Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling

Efficient Policy Construction for MDPs Represented in Probabilistic PDDL

Boris Lesner and **Bruno Zanuttini**
GREYC, Université de Caen Basse-Normandie, CNRS UMR 6072, ENSICAEN
Boulevard du Maréchal Juin 14 032 Caen Cedex, France

Abstract

We present a novel dynamic programming approach to computing optimal policies for Markov Decision Processes compactly represented in grounded Probabilistic PDDL. Unlike other approaches, which use an intermediate representation as Dynamic Bayesian Networks, we directly exploit the PPDDL description by introducing dedicated backup rules. This provides an alternative approach to DBNs, especially when actions have highly correlated effects on variables. Indeed, we show interesting improvements on several planning domains from the International Planning Competition. Finally, we exploit the incremental flavor of our backup rules for designing promising approaches to policy revision.

Introduction

Markov Decision Processes have become a standard framework for modelling decision-theoretic planning tasks. While classical approches rely on dynamic programming algorithms such as Value Iteration, the AI community quickly focused on factored approches to solve MDPs, since practical problems involves millons or even billions of states. Central to such methods are avoiding to enumerate all states and exploiting the structure present in most practical problems. Factored MDP algorithms represent states and actions in a symbolic way. States are described using valuations over a set of variables and actions describe changes on variables.

The Probabilistic Planning Domain Description Language (Younes and Littman 2004) is a widely adopted language for specifying probabilistic and decision-theoretic planning problems. However, state-of-the-art approaches for computing optimal policies for MDPs rely on a representation by Dynamic Bayesian Networks (DBNs), and therefore, though very efficient on most problems, they require a translation from PPDDL.

We propose a new approach for computing optimal policies for MDPs, which directly uses the PPDDL specification. Our approach relies on classical dynamic programming, but performs action backup using the new notion of a "frameless" action-value. Using such backups in classical value iteration yields a new algorithm, which turns out to be complementary to other factored approaches. We demonstrate this on some International Planning Competition benchmarks. Finally, we show that frameless action-values open promising perspectives for policy revision.

Preliminaries

Markov Decision Processes A Markov Decision Process $M = (S, A, T, R)$ involves finite sets of *states* S and *actions* A. For $s, s' \in S$ and $a \in A$, the *transition probability* $T(s'|s, a)$ denotes the probability of reaching state s' after taking action a in s. For states s, s' and action a, the *reward function* $R(s, a, s')$ determines the payoff of taking action a in state s and ending up in state s'.

Given an MDP M, one can compute an *optimal policy* $\pi_h : S \to A$ which maximizes expected reward at some given horizon h. π_h can be derived from the optimal value function at h, written $V_h(s) = \max_{a \in A} Q^a_{V_{h-1}}(s)$ with

$$Q^a_V(s) = \sum_{s' \in S} T(s'|s, a) \left(R(s, a, s') + \gamma V(s')\right) \quad (1)$$

and $V_0(s) = 0$ for all $s \in S$. The optimal policy π_h is then given by $\pi_h(s) = \operatorname{argmax}_a Q^a_{V_{h-1}}(s)$. The *discount factor* $\gamma \in [0, 1]$, reduces the importance of future rewards. Infinite-horizon value functions can be approximated arbitrarily closely when $\gamma < 1$: let $V^* = \lim_{h \to \infty} V_h$ denote the infinite horizon value function, then as soon as $||V_h - V_{h-1}||_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$ holds, $||V_h - V^*||_\infty \leq \epsilon$ holds.

PPDDL The Probabilistic Planning Domain Description Language (Younes and Littman 2004) allows to model probabilistic and decision-theoretic planning problems. We present here its components, domains and problems.

Planning domains consist of predicates and action schemata. Predicates encode Boolean state variables[1]. Action schemata represent both the transition and reward functions. Each action schemata a consists in a precondition, that is, a formula characterizing the states where a is applicable, and an effect. The effect describes the changes that may be applied to a state when taking action a. Importantly, PPDDL actions obey the *frame assumption*: variables not explicitely modified by an action remains the same after taking the action. PPDDL1.0 defines the following types of effects:

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] We omit functions, which represent numeric state-variables.

- **Simple Effects** specify the truth value update for a predicate p; (p) (resp. $(\texttt{not}\ p)$) indicates that p will be true (resp. false) after execution of the effect.
- **Update Effects** specify how the reward function is affected. They are either $(\texttt{increase}\ (\texttt{reward})\ r)$ or $(\texttt{decrease}\ (\texttt{reward})\ r)$ with obvious meaning.
- **Conditional Effects** of the form $(\texttt{when}\ \phi\ e)$, where ϕ is a formula over the domain predicates and e an effect, mean that e occurs only in states satifying ϕ.
- **Probabilistic Effects** of the form $(\texttt{probabilistic}\ p_1\ e_1\ \ldots\ p_k\ e_k)$ mean that effect e_i occurs with probability p_i. We write $(\texttt{prob}\ p_1\ e_1\ \ldots\ p_k\ e_k)$ for short. In case $\sum_i p_i < 1$, the empty effect implicitely occurs with probability $1 - \sum_i p_i$.
- **Conjunctive Effects** of the form $(\texttt{and}\ e_1\ \ldots\ e_k)$, make all e_i's occur synchronously. Effects e_i must be consistent together, *i.e.*, not contain (x) and $(\texttt{not}\ x)$ (whatever the combination of outcomes if e_i's are probabilistic).

