Stochastic Planning with First Order Decision Diagrams

Saket Joshi and **Roni Khardon**
{sjoshi01,roni}@cs.tufts.edu
Department of Computer Science, Tufts University
161 College Avenue
Medford, MA, 02155, USA

Abstract

Dynamic programming algorithms have been successfully applied to propositional stochastic planning problems by using compact representations, in particular algebraic decision diagrams, to capture domain dynamics and value functions. Work on symbolic dynamic programming lifted these ideas to first order logic using several representation schemes. Recent work introduced a first order variant of decision diagrams (FODD) and developed a value iteration algorithm for this representation. This paper develops several improvements to the FODD algorithm that make the approach practical. These include, new reduction operators that decrease the size of the representation, several speedup techniques, and techniques for value approximation. Incorporating these, the paper presents a planning system, FODD-PLANNER, for solving relational stochastic planning problems. The system is evaluated on several domains, including problems from the recent international planning competition, and shows competitive performance with top ranking systems. This is the first demonstration of feasibility of this approach and it shows that abstraction through compact representation is a promising approach to stochastic planning.

Introduction

Relational MDPs have been recently investigated as a formalism and tool to solve stochastic relational planning problems (Boutilier, Reiter, and Price 2001; Kersting, Otterlo, and Raedt 2004; **?**; Gretton and Thiebaux 2004; Sanner and Boutilier 2006; Wang, Joshi, and Khardon 2008) and several of these use dynamic programming MDP algorithms to derive solutions. The advantage of the relational representation is abstraction. One can plan at the abstract level without grounding the domain, potentially leading to more efficient algorithms. In addition, the solution at the abstract level is optimal for every instantiation of the domain. However, this approach raises some difficult computational issues because one must use theorem proving to reason at the abstract level, and because optimal solutions at the abstract level can be infinite in size. Following Boutilier, Reiter, and Price (2001) several abstract versions of the value iteration (VI) algorithm have been developed using different representation schemes. For example, approximate solutions based on linear function approximations have been developed and successfully applied in several problems (Sanner 2008; Sanner and Boutilier 2006).

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

An alternative representation is motivated by the success of algebraic decision diagrams in solving propositional MDPs (Hoey et al. 1999; St-Aubin, Hoey, and Boutilier 2000). Following this work, relational variants of decision diagrams have been defined and used for VI algorithms (Wang, Joshi, and Khardon 2008; Sanner 2008). However Sanner (2008) reports on an implementation that does not scale well to large problems. Our previous work (Wang, Joshi, and Khardon 2008) introduced First-order decision diagrams (FODD), and developed algorithms and reduction operators for them. However, the FODD representation requires non-trivial operations for reductions (to maintain small diagrams and efficiency) leading to difficulties with implementation and scaling.

This paper develops several improvements to the FODD algorithms that make the approach practical. These include new reduction operators that decrease the size of the FODD representation, several speedup techniques, and techniques for value approximation. Incorporating these, the paper presents FODD-PLANNER, a planning system for solving relational stochastic planning problems. The system is evaluated on several domains, including problems from the recent international planning competition, and shows competitive performance with top ranking systems. To our knowledge this is the first application of a pure relational VI algorithm without linear function approximation to problems of this scale. Our results demonstrate that abstraction through compact representation is a promising approach to stochastic planning.

Preliminaries

Relational Markov Decision Processes

A Markov decision process (MDP) is a mathematical model of the interaction between an agent and its environment (Puterman 1994). Formally a MDP is a 4-tuple $< S, A, T, R >$ defining a set of states S, set of actions A, a transition function T defining the probability $P(s_j \mid s_i, a)$ of getting to state s_j from state s_i on taking action a, and an immediate reward function $R(s)$. The objective of solving a MDP is to generate a policy that maximizes the agent's total, expected, discounted, reward. Intuitively, the expected utility or value of a state is equal to the reward obtained in the state plus the discounted value of the state reached by the best action in the state. This is captured by the Bellman equation as $V(s) = Max_a[R(s) + \gamma\Sigma_{s'}P(s'|s,a)V(s')]$. The value iteration algorithm is a dynamic programming algorithm that treats the Bellman equation as an update rule and iteratively updates the value of every state until convergence. Once the optimal value function is known, a policy can be generated by assigning to each state the action that maximizes expected value. Hoey et al. (1999) showed that if $R(s)$, $P(s' \mid s,a)$ and $V(s)$ can be represented using algebraic decision diagrams (ADDs), then value iteration can be performed entirely using the ADD representation thereby avoiding the need to enumerate the state space. Later Boutilier, Reiter, and Price (2001) developed the Symbolic Dynamic Programming (SDP) algorithm in the context of situation calculus. This algorithm provided a framework for dynamic programming solutions to Relational MDPs that was later employed in several formalisms and systems (Kersting, Otterlo, and Raedt 2004; Höolldobler, Karabaev, and Skvortsova 2006; Sanner 2008; Wang, Joshi, and Khardon 2008). One of the important ideas in SDP was to represent stochastic actions as deterministic alternatives under nature's control. This helps segregate regression over deterministic action alternatives from the probabilities of action effects. This segregation is necessary when transition functions are represented as relational schemas. The basic outline of the relational value iteration algorithm is as follows:

