 4]. Reachability analysis exploits the fact that, given some initial state (or state set), certain states may not be reachable. The choices at such unreachable states have no bearing on the optimal policy choice or value at the start state or any reachable state. This idea can be extended to deal with approximate reachability [7].

We investigate the integration of reachability analysis with abstraction techniques. In particular, we develop techniques whereby knowledge of an initial state (or certain initial conditions) and the concomitant reachability considerations influence the abstractions produced for an MDP, forming what is termed by Knoblock [11] a *problem specific abstraction*. We assume that the MDP is described in terms of random variables using *dynamic Bayes nets* (DBNs) [5]; we also assume an initial state has been given. Our approach works as follows: the initial state is used to perform a rough reachability analysis, telling us which variable values (or which combinations of values) can arise with nonzero probability. The results are then used to *reduce* the DBN description of the MDP by deleting mention of certain variable values (or entire variables). The resulting simplified description represents an MDP with a reduced state space in which some, and perhaps all, unreachable states have been removed. This reduced MDP can now be solved using standard abstraction techniques [3, 5, 4].

Our approach gives rise to a family of algorithms in which the computational effort and accuracy of the reachability analysis is varied depending on the quality of the solution required and computational resources available. We describe two particular approaches: one is conceptually simple, very tractable, but can fail to detect certain unreachable states; the second is based on the GRAPHPLAN algorithm [2], and requires more computation, but will generally discover more states to be unreachable, thus resulting in smaller MDPs.

In Section 2, we review MDPs, Bayes net representations of MDPs, and briefly discuss techniques for policy construction that exploit the structure laid bare by this representation. In Section 3, we describe in detail two algorithms for structured reachability analysis and show how to produce reduced MDP descriptions. We conclude in Section 4 with additional discussion.

2 MDPs and Their Representation

2.1 Markov Decision Processes

We assume that the system to be controlled can be described as a fully-observable, discrete state *Markov decision process*

[1] A third approach is *decomposition* (see, e.g., [9]), which does not preclude the enumeration of all states, but saves computation through the reduced interaction of states.

[1, 10], with a finite set of system states S. The controlling agent has available a finite set of actions A which cause stochastic state transitions: we write $\Pr(s, a, t)$ to denote the probability action a causes a transition to state t when executed in state s. A real-valued reward function R reflects the objectives of the agent, with $R(s)$ denoting the (immediate) utility of being in state s. A (stationary) *policy* $\pi : S \rightarrow A$ denotes a particular course of action to be adopted by an agent, with $\pi(s)$ being the action to be executed whenever the agent finds itself in state s. We assume an infinite horizon (i.e., the agent will act indefinitely) and that the agent accumulates the rewards associated with the states it enters.

In order to compare policies, we adopt *expected total discounted reward* as our optimality criterion; future rewards are discounted by rate $0 \leq \beta < 1$. The value of a policy π can be shown to satisfy [10]:

$$V_\pi(s) = R(s) + \beta \sum_{t \in S} Pr(s, \pi(s), t) \cdot V_\pi(t)$$

The value of π at any initial state s can be computed by solving this system of linear equations. A policy π is *optimal* if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and policies π'. The *optimal value function* V^* is the same as the value function for any optimal policy. A number of good state-based techniques for constructing optimal policies exist including value iteration [1] and policy iteration [10].

2.2 Structured Representation & Computation

One of the key problems facing researchers regarding the use of MDPs for DTP is Bellman's "curse of dimensionality:" the number of states grows exponentially with the number of problem variables. Fortunately, several good representations for MDPs, suitable for DTP, have been proposed that alleviate the associated representational burdens. We adopt dynamic Bayes nets in this paper [8, 5].

We assume that a set of variables **V** describes our system. To represent actions and their transition probabilities, for each action we have a *dynamic Bayes net* (DBN) with one set of nodes representing the system state prior to the action (one node for each variable), another set representing the world after the action has been performed, and directed arcs representing causal influences between these sets. Each post-action node has an associated *conditional probability table* (CPT) quantifying the influence of the action on the corresponding variable, given the value of its influences (see [5, 6] for a more detailed discussion of this representation). Figure 1(a) illustrates this representation for a single action.