A planning problem, on a given domain, defines a set of objects used to instanciate predicates and action schemata. In turn, instanciated predicates encode a set X of *propositional* state variables and instanciated action schemata define a set of actions A, each action a having precondition p_a and effect e_a defined over X. A problem may also define a formula Γ characterizing the set of goal states associated with a reward R_Γ. If the problem has no goal (like classical MDPs), we assume $\Gamma = \bot$. Goal-oriented problems can be easily dealt with in the MDP framework, by ensuring the following for any goal state $g \models \Gamma$: no action can be taken in g and for any horizon h, $V_h(g) = R_\Gamma$. A problem may also define an initial state, but since we focus on the more general policy construction problem, the initial state is ignored. As an example, Figure 1 presents the PPDDL description of the `move` action of the well-known COFFEE problem.

For t, ℓ two sets of literals, write $t \triangleright \ell$ for the set of literals matching t, and completed as in ℓ over other variables, that is, $t \triangleright \ell = t \cup (\ell \setminus \{p \mid \bar{p} \in t\})$; e.g., $xy \triangleright x\bar{y}\bar{z}t = xy\bar{z}t$. Given a state s (a full instanciation of states variables X), each effect e described by a PPDDL problem induces a probability distribution $D(e, s)$ over a set of pairs (t, r) (Younes and Littman 2004, Sec. 4.1). Each such pair consists in a "basic effet" t, that is, a consistent set of literals modified in state s to obtain next state $s' = t \triangleright s$, and an immediate reward r. Writing $(t, r) : p$ for "effect (t, r) occurs with probability p", $D(e, s)$ is recursively defined as follows:

- $D(x, s) = \{(x, 0) : 1\}$
- $D(\texttt{not}\ x, s) = \{(\bar{x}, 0) : 1\}$
- $D(\texttt{increase}\ (\texttt{reward})\ r, s) = \{(\emptyset, r) : 1\}$
- $D(\texttt{decrease}\ (\texttt{reward})\ r, s) = \{(\emptyset, -r) : 1\}$
- $D(\texttt{when}\ \phi\ e, s) = D(e, s)$ if $s \models \phi$ otherwise $\{(\emptyset, 0) : 1\}$
- $D(\texttt{prob}\ p_1\ e_1 \ldots, s) = \bigcup_i \{(t, r) : p_i p \mid (t, r) : p \in D(e_i, s)\}$
- $D(\texttt{and}\ , s) = \{(\emptyset, 0) : 1\}$
- $D(\texttt{and}\ e_1\ e_2 \ldots, s) = \{(t' \cup t, r' + r) : p'p \mid (t', r') : p' \in D(e_1, s), (t, r) : p \in D(\texttt{and}\ e_2\ \ldots, s)\}$

With this in hand, we can rewrite Equation 1 as:

$$Q^a_V(s) = \mathop{\mathbf{E}}_{(t,r) \sim D(e_a, s)} [r + \gamma V(t \triangleright s)]$$

Algebraic Decision Diagrams An *Algebraic Decision Diagram* (Bahar et al. 1997), or ADD for short, represents a function $\{0,1\}^n \to \mathbb{R}$ mapping the values of n Boolean variables to a real value. ADDs are Directed Acyclic Graphs, where each node is either a terminal node labelled by a real-valued constant, or an internal node labelled with some variable x_i with one subgraph for each value of x_i. ADDs are expressionnally equivalent to decision trees, but allow for a smaller representation since nodes share isomorphic subgraphs. Moreover, for any fixed variable ordering, each function admits a unique ADD representation.

ADDs come with efficient algorithms for many operations. We use the following notation for functions and their ADDs. Let $V_1, V_2 : \{0,1\}^n \to \mathbb{R}$, $r \in \mathbb{R}$, $s \in \{0,1\}^n$, ϕ a formula on $\{x_1, \ldots, x_n\}$ and t a conjunction of literals:

$$\begin{array}{rcl}
(V_1\ op\ V_2)(s) & = & V_1(s)\ op\ V_2(s) \text{ for } op \in \{+, \times, -\} \\
\max(V_1, V_2)(s) & = & \max(V_1(s), V_2(s)) \\
(V_1 + r)(s) & = & V_1(s) + r \\
(\phi \times V_1)(s) & = & V_1(s) \text{ if } s \models \phi, \quad 0 \text{ otherwise} \\
\text{ITE}(\phi, V_1, V_2)(s) & = & V_1(s) \text{ if } s \models \phi, \quad V_2(s) \text{ otherwise} \\
(V_1)_{[t]}(s) & = & V_1(t \triangleright s) \\
(\exists x_i V_1)(s) & = & (V_1)_{[x_i]}(s) + (V_1)_{[\bar{x}_i]}(s)
\end{array}$$

In the factored MDP litterature (see below), ADDs revealed to be candidate of choice for representing structured transitions matrices and value functions, since in practical problems many states have the same value, allowing memory savings and fast value function manipulation.