1. **Regression:** Regress the $n-1$ step-to-go value function V^{n-1} over every deterministic variant a_i^j of every action a_i to produce $RegV_i^j$
2. **Add Action Variants:** Generate the Q-function $QReg_i^a = \Sigma_j Pr(a_i^j)RegV_i^j$ for each action a_i
3. **Object Maximization:** Maximize over the action parameters of $QReg_i^a$ to produce Q^{a_i} for each action a_i, thus obtaining the value achievable by the best ground instantiation of a_i.
4. **Maximize over Actions:** Generate the n step-to-go value function $V^n = Max_i[R(S) + \gamma Q^{a_i}]$

First Order Decision Diagrams

This section briefly reviews previous work on FODDs and their use for relational MDPs (Wang, Joshi, and Khardon 2008). We use standard terminology from first order logic (Lloyd 1987). A First order decision diagram is a labeled directed acyclic graph, where each non-leaf node has exactly 2 outgoing edges labeled `true` and `false` . The non-leaf nodes are labeled by atoms generated from a predetermined signature of predicates, constants and an enumerable set of variables. Leaf nodes have non-negative numeric values. The signature also defines a total order on atoms, and the FODD is ordered with every parent smaller than the child according to that order. Two examples of FODDs are given in Figure 1; in these and all diagrams in the paper left going edges represent the `true` branches and right edges are the `false` branches.

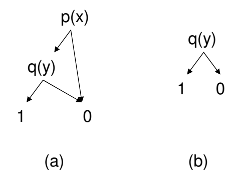

Figure 1: An example FODD

Thus, a FODD is similar to a formula in first order logic. Its meaning is similarly defined relative to interpretations of the symbols. An *interpretation* defines a domain of objects, identifies each constant with an object, and specifies a truth value of each predicate over these objects. In the context of relational MDPs, an interpretation represents a state of the world with the objects and relations among them. Given a FODD and an interpretation, a *valuation* assigns each variable in the FODD to an object in the interpretation. Following Groote and Tveretina (2003), the semantics of FODDs are defined as follows. If B is a FODD and I is an interpretation, a valuation ζ that assigns a domain element of I to each variable in B fixes the truth value of every node atom in B under I. The FODD B can then be traversed in order to reach a leaf. The value of the leaf is denoted $Map_B(I,\zeta)$. $Map_B(I)$ is then defined as $max_\zeta Map_B(I)$, i.e. an aggregation of $Map_B(I,\zeta)$ over all valuations ζ. For example, consider the FODD in Figure 1(a) and the interpretation I with objects a, b and where the only true atoms are $p(a), q(b)$. The valuations $\{x/a, y/a\}$, $\{x/a, y/b\}$, $\{x/b, y/a\}$, and $\{x/b, y/b\}$, will produce the values 0, 1, 0, 0 respectively. By the max aggregation semantics, $Map_B(I) = max\{0,1,0,0\} = 1$. Thus, this FODD is equivalent to the formula $\exists x, y, p(x) \wedge q(y)$. In general, max aggregation yields existential quantification when leaves are binary. When using numerical values we can similarly capture value functions for relational MDPs.

Akin to ADDs, FODDs can be combined under arithmetic operations, and reduced in order to remove redundancies. Groote and Tveretina (2003) introduced four reduction operators (R1 $\cdots$ R4) and these were augmented with five more (R5 $\cdots$ R9) (Wang, Joshi, and Khardon 2008). Intuitively, redundancies in FODDs arise in two different ways. The first, observes that some edges may never be traversed by any valuation. Reduction operators for such redundancies are called strong reduction operators. The second requires more subtle analysis: there may be parts of the FODD that are traversed under some valuations but because of the max aggregation, the valuations that traverse those parts are never instrumental in determining the map. Operators for such redundancies are called weak reductions operators. Strong reductions preserve $Map_B(I,\zeta)$ for every valuation ζ (thereby preserving $Map_B(I)$) and weak reductions preserve $Map_B(I)$ but not necessarily $Map_B(I,\zeta)$ for every ζ. Using this classification R1-R5 are strong reductions and R6-R9 are weak reductions. Another subtlety arises because for RMDP domains we may have some background knowledge about the predicates in the domain. For example, in the blocks world, if block a is clear then $on(x, a)$ is false for all values of x. We denote such *background knowledge* by $\mathcal{B}$ and allow reductions to rely on such knowledge. Below, we discuss operator R7 in some detail because of its relevance to the next section.

We use the following notation. If e is an edge from node n to node m, then source(e) $= n$, target(e) $= m$ and sibling(e) is the other edge out of n. For node n, the symbols $n_{\downarrow t}$ and $n_{\downarrow f}$ denote the `true` and `false` edges out of n respectively. $l(n)$ denotes the atom associated with node n. Node formulas (NF) and edge formulas (EF) are defined recursively as follows. For a node n labeled $l(n)$ with incoming edges $e_1, \ldots, e_k$, the node formula NF(n) $= (\vee_i \text{EF}(e_i))$. The edge formula for the `true` outgoing edge of n is EF($n_{\downarrow t}$) $=$ NF(n) $\wedge l(n)$. The edge formula for the `false` outgoing edge of n is EF($n_{\downarrow f}$) $=$ NF(n) $\wedge \neg l(n)$. These formulas, where all variables are existentially quantified, capture the conditions under which a node or edge are reached. Similarly, if B is a FODD and p is a path from the root to a leaf in B, then the path formula for p, denoted by PF(p) is the conjunction of literals along p. The variables of p, are denoted $\vec{x^p}$. When $\vec{x^p}$ are existentially quantified, satisfiability of PF(p) under an interpretation I is a necessary and sufficient condition for the path p to be traversed by some valuation under I. If ζ is such a valuation, then we define $Path_B(I, \zeta) = p$. The leaf reached by path p is denoted as $leaf(p)$. We let PF(p)$\backslash Lit$ denote the path formula of path p with the literal Lit removed (if it was present) from the conjunction. Let $n^{n_{\downarrow t}}.lit = l(n)$ and $n^{n_{\downarrow f}}.lit = \neg l(n)$. FODD combination under the subtract operation is denoted by the binary operator $\ominus$.