The lack of an arc from a pre-action variable X to a post-action variable Y in the network for action a reflects the independence of a's effect on Y from the prior value of X. We capture additional independence by assuming structured CPTs. In particular, we use a *decision tree* to represent the function that maps combinations of parent variable values to (conditional) probabilities. For instance, the tree in Figure 1(a) shows that W influences the probability of *HCU* becoming true only if L, *HCR* are true and *HCU* is false (left arrows denote "true" and right arrows "false").[2] A similar

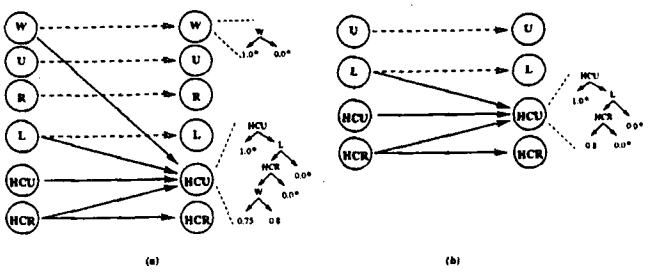

Figure 1: (a) Action Network; and (b) Reduced Network

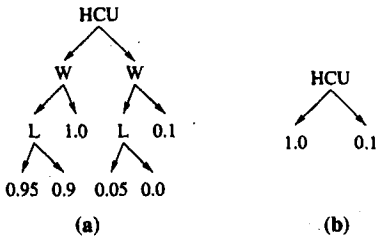

Figure 2: (a) Reward Tree; and (b) Reduced Tree

representation can be used to represent the reward function R, as shown in Figure 2(a).

The example in the figures above is taken from [3, 5], and describes a robot that runs across the street to a coffee shop to get coffee, but it gets wet when it is raining outside unless it has an umbrella. The action in Figure 1(a) describes delivering coffee: the user gets coffee (*HCU*) with reasonably high probability if the robot has coffee (HCR) and is in the right location (but wetness increases the chance of slippage). The reward function in Figure 2(a) gives a substantial reward for *HCU* and a slight penalty for W (and slightly larger penalty for dripping on the office floor).

Apart from the naturalness and conciseness of representation offered by DBNs and decision trees, these representations lay bare a number of regularities and independencies that can be exploited in optimal and approximate policy construction. Methods for optimal policy construction can use compact representations of policies and value functions in order to prevent enumeration of the state space.

In [5] a structured version of modified policy iteration is developed, in which value functions and policies are represented using decision trees and the DBN representation of the MDP is exploited to build these compact policies. Roughly, the DBN representation can be used to dynamically detect the relevance of various variables under certain conditions at any point in the computation, thus allowing states to be aggregated by ignoring irrelevant variables. For instance, in the example described above, states where W holds are never distinguished by the truth values of U or R, since once

[2] Certain persistence relations can be exploited in the specification of actions: the "starred" leaves correspond to persistence distributions where the variable retains its value after the action; we refer to [6] for a detailed discussion of persistence in DBNs.

W holds these facts are irrelevant to value or the optimal choice of action. The method can be extended to deal with approximation by using "degrees" of relevance as well [4].

A simpler abstraction technique developed in [3] does an *a priori* analysis of the problem (as opposed to a dynamic analysis) to delete irrelevant or marginally relevant variables from the problem description. For instance, the fine differences in reward associated with W and L in Figure 2(a) can be ignored by deleting these variables. The DBN description of actions allows one to easily detect that variables W, R, and U have no impact, through any "causal chain" (or sequence of actions), on the truth of *HCU*, which is the only remaining *immediately relevant* variables (i.e., determiner of reward). The abstract MDP uses only the three remaining variables and is much smaller (hence easier to solve)---the price is that the resulting policy will usually be nonoptimal.

3 Problem Specific Abstraction

One of the distinguishing features of MDPs *vis-a-vis* classical planning models is the need to construct policies that describe appropriate actions at all states. If we know the initial state (e.g., as in a planning problem), computational effort may be "wasted" in the determination of appropriate actions for states that can never be realized given the initial state, even if abstraction methods are used.