Related Work

Most work on computing policies for factored MDPs has focused on DBN representations. Such representations use conditional probability tables (CPTs), which give the probability of each x' being made true or false by each action, depending on the values of its parents in the DBN. PPDDL descriptions can be converted to DBNs (Younes and Littman 2004), and CPTs can be compactly represented using ADDs (Hoey et al. 1999; St-Aubin, Hoey, and Boutilier 2001; Feng and Hansen 2002; Feng, Hansen, and Zilberstein 2003; Guestrin et al. 2003). Then action regression is performed by simple ADD operations.

Though this representation is natural when variables are independent, correlated effects such as $(\texttt{prob}\ 0.5\ xy\ 0.5\ \bar{x}\bar{y})$ introduce dependencies between post-action values. The usual trick is to introduce auxiliary variables whose values encode which effect occurred, and deterministic dependencies from them to post-action variables (Younes and Littman 2004, Sec. 5), but naturally, this increases the size of diagrams manipulated. Another problem is independent conditions on the same variable, e.g., $(\texttt{and}\ (\texttt{when}\ x_1\ (\texttt{prob}\ p_1\ x))\ (\texttt{when}\ x_2\ (\texttt{prob}\ p_2\ x)))$, which make CPTs blow up if auxiliary variables are not

```
(:action move
        :effect (and (when (in-office) (probabilistic 0.9 (not (in-office))))
                     (when (not (in-office)) (probabilistic 0.9 (in-office)))
                     (when (and (raining) (not (has-umbrella)))
                          (probabilistic 0.9 (is-wet)
                                        0.1 (when (not (is-wet)) (increase (reward) 0.2))))
                     (when (or (not (raining)) (has-umbrella))
                          (when (not (is-wet)) (increase (reward) 0.2)))
                     (when (user-has-coffee) (increase (reward) 0.8))))
```

Figure 1: PPDDL descritpion of the `move` action of the COFFEE problem. The three first "when" clauses read: with probability 0.9 the agent will change location; concurrently, starting from a state where it rains and the agent has no umbrella, with probability 0.9 the agent will be wet in next state; otherwise, if it is not already wet, it will remain dry and receive a 0.2 reward.

used. Nevertheless, algorithms using such representations can be very efficient, both in the exact (Hoey et al. 1999) and approximative (St-Aubin, Hoey, and Boutilier 2001; Guestrin et al. 2003) cases. Some work has also been done at the relational level (Lang and Toussaint 2010).

A somehow intermediate representation between PPDDL and DBNs is Probabilistic STRIPS Operators (Kushmerick, Hanks, and Weld 1995). Under this representation, actions have mutually exclusive conditions, each associated with a probabilistic effect of the form ($\texttt{prob}\ p_1\ e_1 \dots p_k\ e_k$) where each e_i is a conjunction of literals. As far as we know, no work has focused on computing optimal MDP policies for such representations in the grounded case. Nevertheless, they are directly used by algorithms which work at the relational level (Boutilier and Sanner 2009).

When using ADDs, DBN representation can be exponentially more succinct than PPDDL. Conversely, because of nested effects, PPDDL may be exponentially more succinct than DBNs or PSOs. For more details about these issues, we refer the reader to (Boutilier, Dean, and Hanks 1999; Rintanen 2003; Younes and Littman 2004).

Updating Value Functions

We now come to our new approach for exact backup of value functions in MDPs, which we integrate in value iteration.

F-values The process of *action backup* refers to the task of computing the value Q_V^a from a value function V. We propose to perform this task incrementally with respect to a PPDDL description of actions, using an intermediate representation of values as *frameless* action-values. As other approaches, we use ADDs for efficient storage and manipulation of real-valued functions of Boolean variables.

Definition 1 *Let $V : S \to \mathbb{R}$ be a value function and e be an effect. The* frameless action-value *(F-value for short) of e with respect to V is the function $F_V^e : S \times 2^X \to \mathbb{R}$ which for $s \in S$ and $\ell' \in 2^X$, gives the expected reward of applying e in s and forcing the values of ℓ' on unaffected variables:*

$$F_V^e(s, \ell') = \mathop{\mathbf{E}}_{(t,r) \sim D(e,s)} [r + V(t \rhd \ell')]$$

Intuitively, F-values are just like usual action-values Q_V^a, but they make no frame assumption on the unmodified variables. As we will see, this allows to handle conjunctive effects in an incremental fashion.