The R7 Reduction: Intuitively, given a FODD B, edges e_1 and e_2 in B, if for every valuation going through edge e_2, there always is another valuation going down e_1 that gives a better value, we can replace target(e_2) by 0 without affecting $Map_B(I)$ for any interpretation I. R7 formalizes this notion with a number of conditions. For example:

(P7.2) : $\mathcal{B} \models \forall \vec{u}, [[\exists \vec{w}, \text{EF}(e_2)] \rightarrow [\exists \vec{v}, \text{EF}(e_1)]]$ where $\vec{u}$ are the variables that appear in both $target(e_1)$ and $target(e_2)$, and $\vec{v}$, $\vec{w}$ are the remaining variables in EF(e_1), EF(e_2) respectively. This condition requires that for every valuation ζ_1 that reaches e_2 there is a valuation ζ_2 that reaches e_1 such that ζ_1 and ζ_2 agree on all variables that appear in both $target(e_1)$ and $target(e_2)$. **(V7.3)** : all leaves in $D = target(e_1) \ominus target(e_2)$ have non-negative values, denoted as $D \geq 0$. In this case for any fixed valuation it is better to follow e_1 instead of e_2. **(S1)** : There is no path from the root to a leaf that contains both e_1 and e_2.

The operator R7-replace(b, e_1, e_2) replaces $target(e_2)$ with a 0 and the operator R7-drop(e_1, e_2) drops the node $q = source(e_2)$ and connects its parents to $target(sibling(e_2))$. It is safe to perform R7-replace when **(P7.2)** , **(V7.3)** , and **(S1)** hold. Several other conditions exist allowing safe applicability for R7-replace and R7-drop but we omit the details due to space constraints.

R7 captures the fundamental intuition behind weak reductions and hence is widely applicable. Unfortunately it is also very expensive to run. R7-replace conditions have to be tested for all pairs of edges in the diagram. Each test requires theorem proving with disjunctive first order formulas.

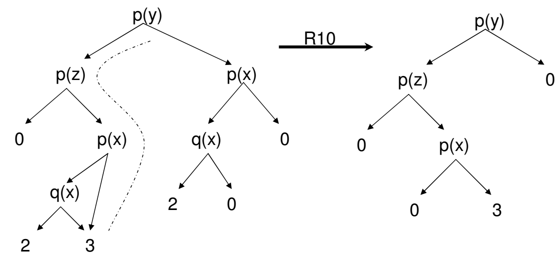

Figure 2: Example of R10 reduction

VI with FODDs

In previous work we showed how to capture the reward function and domain dynamics using FODDs and presented a value iteration algorithm along the lines described in the last section. All operations (regression, $plus$, $times$, max) that are needed in the algorithm can be performed by combining FODDs. To keep the diagram size manageable FODDs have to be reduced at every step. One of the advantages of this algorithm is that due to the semantics of FODDs, the step of Object Maximization comes for free. However, efficient and successful reductions are the key to this procedure. The reductions existing so far do not provide sufficient coverage and, as discussed for R7, can be expensive to apply. Thus the set of reductions was not sufficient for practical application of the value iteration algorithm. In the next section we present new reduction operations and speedup techniques to make value iteration practical.

Additional Reduction Operators

The R10 Reduction

A path in FODD B is dominated if whenever a valuation traverses it, there is always another valuation traversing another path and reaching a leaf of greater or equal value. Now if all paths through an edge e are dominated, then no valuation crossing that edge will ever determine the map under max aggregation semantics. In such cases we can replace target(e) by a 0 leaf. This is the basic intuition behind the R10 operation.

Although its objective is the same as that of R7-replace, R10 is faster to compute and has two advantages over R7-replace. First, because paths can be ranked by the value of the leaf they reach, we can perform a single ranking and check for all dominated paths (and hence all dominated edges). Hence, while all other reduction operators are local, R10 is a global reduction. Second, the theorem proving required for R10 is always on conjunctive formulas with existentially quantified variables. This gives a speedup over R7-replace. Figure 2 shows an example of one pass of R10 on a FODD. Here R10 is able to identify one path (shown along a curved indicator line) that dominates all other paths.

To preserve the map, only this one path needs to be preserved. This is reflected in the resultant FODD produced. To achieve the same reduction, R7-replace takes 2-3 passes depending on the order of application. Since every pass of R7-replace has to check for implication of edge formulas for every pair of edges, this can be expensive. On the other hand, there are cases where R10 is not applicable but R7-replace is.