However, it is often the case that we can rather easily determine that certain variable values or combinations cannot be made true given the initial state. Any such knowledge of reachability can be exploited to reduce the range of dynamic programming. Furthermore, if this reachability analysis can be performed in a structured manner, the results can be combined with the policy construction methods and abstraction techniques described above. For instance, in our simple example, if the initial state satisfies $\overline{R}$, then values of variables R and W cannot be changed (we assume $\overline{W}$ is also known of the initial state). As a result, one can legitimately remove all mention of these variables from the MDP description, resulting in a *reduced MDP*. The reward function description for this reduced MDP is shown in Figure 2(b). A reduced action description (for delivering coffee) is shown in Figure 1(b). Intuitively, this reward and action refer to the MDP whose states vary over the four listed variables, but whose values of R and W are fixed to be false. We now describe two algorithms that reflect these considerations.

3.1 Reachability Analysis without Interactions

We begin with a simple and efficient algorithm, REACHABLE1, for determining the set of reachable variable values given an initial state description and a set of DBNs characterizing the MDP dynamics. The initial state is simply an assignment of specific values to variables.[3] The output is a set, for each variable, of the values of that variable that (we treat as if they) can be made true with positive probability given that the process begins in the initial state. Roughly, the *reachable set* Rch_i for variable V^i is instantiated with

[3]The extension to a *set of possible initial states* is obvious.

Input: Initial State $V^i = v^i_j$ $(1 \leq i \leq n)$
Output: Reachable Value List Rch_i for each variable V^i

Let $Rch_i = \{v^i_j\}$ for each $i \leq n$ (initial state)
Loop until no change in any Rch_i
 For each V^i $(i = 1$ to $n)$
  For each value $v^i_k \notin Rch_i$ $(k = 1$ to $m_i)$
   If there is action a, asst. A to parents of V^i in DBN(a) s.t.:
   $\Pr(V^i = v^i_k | a, A) > 0$; and each element $v^j_l \in A$ is in Rch_j; then
    Add v^i_k to Rch_i

Figure 3: Algorithm REACHABLE1

V^i's initial value. We then sweep through the actions a and variable-value pairs $V^i = v^i_j$ one at a time to determine if there is an assignment A of values to the parents of V^i in the DBN for a such that $\Pr(v^i_j) > 0$ and A is consistent with the set of current reachable value list; if so v^i_j is added to the reachable list for V^i. The algorithm is described in Figure 3. In our running example, with initial state satisfying $\overline{R}$, variables R and W will be discovered to be *removable*: their values cannot be changed from those in the initial state.

It is easy to show the algorithm will converge to a fixed point reasonably quickly:

Proposition 1 *Algorithm* REACHABLE1 *runs in* $O(n^2d^2a)$ *time, where* n *is the number of problem variables,* d *is the maximum domain size for any variable, and* a *is the number of actions.*

We note that this analysis holds only for the very naive version of REACHABLE1 shown in Figure 3. Clever indexing schemes, or even straightforward use of persistence relationships, will make this algorithm much faster in practice. In fact, we can easily construct an algorithm that runs in $O(ndb)$ time, where b is the size of the nontrivial (nonpersistent) portion of the DBN action descriptions.

Two important properties of reachability algorithms are *completeness* and *soundness*. An algorithm is *sound* if every state considered unreachable by the algorithm is, in fact, unreachable (or equivalently, all reachable states are said to be reachable by the algorithm). This is important for accurate solution of an MDP: if all reachable states are included in the reduced MDP, the optimal policy for the reduced MDP will be an accurate reflection of optimal behavior with respect to the given initial state. More specifically, let $\widehat{M}$ denote a reduced version of MDP M obtained by removing all states deemed unreachable by a reachability algorithm A. Let $\widehat{S} \subseteq S$ be the state space for $\widehat{M}$, let $\widehat{\pi}*$ be an optimal policy for $\widehat{M}$, and let V^* denote the optimal value function for M. If A is sound, we are assured of the following:

Theorem 2 *Let* π *be any policy for* M *that extends* $\widehat{\pi}*$ *(i.e.,* $\pi(s) = \widehat{\pi} * (s)$ *for any* $s \in \widehat{S}$*). Then, for any* $s \in \widehat{S}$*, we have:* $V_{\widehat{\pi*}}(s) = V_\pi(s) = V^*(s)$

An algorithm is *complete* if all unreachable states are said to be unreachable by the algorithm, i.e., all unreachable states are recognized. Completeness ensures that no unreachable states are included in the reduced MDP, and it has the effect

of keeping the reduced MDP small, though, on its own, does not guarantee an optimal solution.[4]

The output of REACHABLE1 is interpreted as follows: any state consisting of variable values that are in the reachable value lists is accepted as a reachable state. On this interpretation, REACHABLE1 is sound.