Example 1 *Let x be a variable and $V(x) = 0, V(\bar{x}) = 1$. Then the F-value F_V^e of $e =$ ($\texttt{when}\ x$ ($\texttt{not}\ x$)) wrt V is given by $F_V^e(x, \cdot) = 1, F_V^e(\bar{x}, x') = 0, F_V^e(\bar{x}, \bar{x}') = 1$. In words, for $s = \bar{x}$, the F-value leaves the possibility that $\bar{x}$ persists, but also that it changes to x'. This is to be compared with the action-value $Q_V^e(\cdot) = 1$.*

Another example is given by the right branch of the F-value depicted on Figure 2 (d) for $e =$ ($\texttt{when}\ x$ ($\texttt{prob}\ 0.6\ x\ 0.4\ y$)): for s satisfying x and $\ell' = \bar{x}'\bar{y}'$, the expected value is 5.2. Indeed, in s, either:

- *effect x occurs, and y is not affected; hence the value of y' is taken from ℓ', resulting in state $x'\bar{y}'$, which has value 2 (Figure 2 upper left); this occurs with probability 0.6;*
- *or y occurs, resulting in state $\bar{x}'y'$ with value 10; this occurs with probability 0.4.*

Hence the F-value of e in s, with unaffected variables modified as in ℓ', is $0.6 \times 2 + 0.4 \times 10 = 5.2$.

An important feature of F-values is that they embed action-values. Indeed, when discounting and enforcing the frame assumption we get (see the proof in Theorem 2):

$$Q_V^a(s) = F_{\gamma V}^{e_a}(s, s) \qquad (2)$$

Example 2 *Considering Figure 2 (d) and $s = x\bar{y}$, $Q_V^e(s)$ is retrieved by evaluating the ADD in $x\bar{y}x'\bar{y}'$, yielding 3.2, which is indeed the expected value of e in s (probability 0.6 of setting x to true, resulting in state $x'\bar{y}'$ with value 2, and 0.4 of setting y to true, resulting in $x'y'$ with value 5).*

Backup rules Our rules take as input an F-value V' and an effect e described in PPDDL, and produce a new F-value $B_{V'}(e)$ which accounts for the application of e on V'. Since F-values are functions of $S \times 2^X$, we introduce a second set of *primed* variables $X' = \{x' \mid x \in X\}$ to represent them with ADDs. $B_{V'}(e)$ is recursively defined as follows:

R1.1 $B_{V'}(x) = V'_{[x']}$

R1.2 $B_{V'}(\texttt{not}\ x) = V'_{[\bar{x}']}$

R2.1 $B_{V'}(\texttt{increase (reward)}\ r) = V' + r$

R2.2 $B_{V'}(\texttt{decrease (reward)}\ r) = V' - r$

R3 $B_{V'}(\texttt{when}\ \phi\ e) = \text{ITE}(\phi, B_{V'}(e), V')$

R4 $B_{V'}(\texttt{prob}\ p_1\ e_1 \dots p_k\ e_k) = \sum_{i=1}^k p_i \times B_{V'}(e_i)$

R5.1 $B_{V'}(\texttt{and}) = V'$

R5.2 $B_{V'}(\texttt{and}\ e_1\ e_2 \dots e_k) = B_{B_{V'}(e_1)}(\texttt{and}\ e_2\ \dots\ e_k)$

Theorem 1 *Given a, V, let $V'(\cdot, s) = V(s)$ be the primed version of V. Then the backup rule $B_{V'}(e_a)$ computes $F_V^{e_a}$.*

Proof. We show by induction on the PPDDL description of e that for *any* function $W : S \times 2^X \mapsto \mathbb{R}$, $B_W(e)(s, \ell')$ is $\mathbf{E}_{(t,r)\in D(e,s)}\left[r + W(s, t \triangleright \ell')\right]$. The result follows because V' depends on its second argument only, hence $V'(s, t \triangleright \ell') = V(t \triangleright \ell')$. For brevity, we write $\mathbf{E}_{(t,r)}[\cdot]$ for $\mathbf{E}_{(t,r)\sim D(e,s)}[\cdot]$.

Let $e = x$. Then $B_W(e)(s, \ell') = W_{[x']}(s, \ell')$ (Rule 1.1). Because $D(e, s) = \{(x, 0) : 1\}$, this is $W(s, \{x'\} \triangleright \ell') = \mathbf{E}_{(t,r)}\left[r + W(s, t \triangleright \ell')\right]$, as desired. Case $(\texttt{not}\ x)$ is dual.

For $e = (\texttt{increase}\ (\texttt{reward})\ r)$, $B_W(e)(s, \ell')$ is $r + W(s, \ell')$. Since $D(e, s) = \{(\emptyset, r) : 1\}$, this is indeed $\mathbf{E}_{(t,r)}\left[r + W(s, t \triangleright \ell')\right]$. Case `decrease` is dual.