Procedure 1 *R10(B)*

1. *Let E be the set of all edges in B*
2. *Let $P = [p_1, p_2 \cdots p_n]$ be the list of all paths from the root to a leaf in B, sorted in descending order by the value of the leaf reached by the path. Thus p_1 is a path reaching the highest leaf and p_n is a path reaching the lowest leaf.*
3. *For $j = 1$ to n, do the following*
   (a) *Let E_{p_j} be the set of edges on p_j*
   (b) *If $\neg\exists i$, $i < j$ such that $\mathcal{B} \models (\exists \vec{x^{p_j}}$, PF($p_j$)) $\rightarrow$ ($\exists \vec{x^{p_i}}$, PF(p_i)), then set $E = E - E_{p_j}$*
4. *For every edge $e \in E$, set target(e) = 0 in B*

Theorem 1 *Let B be any FODD. If B' = R10(B) then $\forall$ interpretations I, $Map_B(I) = Map_{B'}(I)$*

The theorem is proved by defining a notion of instrumental paths that determine the map for some interpretations and showing that edges on instrumental paths are not removed. The details are omitted due to space constraints.

The R11 Reduction

Consider the FODD B in Figure 1(a). Clearly, with no background knowledge this diagram cannot be reduced. Now assume that the background knowledge $\mathcal{B}$ contains a rule $\forall x, [q(x) \rightarrow p(x)]$. In this case if there exists a valuation that reaches the 1 leaf, there must be another such valuation ζ that agrees on the values of x and y. ζ dominates the other valuations under the max aggregation semantics. The background knowledge rule implies that for ζ, the test at the root node is redundant. However, we cannot set the left child of the root to 0 since the entire digram will be eliminated. Therefore R7 is not applicable, and similarly none of the other existing reductions is applicable. Yet similar situations arise often in runs of the value iteration algorithm.[1] We introduce the R11 reduction operator that can handle such situations. R11 reduces the FODD in Figure 1(a) to the FODD in Figure 1(b)

Let B be a FODD, n a node in B, e an edge such that $e \in \{n_{\downarrow t}, n_{\downarrow f}\}$, e' = sibling(e) (so that when $e = n_{\downarrow t}$, $e' = n_{\downarrow f}$ and vice versa), and P the set of all paths from the root to a non-zero leaf going through edge e. Then the $R11(B, n, e)$ reduction drops node n from diagram B and connects its parents to target(e). R11 preserves the map when the following conditions are satisfied.

Condition 1 target(e') $= 0$

Condition 2 $\forall p \in P, \mathcal{B} \models [\exists \vec{x^p}$, PF($p$)$\backslash n^e.lit \wedge n^{e'}.lit] \rightarrow [\exists \vec{x^p}$, PF($p$)]

Theorem 2 *If $B' = R11(B, n, e)$, and conditions 1 and 2 hold, then $\forall$ interpretations I, $Map_B(I) = Map_{B'}(I)$*

Proof: Let I be any interpretation and let Z be the set of all valuations. We can divide Z into three disjoint sets depending on the path taken by valuations in B under I. Z^e - the set of all valuations crossing edge e, $Z^{e'}$ - the set of all valuations crossing edge e' and Z^{other} - the set of valuations not reaching node n. We analyze the behavior of the valuations in these sets under I.

- Since structurally the only difference between B and B' is that in B' node n is bypassed, all paths from the root to a leaf that do not cross node n remain untouched. Therefore $\forall \zeta \in Z^{other}$, $Map_B(I, \zeta) = Map_{B'}(I, \zeta)$.
- Since, in B' the parents of node n are connected to target(e), all valuations crossing edge e and reaching target(e) in B under I will be unaffected in B' and will, therefore, produce the same map. Thus $\forall \zeta \in Z^e$, $Map_B(I, \zeta) = Map_{B'}(I, \zeta)$.
- Now, let m denote the node target(e) in B. Under I, all valuations in $Z^{e'}$ will reach the 0 leaf in B but they will cross node m in B'. Depending on the leaf reached after crossing node m, the set $Z^{e'}$ can be further divided into 2 disjoint subsets. $Z^{e'}_{zero}$ - the set of valuations reaching a 0 leaf and $Z^{e'}_{nonzero}$ - the set of valuations reaching a non-zero leaf. Clearly $\forall \zeta \in Z^{e'}_{zero}$, $Map_B(I, \zeta) = Map_{B'}(I, \zeta)$.

  By the structure of B, every $\zeta \in Z^{e'}_{nonzero}$, traverses some $p \in P$, that is, (PF(p)$\backslash n^e.lit \wedge n^{e'}.lit$)$\zeta$ is true in I. Condition 2 states that for every such ζ, there is another valuation η such that (PF(p))η is true in I, so η traverses the same path. However, every such valuation η must belong to the set Z^e by the definition of the set Z^e. In other words, in B' every valuation in $Z^{e'}_{nonzero}$ is dominated by some valuation in Z^e.

From the above argument we conclude that in B' under I, every valuation either produces the same map as in B or is dominated by some other valuation. Under the max aggregation semantics, therefore, $Map_B(I) = Map_{B'}(I)$. ■

Subtracting Apart

Consider the FODD B in Figure 3. Intuitively a weak reduction is applicable on this diagram because of the following argument. Consider a valuation $\zeta = \{x\backslash 1, y\backslash 2, z\backslash 3\}$ crossing edge e_2 under interpretation I. Then $I \models \mathcal{B} \rightarrow \neg p(1) \wedge p(2)$. Therefore there must be a valuation $\eta = \{x \setminus 2, z \setminus 3\}$ (and any value for y), that crosses edge e_1. Now depending on the truth value of $I \models \mathcal{B} \rightarrow q(1)$ and $I \models \mathcal{B} \rightarrow q(2)$, we have four possibilities of where ζ and η would reach after crossing the nodes target(e_2) and target(e_1) respectively. In

[1] This happens naturally, without the artificial background knowledge used for our example. The main reason is that standardizing apart (which is discussed further below) introduces multiple renamed copies of the same atoms in the different diagrams. When the diagrams are added, many of the atoms are redundant but some are not removed by old operators. These atoms may end up in a parent-child relation with weak implication from child to parent, similar to the example given.