Theorem 3 *If state* $t = \langle v^1_{i_1}, v^2_{i_2} \cdots v^n_{i_n} \rangle$ *is reachable from initial state* s*, then each value* $v^k_{i_k}$ *that makes up state* t *is on the reachable list for* V^k *returned by algorithm* REACHABLE1.

Thus, the reduced MDP is suitable for the planning problem posed. However, REACHABLE1 is not complete, so the reduced MDP can be larger than necessary. We discuss this further in the next section.

Once the set of reachable states is produced, we have implicitly determined an abstract MDP whose states consist of those assignments returned by REACHABLE1. To exploit this fact in the algorithms that use DBN representations of MDPs, we would like to reduce the DBNs (and reward tree) of the MDP. This process is reasonably straightforward. Intuitively, we remove any unreachable variables values from the reward tree or CPT-trees, as shown in Figures 2(b) and 1(b), by removing any edges in the tree labeled with unreachable values. If this results in a node with only one outgoing edge, the variable itself is removable (i.e., has only one reachable value), and is deleted from the tree (the subtree attached to the remaining edge is promoted). Any removable variables can be completely deleted from the DBNs as well. The reduction of this MDP representation can be performed in linear time, and results in reward tree and set of DBNs that accurately reflects the reduced MDP. By retaining the structured nature of the representation, the result can be used by any standard abstraction algorithm (Section 3.3). In our coffee-serving robot example, REACHABLE1 will discover that the values of the variables W and R are uncontrollable, leading to a four-fold reduction in the size of the MDP. In addition, using a single backwards sweep, as performed by any basic abstraction technique, the variable U will be found to be irrelevant. This leads to an overall eight-fold reduction in the MDP size, from 64 to 8 states.

3.2 Reachability Analysis with Interactions

As noted above, REACHABLE1 is not complete: some unreachable states may not be recognizes as such, and may be deemed reachable. This is due to the fact that it does not take into account *interactions* among action effects or "exclusion relations" among the conditions that lead to specific effects. For instance, if a light can only be on if the switch is on, REACHABLE1 will not detect the correlation and judge a state where, say, *switch-on* and *light-off* hold to be reachable. The advantage of ignoring such interactions is the computational efficiency gained. We now describe a more sophisticated (and computationally expensive) algorithm that extends REACHABLE1 by accounting for certain interactions

[4]Notice that the terms *sound* and *complete* are w.r.t. statements of *un*reachability, which is really what we are interested in.

Input: Initial State $V^* = v^*_i$ $(1 \leq i \leq n)$
Output: Reachable Value List *Values* together with exclusion relationships *Excl* on pairs of values for distinct variables.

Let *Values* $= \{v^i_j : i \leq n\}$ (initial state)
Let *Excl* $= \emptyset$
Loop until no change in *Values*
  Set *ActNodes*, *ExclActs* $= \emptyset$
  For each action a, variable V and nontrivial C s.t.
  $C \subseteq$ *Values*; and no pair $\langle x, y \rangle$ of values in C is in *Excl*
    Add ca-node $\langle a, C, V \rangle$ to *ActNodes*.
  For each $v \in$ *Values*
    Add ca-node $\langle Noop(v), v, V \rangle$ to *ActNodes*.
  For each $\langle a_1, C_1, V_1 \rangle, \langle a_2, C_2, V_2 \rangle \in$ *ActNodes*
    If a) there is some $v_1 \in C_1, v_2 \in C_2$ such that $\langle v_1, v_2 \rangle \in$ *Excl*; or
    if b) an effect of $a_1; C_1$ conflicts with C_2 or an effect of $a_2; C_2$:
      Add $\langle \langle a_1, C_1 \rangle, \langle a_2, C_2 \rangle \rangle$ to *ExclActs*
  Set *Values*, *Excl* $= \emptyset$
  For each $\langle a, C, V \rangle \in$ *ActNodes*
    Add effects v_i to *Values*; record $\langle a, C, V \rangle$ as "a way to achieve" v_i
  For each $v^k_i, v^l_j \in$ *Values* (where $V^k \neq V^l$)
    Add $\langle v^k_i, v^l_j \rangle$ to *Excl* if all ways of achieving v^k_i
    are marked as exclusive of all ways of achieving v^l_j