Now let $e = (\texttt{when}\ \phi\ e')$. If $s \models \phi$, then $B_W(e)(s, \ell')$ is $B_W(e')(s, \ell')$, which is correct by the induction hypothesis (IH). If $s \not\models \phi$, we get $B_W(e)(s, \ell') = W(s, \ell')$, which is $\mathbf{E}_{(t,r)\sim D(e,s)}\left[r + W(s, t \triangleright \ell')\right]$ since $D(e, s) = \{(\emptyset, 0) : 1\}$.

For $e = (\texttt{prob}\ p_1\ e_1 \dots p_k\ e_k)$, we have $B_W(e)(s, \ell') = \sum_i p_i B_W(e_i)(s, \ell')$. This is $\sum p_i\, \mathbf{E}_{(t_i,r_i)}\left[r_i + W(s, t_i \triangleright \ell')\right]$ by IH, and correctness follows by linearity of expectations.

The $(\texttt{and})$ case is obvious. Finally, for $e = (\texttt{and}\ e_1\ e_2)$ we have $B_W(e)(s, \ell') = B_{B_W(e_1)}(e_2)(s, \ell')$. This is $\mathbf{E}_{(t_2,r_2)}\left[r_2 + B_W(e_1)(s, t_2 \triangleright \ell')\right]$ by IH on e_2, and $\mathbf{E}_{(t_2,r_2)}\left[r_2 + \mathbf{E}_{(t_1,r_1)}\left[r_1 + W(s, t_1 \triangleright (t_2 \triangleright \ell'))\right]\right]$ by IH on e_1. Now because effects are consistent we have $t_1 \triangleright (t_2 \triangleright \ell') = (t_1 \cup t_2) \triangleright \ell'$, hence as desired we get $\mathbf{E}_{(t_1,r_1),(t_2,r_2)}\left[r_1 + r_2 + W(s, (t_1 \cup t_2) \triangleright \ell')\right]$. □

At this point, we described how to compute F-values F_V^e using ADDs. For a complete action backup, it remains to compute Q_V^a from $F_V^{e_a}$. Equation 2 suggests to enforce the frame assumption on the ADD $B_V(e_a)$ by computing the ADD $\texttt{Persist}(B_V(e_a))$, where `Persist` is defined by:

$$\texttt{Persist}(F) = \exists X' \left[x_1 \leftrightarrow x_1' \times \cdots \times x_n \leftrightarrow x_n' \times F\right] \quad (3)$$

Finally, given $V'(\cdot, s) = \gamma V(s)$ (discounted, primed version of value function V), and accounting for the precondition p_a and the goal Γ, we get:

$$Q_V^a = \mathrm{ITE}(p_a \times \neg\Gamma, \texttt{Persist}(B_{V'}(e_a)), -\infty)$$

Assigning $-\infty$ to states not satisfying p_a and to goal states, rules this action out when maximizing over actions. A complete value function update example is given on figure 2.

Putting it all together, we propose the algorithm RBAB (for "Rule-Based Action Backup"), whose pseudocode is given in Algorithm 1. RBAB performs symbolic value iteration to compute value functions and eventually a policy.

Theorem 2 *RBAB computes an ϵ-optimal infinite-horizon policy.*

Proof. We show by induction on h that after the h-th iteration of the main loop, W is the optimal h-steps-to-go value function V_h. Let a be an action. By Theorem 1 and the induction hypothesis, $B_{V'}(e_a)(s, \ell')$ is $\mathbf{E}_{(t,r)\sim D(e_a,s)}\left[r + \gamma V_{h-1}'(s, t \triangleright \ell')\right]$. Since `Persist` forces $x' \leftrightarrow x$ for all primed variables in $B_{V'}(e_a)$, we get $V_{h-1}'(s, t \triangleright \ell') = V_{h-1}(t \triangleright s)$. Moreover, `Persist` abstracts primed variables away, hence the function Q computed by the algorithm depends on unprimed variables only, so that when s satisfies $p_a \times \neg\Gamma$ we have $Q(s) = \mathbf{E}_{(t,r)\sim D(e_a,s)}\left[r + \gamma V_{h-1}(t \triangleright s)\right] = Q_{V_{h-1}}^a(s)$, as desired. Observing that the remainder of the algorithm is classical Value Iteration concludes the proof. □