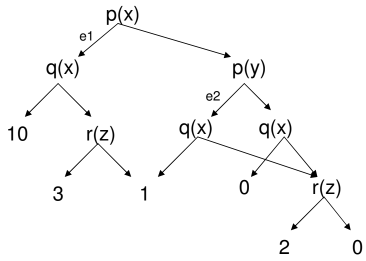

Figure 3: Sub-Apart

all cases, $Map_B(I, \eta) \geq Map_B(I, \zeta)$. Therefore we should be able to replace target(e_2) by a 0 leaf. A similar argument shows that we should also be able to drop the node source(e_2). Surprisingly, though, none of the R7 conditions apply in this case and this diagram cannot be reduced. On closer inspection we find that the reason for this is that the conditions **(P7.2)** and **(V7.3)** are too restrictive. **(V7.3)** holds but **(P7.2)** requires valuations to agree on the value of x and z and this does not hold. However, from our argument above, for η to dominate ζ, the two valuations need not agree on the value of x. We observe that if we rename variable x so that its instances are different in the subFODDs rooted at target(e_1) and target(e_2) (i.e. we standardized apart w.r.t. x) then both **(P7.2)** and **(V7.3)** go through and the diagram can be reduced. To address this problem, we introduce a new FODD subtraction algorithm *sub-apart*. Given diagrams B_1 and B_2 the algorithm tries to standardize apart as many of their common variables as possible, while keeping the condition $B_1 \ominus B_2 \geq 0$ true. The algorithm returns a 2-tuple $\{T, V\}$, where T is a boolean variable indicating whether the combination can produce a diagram that has no negative leaves when all variables except the ones in V are standardized apart.

Procedure 2 *sub-apart(A, B)*

1. *If A and B are both leaves return $\{A - B \geq 0?, \{\}\}$*
2. *If $l(A) < l(B)$, let*
   (a) *$\{L, V_1\}$ = sub-apart(target($A_{\downarrow t}$), B)*
   (b) *$\{R, V_2\}$ = sub-apart(target($A_{\downarrow f}$), B)*
   Return $\{L \wedge R, V_1 \cup V_2\}$
3. *If $l(A) > l(B)$, let*
   (a) *$\{L, V_1\}$ = sub-apart(A, target($B_{\downarrow t}$))*
   (b) *$\{R, V_2\}$ = sub-apart(A, target($B_{\downarrow f}$))*
   Return $\{L \wedge R, V_1 \cup V_2\}$
4. *If $l(A) = l(B)$, let V be the variables of A (or B). Let*
   (a) *$\{LL, V_3\}$ = sub-apart(target($A_{\downarrow t}$), target($B_{\downarrow t}$))*
   (b) *$\{RR, V_4\}$ = sub-apart(target($A_{\downarrow f}$), target($B_{\downarrow f}$))*
   (c) *$\{LR, V_5\}$ = sub-apart(target($A_{\downarrow t}$), target($B_{\downarrow f}$))*
   (d) *$\{RL, V_6\}$ = sub-apart(target($A_{\downarrow f}$), target($B_{\downarrow t}$)*
   (e) *If $LL \wedge RR = false$, return $\{false, V_3 \cup V_4\}$*
   (f) *If $LR \wedge RL = false$ return $\{true, V \cup V_3 \cup V_4\}$*
   (g) *Return $\{true, V_3 \cup V_4 \cup V_5 \cup V_6\}$*

The next theorem shows that the procedure is correct. The variables common to B_1 and B_2 are denoted by $\vec{u}$ and $B^{\vec{w}}$ denotes the combination diagram of B_1 and B_2 under the subtract operation when all variables except the ones in $\vec{w}$ are standardized apart. Let n_1 and n_2 be the roots nodes of B_1 and B_2 respectively.

Theorem 3 *sub-apart(n_1, n_2) = $\{true, \vec{v}\}$ implies $B^{\vec{v}}$ contains no negative leaves and sub-apart(n_1, n_2) = $\{false, \vec{v}\}$ implies $\neg\exists\vec{w}$ such that $\vec{w} \subseteq \vec{u}$ and $B^{\vec{w}}$ contains no negative leaves.*

The theorem is proved by indunction on the sum of number of nodes in the diagrams. The details are omitted due to space constraints. The theorem shows that the algorithm is correct but does not guarantee minimality. In fact, the smallest set of variables $\vec{w}$ for $B^{\vec{w}}$ to have no negative leaves may not be unique (Wang, Joshi, and Khardon 2008). One can also show that the output of sub-apart may not be minimal. In principle, one can use a greedy procedure that standardizes apart one variable at a time and arrives at a minimal set $\vec{w}$. However, although *sub-apart* does not produce a minimal set, we prefer it to the greedy approach because it is fast and often generates a small set $\vec{w}$ in practice. We can now define new conditions for applicability of R7: **(V7.3S)** : sub-apart(target(e_1), target(e_2)) = $\{true, V_1\}$ and **(P7.2S)** : $\mathcal{B} \models \forall V_1$, [[$\exists\vec{w}$, EF($e_2$)] $\rightarrow$ [$\exists\vec{v}$, EF(e_1)]] where as above and $\vec{v}$, $\vec{w}$ are the remaining variables (i.e. not in V_1) in EF(e_1), EF(e_2) respectively. One can verify that conditions **(P7.2S)** , **(V7.3S)** subsume the previous conditions, as well as other conditions for applicability of R7-replace. Similar simplification occurs for the conditions of R7-drop. Thus the use of sub-apart both simplifies the conditions to be tested and provides more opportunities for reductions. We use the new conditions with sub-apart (instead of the old conditions) whenever testing for applicability of R7.