Figure 4: Algorithm REACHABLE2

among values using *exclusion constraints*. The algorithm is inspired largely by the graph building phase of GRAPHPLAN [2] for STRIPS planning, but deals with the more complex notion of nondeterministic reachability. In particular, it extends GRAPHPLAN by dealing with both conditional action effects and, more substantially, with nondeterministic effects.

The Bayes net representation of actions described in Section 2.2 assumes that the effect an action has on the value of each proposition is independent. Hence, an action can be thought of as having multiple *aspects*, one with respect to each proposition. More generally, an action's aspects can encompass multiple propositions (e.g., see [3]). In the following discussion, we assume that each action definition contains only a single aspect. That is, for each condition, we specify the joint effect of this action on all propositions. This does not restrict the generality of the algorithm presented, as it is straightforward to convert the Bayes net action representation to the above representation. However, this is likely to result in a significant increase in the number of actions and therefore the size of the search space. Our algorithm can be extended to directly handle the more compact Bayes net representation; this will be discussed in a full version of this paper.

We begin with some preliminary definitions. Let B be the network for an action a, let V be some variable, and let $CPT(a, V)$ denote the tree quantifying a's effect on V. A *nontrivial condition* for action a with respect to variable V is any branch of $CPT(a, V)$ whose leaf is not labeled with a persistence distribution for V. The *conditional effect* of action a on variable V under condition C is the set of variable values $\{v_i\}$ such that $\Pr(V = v_i | C, a) > 0$ (i.e., the set of values that V *might* take if a is executed when C holds).

The algorithm REACHABLE2 is sketched in Figure 4. As with GRAPHPLAN, we construct a "graph" whose nodes

are arranged in levels, alternating *propositional levels* with *action levels*. A propositional level contains a number of *value nodes* (v-nodes) labeled with variable values, and a set of *exclusion constraints* --- each such constraint is a pair of variable values (for distinct variables) labeling v-nodes at that level. Intuitively, each value labeling a v-node is reachable, but any pair of values marked as exclusive cannot occur together in any reachable state (at least at that stage of the process).

An action level contains *conditional action nodes* (ca-nodes), each labeled with a tuple $\langle a, C, V\rangle$, where a is some action and C is a nontrivial condition for a with respect to variable V. Intuitively, we think of each condition-action pair as a distinct action that has "precondition" C. Action levels also contain exclusion constraints. Intuitively, two (conditional) actions are marked as exclusive if one "interferes" with the execution of the other. For instance, if they have conflicting effects or if one destroys the preconditions of the other, two actions will be marked as exclusive. Intuitively, any set of actions at a particular action level that are not marked by any exclusions can all be performed in sequence (at that stage of the process), in *any* order, and the resulting state will be one in which the effects of each of these actions holds.

The "graph" is constructed by REACHABLE2 as follows. Level 0 is a propositional level consisting of a v-node for each variable value comprising the initial state. At this initial level, there are no exclusion constraints. For a given propositional level, we create a subsequent action level: for each action a, variable V and nontrivial condition C in $CPT(a, V)$, we add a ca-node labeled $\langle a, C, V\rangle$, as long as: value $X = x_i$ occurring in C exists at the previous propositional level; and no pair of values $X = x_i$ and $Y = y_j$ occurring in C is marked as exclusive at the previous propositional level. In addition to these nontrivial ca-nodes, we add, for each value v_i occurring at the previous propositional level, a ca-node labeled $\langle No\text{-}op(v_i), v_i, V\rangle$ to the action level. This corresponds to a *No-op* action with respect to v_i, whose precondition is v_i and whose only effect is to ensure v_i persists in value to the next stage of the process.