Algorithm 1: Rule-Based Action Backup (RBAB)

```
$V \leftarrow \Gamma \times R_\Gamma$
repeat
    $W \leftarrow -\infty$;    $V' \leftarrow \gamma \times prime\_variables(V)$
    foreach action $a$ do
        $Q \leftarrow \texttt{ITE}(p_a \times \neg\Gamma, \texttt{Persist}(B_{V'}(e_a)), -\infty)$
        $W \leftarrow \max(W, Q)$
    $W \leftarrow \texttt{ITE}(\Gamma, R_\Gamma, W)$ // Goal states $\mapsto$ $R_\Gamma$
    converged $\leftarrow ||V - W||_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$
    $V \leftarrow W$
until converged
// Extract policy $\pi$
$W \leftarrow -\infty$;    $V' \leftarrow \gamma \times prime\_variables(V)$
$\pi \leftarrow 0$
foreach action $a_i$, $i = 1, \dots, |A|$ do
    $Q \leftarrow \texttt{ITE}(p_a \times \neg\Gamma, \texttt{Persist}(B_{V'}(e_a)), -\infty)$
    $G \leftarrow Q > W$//  $\forall s, G(s) \Leftrightarrow Q(s) > W(s)$
    $\pi \leftarrow \max(\pi, G \times i)$
    $W \leftarrow \texttt{ITE}(G, Q, W)$
return $\pi \times \neg\Gamma$ ; // Goal states $\mapsto$ action 0
```

Algorithm 2: Persist

```
Input: $N$, an ADD with primed and unprimed variables
Input: $A$, a set of literals to be forced (initially $\emptyset$)
if $N$ is a constant ADD then return $N$
$v \leftarrow var(N)$
// Force literals.
if $v \in A$ then return $\texttt{Persist}(Then(N), A \setminus \{v\})$
if $\bar{v} \in A$ then return $\texttt{Persist}(Else(N), A \setminus \{\bar{v}\})$
if $(N, A) \rightarrow R$ is in cache then  return $R$
$x \leftarrow$swap($v$)//  swap($v'$)=$v$;swap($v$)=$v'$
$T \leftarrow \texttt{Persist}(Then(N), A \cup \{x\})$
$E \leftarrow \texttt{Persist}(Else(N), A \cup \{\bar{x}\})$
// Merge over unprimed variable.
if $v$ is primed then $u \leftarrow x$ else $u \leftarrow v$
$R \leftarrow \texttt{ITE}(u, T, E)$
insert $(N, A) \rightarrow R$ into cache
return $R$
```

Optimizations A critical step of Algorithm 1 is the computation of `Persist`. While Equation 3 directly maps to ADD operations, this requires several traversals of the ADD. To circumvent this, we use the dedicated implementation given as Algorithm 2, which requires a single pass.

One major drawback to ADDs is their sensitivity to variable ordering. Indeed, the size of the diagrams may increase dramatically if a wrong ordering is choosen, resulting in poor performance. Reordering algorithms provided by ADD packages may take some time to find a good, let alone optimal, variable ordering which reduces diagrams sizes, hence they must be used with caution. Experiments have shown that forcing x, x' to stay adjacent and using approximate reordering at the end of each iteration provides the best results.

Finally, note that the backup of probabilistic effects may benefit from a dedicated (possibly n-ary) multiply-and-add operator, but we haven't evaluated this empirically.

Empirical Results

RBAB was implemented as a client for the MDPsim server, allowing us to use domains from the International Planning Competition (IPC) and compare our approach with SPUDD.

Along with MDPsim is shipped MTBDDclient, an implementation of SPUDD which first translates PPDDL to DBNs. Since MTBDDclient is intended to be a demonstration program, we optimized it in two ways: first, a policy is built only in a last iteration; second, we introduced variable reordering heuristics to enhance its performance further. We tested two versions of SPUDD: the one called "1 by 1" considers variables one at a time (backup and abstraction) at each iteration, and "matrix" precomputes the "complete action diagram" (Hoey et al. 1999, Sec. 4.2).

We empirically compared the running times of all three solvers on domains taken from the 5th and 6th IPC (probabilistic track). Obviously enough, since we focused on the problem of finding optimal control policies, the hardest instances of the domains were out of reach for the solvers. We kept the problems where at least one solver could produce an ϵ-optimal discounted infinite-horizon policy with $\epsilon = 0.1$ and $\gamma = 0.9$ in less than one hour.

Figure 3 shows the results on domains *pitchcatch, rectange-tireworld, search-and-rescue, drive*, and *drive-unrolled*. Clearly, these results show the complementarity of the different solvers. Indeed, no algorithm is always better (or worse, for that matter) than any other.

On the *pitchcatch* domain, RBAB behaves better than "matrix", but "1 by 1" is still better. On the other hand, on the *rectangle-tireworld* domain RBAB outperforms both. The most conclusive results are on *search-and-rescue*, where RBAB appears to be exponentially faster than both. Moreover this problem is one of the two we tested (along with *pitchcatch*) for which the number of non exclusive conditions under an `and` statement, was greater than 1 and increasing linearly with the instance size. This suggests that this is a good indicator to expect good RBAB performance.