More Speedup Techniques

Not Standardizing Apart

Recall that the FODD-based VI algorithm must add functions represented by FODDs (in Steps 2 and 4) and take the maximum over functions represented by FODDs (in Step 4). Since the individual functions are independent functions of the state, the variables of different functions are not related to one another. Therefore, before adding or maximizing, the algorithm in (Wang, Joshi, and Khardon 2008) standardizes apart the diagrams - that is, all variables in the diagrams are given new names so they do not constrain each other. On the other hand, since the different diagrams are structurally related this often introduces redundancies (in the form of renamed copies of the same atoms) that must be be removed by reduction operators. However, our reduction operators are not ideal and avoiding this step can lead to significant speedup in the system. Here we observe that for maximization (in Step 4) standardizing apart is not needed and therefore can be avoided.

Theorem 4 *Let B_1 and B_2 be FODDs. Let B be the result of combining B_1 and B_2 under the max operation when B_1 and B_2 are standardized apart. Let B' be the result* *of combining B_1 and B_2 under the max operation when B_1 and B_2 are not standardized apart. $\forall$ interpretations I, $Map_B(I) = Map_{B'}(I)$.*

The theorem is proved by showing that for any I a valuation for the maximizing diagram can be completed into a valuation over the combined diagram giving the same value. The details are omitted due to space constraints.

Value Approximation

Reductions help keep the diagrams small in size by removing redundancies but when the true n step-to-go value function itself is large, legal reductions cannot help. There are domains where the true value function is unbounded. For example in the tireworld domain from the international planning competition, where the goal is always to get the vehicle to a destination city, one can have a chain of cities linked to one another up to the destination. This chain can be of any length. Therefore when the value function is represented using state abstraction, it must be unbounded. Sanner (2008) observes that SDP-like algorithms are less effective on domains where the dynamics lead to such transitive structure and every iteration of value iteration increases the size of the n step-to-go value function. In other cases the value function is not infinite but is simply too large to manipulate efficiently. When this happens we can resort to approximation keeping as much of the structure of the value function as possible while maintaining efficiency. One must be careful about the tradeoff here. Without approximation runtimes can be prohibitive and too much approximation causes loss of structure and value. We next present two methods to get approximations.

Not Standardizing Apart Action Variants

Standardizing apart of action variant diagrams before adding them is required for the correctness of the FODD based VI algorithm. That is, if we do not standardize apart action variant diagrams before adding them, value may be lost (Wang, Joshi, and Khardon 2008). Intuitively, this is true since different paths in the value function share atoms and variables. Now, for a fixed action, the best variable binding and corresponding value for different action variants may be different. Thus, if the variables are forced to be the same for the variants, we may rule out viable combinations of value. On the other hand, the value obtained is a lower bound on the true value. This is because every path in the diagram resulting from not standardizing apart is present in the diagram resulting from standardizing apart. We call this approximation method *non-std-apart* and use it as a heuristic to speed up computation. Although this heuristic may cause loss of structure in the representation of the value function, we have observed that in practice it gives significant speedup while maintaining most of the relevant structure.

Merging Leaves

The use of FODDs also allows us to approximate the value function in a simple and controlled way. Here we followed the approximation techniques of SPUDD (St-Aubin, Hoey, and Boutilier 2000) where they were used for propositional problems. The idea is to reduce the size of the diagram by merging substructures that have similar values. One way of doing this is to merge leaves that are within a certain distance of each other. Another way is to just reduce the precision of the leaves. We experimented with both techniques. The granularity of approximation, however, becomes an extra parameter for the system and has to be chosen carefully.

Experimental Results

We made two additional extensions to the basic algorithm:

- The basic algorithm does not allow actions costs because they may lead to diagrams with negative leaves. We note that action costs can be supported as long as there is at least one zero cost action. To see this recall the VI algorithm. The appropriate place to add action costs is just before the Object Maximization step. However, because this step is followed by maximizing over the action diagrams, if at least one action has 0 cost (if not we can create a nop action), the resultant diagram after maximization will never have negative leaves. Therefore we safely convert negative leaves before the maximization step to 0 and thereby avoid conflict with the reduction procedures.
- Our value iteration algorithm cannot handle universal goals. We can handle ground conjunctive goals but in some sense this defeats the purpose (unless the goal is generic) because we would be solving for each concrete ground goal separately. We therefore follow Sanner (2008) and use an approximation of the true value function, that results from a simple additive decomposition of the goal predicates. Thus we plan separately for a generic version of each predicate. Then at execution time, when given a concrete goal, we approximate the true value function by the sum of the generic versions over each ground goal predicate. This is clearly not an exact calculation and will not work in every case. On the other hand, it considerably extends the scope of the technique and works well in many situations.