The most crucial distinction between REACHABLE2 and GRAPHPLAN is the way in which actions are marked as exclusive, reflecting the nondeterminism of effects. In order to guarantee the soundness of our algorithm, we must ensure that actions that possibly do not interfere with one another (under certain conditions) will not be marked as exclusive; only actions that *necessarily* conflict will be marked exclusive. More formally, nodes $\langle a_1, C_1, V_1\rangle$ and $\langle a_2, C_2, V_2\rangle$ are marked as exclusive if:

(a) Their conditions C_1 and C_2 are marked as exclusive at the previous propositional level (some value in C_1 is exclusive of one of C_2's values)

(b) Some effect of a_1 given C_1 *conflicts* with C_2, or with the effects of C_2

The first condition is the same as in GRAPHPLAN. The second is interpreted as follows: if a_1 has effect $S = \{v_1^3, v_2^3, \cdots\}$ on some variable V^3 when C_1 holds, and C_2 requires that $V^3 \notin S$, or the effect of a_2 (under C_2) on V^3 does not intersect S, then the ca-nodes are marked as exclusive. Intuitively, if there is some ca-node $\langle a_2, C_3, V_3\rangle$ such that $C_2 \not\models C_3$ and the effect (set of possible V^3 values) of a_2 under C_3 has no values in common with S then $\langle a_1, C_1\rangle$ "clobbers" $\langle a_2, C_2\rangle$ and they are marked as exclusive (similarly, if the effect of a_1 clobbers C_2 itself).[5]

Finally, to complete the specification of graph construction, given some action level, we create a subsequent propositional level as follows: for each ca-node $\langle a, C, V\rangle$, we add a v-node for each value v_i such that $\Pr(V = v_i | a, C) > 0$; and for any pair of v-nodes v_i, x_j so added, we add the exclusion constraint $\langle v_i, x_j\rangle$ if *every way* of achieving v_i at the previous action level is marked as exclusive of *every way* of achieving x_j.

There are several more superficial differences between REACHABLE2 and GRAPHPLAN that we elaborate on in a longer version of the paper. These include several differences due to the use of DBNs rather than STRIPS rules: the distributed nature of action effects requires some subtlety in conflict detection, and the conditional nature of actions also necessitates certain differences with GRAPHPLAN. There are also differences due to the infinite horizon nature of our problem. In particular, we do not keep track of a complete "graph" but only the leading frontier of action and propositional levels. We are simply trying to determine those variables values that are reachable with positive probability at any stage of the process.

The use of suitable data structures and other shortcuts will make this algorithm quite efficient. For instance, the data structures used in [2] can be exploited by REACHABLE2, and testing for exclusion among all pairs of action nodes at a given level need not require exhaustive comparisons for all pairs---a single test can suffice for many different pairs. As it stands the algorithm converges reasonably quickly. We consider the set of reachable states to be those formed from assignments using values in the final propositional level that respect exclusions. The reachable states determined by the sequence of propositional levels increases monotonically, and the algorithm must terminate within $O(n^2d^2)$ iterations, with a running time polynomial in n, d and a.

We note that REACHABLE2 is sound. It is also "more complete" than REACHABLE1:

Theorem 4 *If state $t = \langle v_{i_1}^1, v_{i_2}^2 \cdots v_{i_n}^n\rangle$ is reachable from initial state s, then each value $v_{i_k}^k$ that makes up state t is on the reachable list for V^k returned by algorithm* REACHABLE2. *Furthermore, no two values $v_{i_j}^j$ and $v_{i_k}^k$ are marked as exclusive.*

Proposition 5 *If S_1 is the set of reachable states returned by* REACHABLE1 *and S_2 is the set of reachable states returned by* REACHABLE2 *(for a fixed initial state s), then $S_2 \subseteq S_1$.*

The reduced MDP is constructed using the output of REACHABLE2 in much the same fashion as REACHABLE1. The key distinction lies in the use of the exclusion constraints,

[5] We note that testing for conflicting effects can be done with various degrees of complexity, as we describe in a longer version.

so that tree edges can be deleted if the labeling value conflicts with values earlier in the branch. The reduction is not local but can still be implemented in one pass through the tree. These constraints may be used also directly in algorithms like those in [5, 4] (see next section).