The results for the *drive* domains (*drive, drive-unrolled* and *drive-unrolled2*) are very interesting. Indeed, all three domains express the same traffic intersection problem, but in the "unrolled" version, actions are decomposed into a set of smaller, equivalent ones. On the *drive* instances RBAB is the worst algorithm, but its behavior clearly improves on the "unrolled" version, exploiting a smaller PPDDL description of actions. Domain *drive-unrolled2* is another variant which we designed, where we merged almost identical actions (look_at_light_east with look_at_light_west and look_at_light_north with look_at_light_south) into more general, but equivalent ones. The effects of these actions were also simplified by moving up redundant effects in the effect hierarchy. The resulting domain exploits as much as possible the succinctness of PPDDL. As depicted in Figure 3, RBAB outperforms other solvers on this variant. This confirms that our approach is able to exploit the succinctness of the problem specification in PPDDL.

For exhaustivity, the table on Figure 3 summarizes the results for domains where only a few instances were solved. Interestingly, on these domains the "1 by 1" solver reaches at least as many instances as the others, whereas it was almost always outperformed in the other domains. Apart from this, each solver clearly outperforms both others on one of these hard domains.

Policy revision using F-values

So far, F-values have been used as a tool for computing usual action-values. We now describe how they can be used for some forms of policy revision.

By "revision", we mean here the general problem of computing the new value of an action a, given its "old" value and a modification of its description. More precisely, we write a for an action, F_V^a for its F-value with respect to some value function V, and a' for another action intended to be a modified version of a. Our general goal is to compute $F_V^{a'}$ given F_V^a and the PPDDL descriptions of a, a'.

For instance, assuming V is the optimal infinite-horizon value function for the problem at hand, with π an associated optimal policy, we have that F_V^a is the optimal (frameless) action-value function for a. Then the revised function $F_V^{a'}$ gives the expected value of performing a', then following the old policy π from the resulting state on. In equations, for any state s, $F_V^{a'}(s, s) = \mathbf{E}_{(t,r)\sim D(e_{a'},s)}\left[r + V(t \triangleright s))\right]$.

Notably, this revision problem arises in model-based reinforcement learning (RL), where action descriptions and rewards are incrementally learnt by the agent, and hence the corresponding action values need to be constantly revised. In particular, Algorithms RTDP-RMAX and RTDP-IE (Strehl, Li, and Littman 2006) perform a single backup step, as above, when rewards and transitions are updated. For instance, in an RL scenario, after it has gathered some further experience an agent may want to revise the relative probabilities of effects x and y in Figure 2, from 0.6/0.4 to 0.8/0.2, resulting in the diagram depicted on Figure 2 (j).

A natural way to solve revision problems is to store the value function V and, when revision is required, to compute $Q_V^{a'}$ from scratch. We propose instead to store the F-value of a (recall that at any time, Q_V^a can be retrieved efficiently by $Q_V^a(s) = F_V^a(s, s)$). As we show next, using the richness of information of F-values, we can save a substantial amount of computation for revision. The counterpart is increased memory needs, since F-values are bigger than usual action-values. How this tradeoff speed/memory is ideally resolved depends on the application at hand.

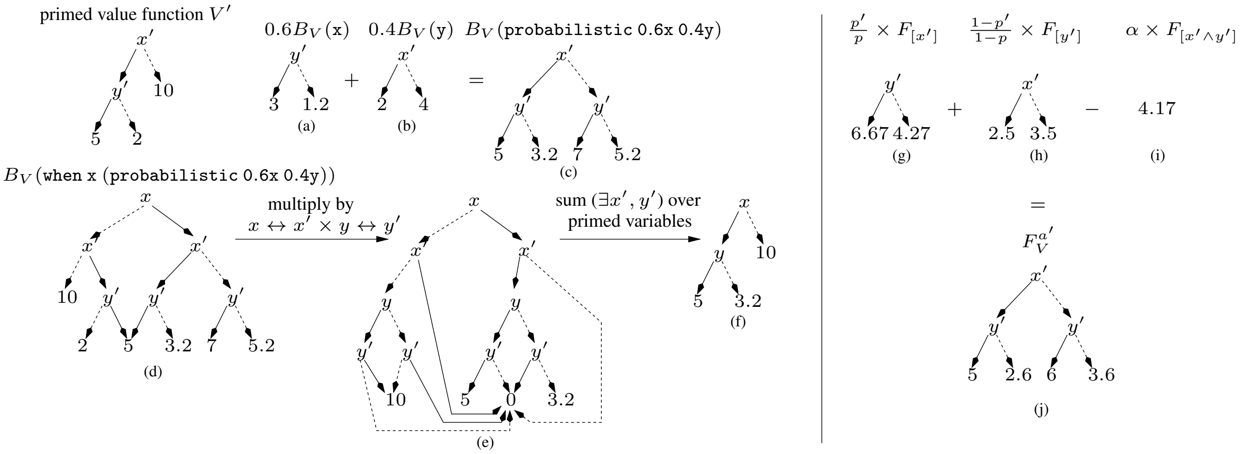
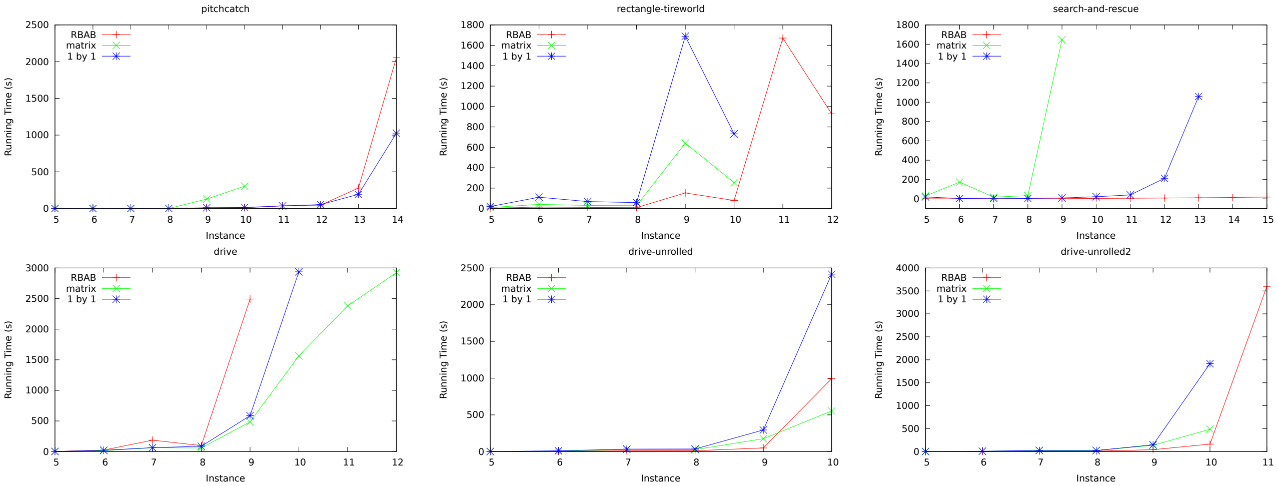