We implemented the FODD-PLANNER system, plan execution routines and evaluation routines under Yap Prolog 5.1.2. Our implementation uses a simple theorem prover making use of the Prolog engine for its search, and supporting background knowledge by state flooding. That is, to prove $\mathcal{B} \models X \rightarrow Y$, where X is a ground conjunction, we use the Prolog engine to find and add to X all atoms provable from X. We then assert the revised X to the database and use the Prolog engine to prove Y. Because of our restricted language, the reasoning problem is decidable and our theorem prover is complete. We use all reductions except R7-replace. Reductions were applied every time two diagrams are combined. For each domain, we constructed by hand background knowledge restricting arguments of predicates (e.g. a box can only be at one city in any time so $Bin(b, c_1), Bin(b, c_2) \rightarrow (c_1 = c_2)$). This is useful in the process of simplifying diagrams.

We ran experiments on a standard benchmark problem as well as probabilistic planning domains from the international planning competitions held in 2004 and 2006. We restricted planning time to 4 hours in order to make a fair

	Coverage	Time (ms)	Reward
GPT	100%	2220	57.66
Policy Iteration with policy language bias	46.66%	60466	36
Re-Engg NMRDPP	10%	290830	-387.7
FODD-Planner	100%	65000	47.3

Figure 4: fileworld domain results

comparison with other offline planners in the competitions. All experiments were run on a Linux machine with an Intel Pentium D processor running at 3 GHz, with 2 GB of memory. All timings, rewards and plan-lengths we report are averages over 30 rounds. Next we present results on three domains.

The Logistics Benchmark Problem

This is the boxworld problem introduced by Boutilier, Reiter, and Price (2001) that has been used as a standard example for exact solution methods for relational MDPs. The domain consists of boxes, cities and trucks. The objective is to get certain boxes to certain cities by loading, unloading and driving. For the benchmark problem, the goal is the existence of a box in Paris. The load and unload actions have probabilities of success that depend on whether it is raining or not. One important assumption made in this domain is that all cities are reachable from each other. This assumption makes it a domain without transitive structure. Like ReBel (Kersting, Otterlo, and Raedt 2004) and FOADD (Sanner 2008) we are able to solve this MDP and identify all relevant partitions of the optimal value function and in fact the value function converges after 10 iterations. In terms of run time the FODD-PLANNER without approximation is slower than ReBel but using approximation, with merging leaves within resolution 0.1 from each other, the runtimes are comparable.

The Fileworld Domain

This domain was part of the probabilistic track of IPC-4 (2004) (http://ls5-web.cs.uni-dortmund.de/∼edelkamp/ipc-4/). The domain consists of files and folders. Every file obtains a random assignment to a folder at execution time and the goal is to place each file in its assigned folder. There is a cost of 100 to handle a folder and a cost of 1 to place a file in a folder. Results have been published for one problem instance which consisted of thirty files and five folders. The optimal policy for this domain is to first get the assignments of files to folders and then handle each folder once, placing all files that were assigned to it. Since the goal is conjunctive we used the additive goal decomposition discussed above. We were able to plan for a generic goal $filed(a)$ and use the policy to solve for any number of files. This domain is ideal for abstract solvers since the optimal value function and policy are compact and can be found quickly. The FODD-PLANNER was able to achieve convergence within 4 iterations even without approximation. The policy was generated in less than a minute. Of the 6 systems that competed on this track, results have been published for 3 on the website cited above. Figure 4 compares the performance of FODD-PLANNER to the others. We observe that we rank ahead of all except GPT in terms of total reward and coverage (both FODD-PLANNER and GPT achieve full coverage). We do not get perfect reward in spite of a converged exact value function because of the additive decomposition of rewards. The policy generated is optimal for one file but it is still a heuristic for many files.