3.3 Abstraction and Reachability Combined

The most important aspect of our algorithms for MDP reduction through reachability analysis is the fact that they produce compact DBN representations of the reduced MDP. As a result, one can directly apply structured abstraction techniques to the reduced MDP in order to solve it. The advantage of first producing a reduced MDP is that the descriptions are generally smaller (and can certainly be no larger). This generally results in fewer distinctions being made by the structured algorithm used. If one needs only solve a planning problem (for a given initial state or set of initial states) rather than the complete policy construction problem, this can provide considerable advantages.

For example, in the 64-state example described above, the algorithm of [5] produces a tree-structured policy and value function with 8 and 18 leaves respectively. When augmented by REACHABLE1, the same algorithm, for 48 of the 64 possible initial states, produces trees (for both the policy and value function) that have *at most* 5 leaves, exploiting the irrelevance of certain propositions given knowledge of specific initial states. For instance, if R is known to be false, REACHABLE1 fixes the value of R and W and this policy algorithm will never use the variable U (detecting its complete irrelevance). We note that algorithms like these must be augmented slightly to *fully* exploit the output of REACHABLE2: since these algorithms put together combinations of variables dynamically, they may start to compute policy values for states satisfying variable values that are marked as exclusive. It is simple to add tests that use the constraints returned by REACHABLE2 to rule out unrealizable variable value combinations when building policy and value trees.

The simpler abstraction algorithm of [3] also benefits. When variables W, R and U are deleted from the problem description, an approximate policy results with a given error (bounded by the algorithm). However, when REACHABLE1 is used, the same variables are deleted and no loss of value is accrued (and this fact is known) given specific starting states. The effect, in general, is to allow more aggressive pruning of variables within acceptable error tolerances.

4 Concluding Remarks

We have described techniques for performing a structured reachability analysis using compact, Bayesian network MDP representations, the result of which is a reduced MDP description in which only reachable values or value combinations are specified. This analysis can be exploited by abstraction methods since the structured nature of the representation is retained, and provides the advantage of reducing the number of distinctions required in different abstraction methods. These algorithms bring together two distinct approaches to addressing the computational difficulties associated with solving MDPs and illustrate the synergistic relation between the two. One cannot expect reachability to play a substantial role in reducing the size of MDPs in all cases; but there are many circumstances in which reachability analysis will be significant, for example, where rewards are conditional, or a number of variables (observables) are uncontrollable (like the weather, road conditions, interest rates).

In a longer version of the paper, we describe how other algorithms fit into this framework. For instance, REACHABLE2 explores only *pairs* of conflicting values, though more sophisticated interactions could be dealt with at greater computational expense, ranging up to full-fledged (sound and complete) reachability analysis. We also discuss ideas pertaining to unsound reachability algorithms, leading to MDPs whose solution may not be optimal, but which may be solved more quickly by the elimination of reachable states (e.g., states with low probability of being reached).

We note that these ideas can be applied to other representations (e.g., STRIPS) and planning algorithms. For instance, our results suggest how GRAPHPLAN can be extended to deal with nondeterministic or probabilistic planning, or how it might deal with more than pairwise constraints on actions and propositions. This is the topic of future investigations.

References

[1] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[2] Avrim L. Blum and Merrick L. Furst. Fast planning through graph analysis. In *Proc. 14th IJCAI*, 1995.

[3] Craig Boutilier and Richard Dearden. Using abstractions for decision-theoretic planning with time constraints. In *Proc. 12th Nat. Con. on AI*, 1994.

[4] Craig Boutilier and Richard Dearden. Approximating value trees in structured dynamic programming. In *Proc. 13th Int. Conf. .on Machine Learning*, 1996.

[5] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proc. 14th IJCAI*, 1995.

[6] Craig Boutilier and Moises Goldszmidt. The frame problem and Bayesian network action representations. In *11th Biennial Canadian Conf. on AI*, 1996.

[7] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In *Proc.11th Nat. Conf. on AI*, 1993.

[8] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142--150, 1989.

[9] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *Proc. 14th IJCAI*, 1995.

[10] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[11] Craig A. Knoblock. Automatically generating abstractions for planning. *Art. Int.*, 68:243--302, 1994.