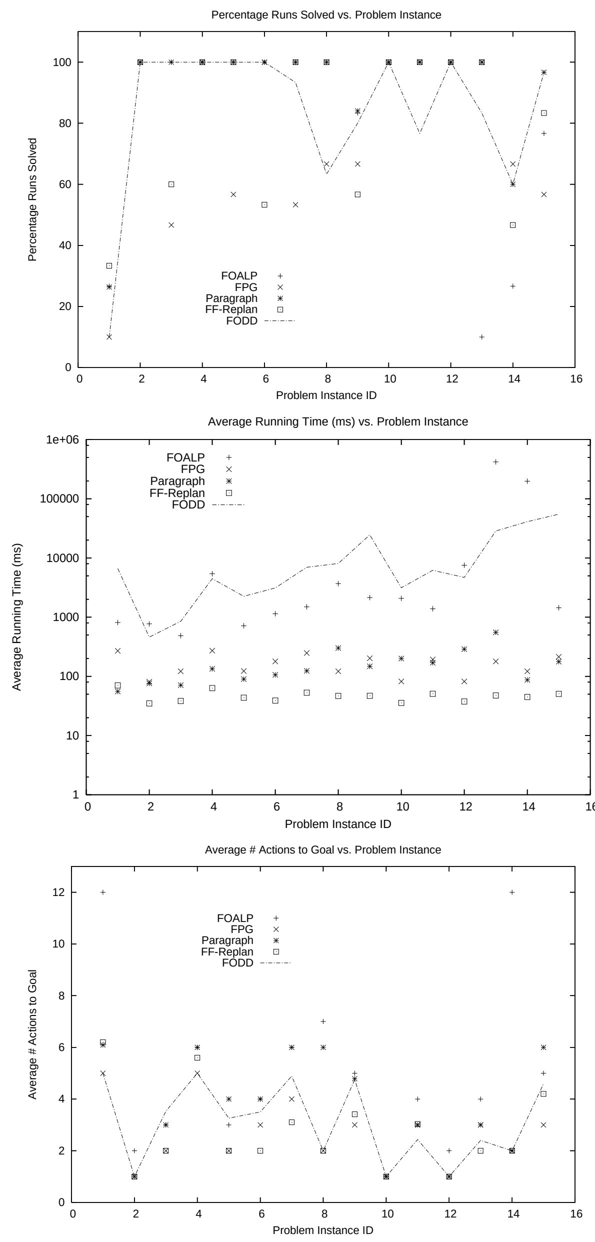

Figure 5: Result of tireworld experiments

The Tireworld Domain

This domain was part of the probabilistic track of IPC-5 (2006) (http://www.ldc.usb.ve/∼bonet/ipc5/). The domain consists of a network of locations (or cities). A vehicle starts from one city and moves from city to city with the objective of reaching a destination city. Moves can only be made between cities that are directly connected by a road. However, on any move the vehicle may lose a tire with 40% probability. Some cities have a spare tire that can be loaded onto the vehicle. If the vehicle contains a spare tire, the flat tire can be changed with 50% success probability. This domain is simple but not trivial owing to the possibility of a complex network topology and high probabilities of failure. Participants at IPC-5 competed over 15 problem instances on this domain. To limit offline planning time we restricted FODD-PLANNER to 7 iterations without any approximation for the first 3 iterations and with the non-std-apart approximation for the remaining iterations. We did not use merging of leaves. The policy was generated in 2.5 hours. The performance of FODD-PLANNER and systems competing in the probabilistic track of IPC-5, for which, data is published on the website cited above, is summarized in Figure 5. The figure shows a comparison of the percentage of problem instances each planner was able to solve (coverage), the average time per instance taken by each planner to generate an online solution, and the average number of actions taken by each planner to reach the goal on every instance. We observe that the overall performance of FODD-PLANNER is competitive with (and in a few cases better than) the other systems. Runtimes to generate online solutions are high for FODD-PLANNER but are comparable to FOALP which is the only other first-order planner. On the other hand, in comparison with the other systems, we are able to achieve high coverage and short plans on most of the problems.

Above we present the performance of all improvements simultaneously. In general, different improvements are useful for different contexts, so it is hard to quantify the contributions of each component (reductions, sub-apart, approximations) separately. For example, merging leaves is crucial for efficiency in the logistics domain but does not help in tireworld. A quantitative illustration of performance across domains will be provided in the full version of the paper.

Conclusion and Future Work

The main contribution of this paper is the introduction of FODD-PLANNER, a relational planning system based on First Order Decision Diagrams. This is the first planning system that uses lifted algebraic decision diagrams as its representation language and successfully solves planning problems from the international planning competition. FODD-PLANNER provides several improvements over previous work on FODDs (Wang, Joshi, and Khardon 2008). The improvements include the reduction operators R10, R11 and the Sub-Apart operator, and several speedup and value approximation techniques. Taken together, these improvements provide substantial speedup making the approach practical. Therefore, the results show that abstraction through compact representation is a promising approach to stochastic planning.

While this work makes a significant improvement in terms of run time and processing power of FODDs there are many open questions. Our set of reductions is still heuristic and does not guarantee a canonical form for diagrams which is instrumental for efficiency of propositional algorithms. Identifying such "complete" sets of reductions operators and canonical forms is an interesting challenge. Identifying a practically good set of operators trading off complexity for reduction power is crucial for further speedup. Another improvement may be possible by using the FODD based policy iteration algorithm (Wang and Khardon 2007). This may allow us to avoid approximation of infinite size value functions in cases where the policy is still compact (for which there are known examples). Finally, this work suggests the potential of using FODDs as the underlying representation for relational reinforcement learning. Therefore, it will be interesting to develop learning algorithms for FODDs.

Acknowledgments

We thank Kristian Kersting for valuable input on the system and insightful discussions.

References

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order mdps. In *Proceedings of IJCAI*, 690–700.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving Relational Markov Decision Processes. In *Journal of Artificial Intelligence Research* 25:75–118.

Gretton, C., and Thiebaux, S. 2004. Exploiting first-order regression in inductive policy selection. In *Proceedings of UAI*.

Groote, J., and Tveretina, O. 2003. Binary decision diagrams for first order predicate logic. *Journal of Logic and Algebraic Programming* 57:1–22.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *Proceedings of UAI*, 279–288.

Höolldobler, S.; Karabaev, E.; and Skvortsova, O. 2006. FluCaP: a heuristic search planner for first-order MDPs. *Journal of Artificial Intelligence Research* 27:419–439.

Kersting, K.; Otterlo, M. V.; and Raedt, L. D. 2004. Bellman goes relational. In *Proceedings of ICML*.

Lloyd, J. 1987. *Foundations of Logic Programming*. Springer Verlag. Second Edition.

Puterman, M. L. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley.

Sanner, S., and Boutilier, C. 2006. Practical linear value-approximation techniques for first-order MDPs. In *Proceedings of UAI*.

Sanner, S. 2008. *First-order decision-theoretic planning in structured relational environments*. Ph.D. Dissertation, University of Toronto.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2000. Apricodd: Approximate policy construction using decision diagrams. In *Proceedings of NIPS*, 1089–1095.

Wang, C., and Khardon, R. 2007. Policy iteration for relational mdps. In *Proceedings of UAI*.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational mdps. *JAIR* 31:431–472.