primed value function V'
$0.6B_V(\mathtt{x})$
$0.4B_V(\mathtt{y})$
$B_V(\mathtt{probabilistic\ 0.6x\ 0.4y})$
(a)
(b)
(c)
$B_V(\mathtt{when\ x\ (probabilistic\ 0.6x\ 0.4y)})$
multiply by $x \leftrightarrow x' \times y \leftrightarrow y'$
sum $(\exists x', y')$ over primed variables
(d)
(e)
(f)
$\frac{p'}{p} \times F_{[x']}$
$\frac{1-p'}{1-p} \times F_{[y']}$
$\alpha \times F_{[x' \wedge y']}$
6.67
4.27
2.5
3.5
4.17
(g)
(h)
(i)
$F_V^{a'}$
(j)

Figure 2: (Left) Application of the backup rules on a value function V for the effect (when x (prob 0.6 x 0.4 y)). Diagrams (a) and (b) present the backup of simple effects (weighted by their probabilities), (c) is the diagram for the probabilistic effect and (d) for the conditional one. Diagram (e) is the intermediate diagram of Equation 3 and (f) is the final action-value function. (Right) Revision of the F-value for the prob clause with $0.8/0.2$ instead of $0.6/0.4$.



Other reachable domains

domain	#instances solved			best solver	worst solver
	RBAB	matrix	1by1		
schedule	3	3	3	RBAB	matrix
sysAdmin-SLP	4	4	5	1by1	RBAB
triangle-tireworld	1	2	2	matrix	RBAB

Figure 3: Solvers running times on International Planning Competition benchmarks

Adding an effect The first, simplest case is when the revision of a consists in adding an effect e, namely, when $e_{a'} = (\mathtt{and}\ e_a\ e)$ (we assume that e is consistent with a). Then it follows directly from Theorem 1 that $F_V^{a'}$ can be computed from F_V^a simply as $B_{F_V^a}(e)$.

Observe that no knowledge of V is needed, and that only the "difference" between the revised and the old action has to be treated. Moreover, we wish to emphasize that F-values are necessary for this to be possible, in the sense that usual action-values do not carry enough information.

Example 3 *As shown in Example 1, for $a = (\mathtt{when}\ x\ (\mathtt{not}\ x))$ and $V(x) = 0, V(\bar{x}) = 1$, we have $Q_V^a(\cdot) = 1$. Hence when revising a into $a' = (\mathtt{and}\ (\mathtt{when}\ x\ (\mathtt{not}\ x))\ (\mathtt{when}\ (\mathtt{not}\ x)\ x))$, usual action values alone cannot distinguish between cases $V(x) = 0$ as above, and, say, $V(x) = 10$, whereas revision should yield $Q_V^{a'}(\bar{x}) = 0$ in the former case, but $Q_V^{a'}(\bar{x}) = 10$ in the latter.*

Revising rewards/costs It is straightforward to see that revising the reward or cost of an action in a group of states S can be handled by simply adding an effect of the form (`when` ϕ (`increase` (`reward`) r)), where ϕ characterizes S and r is the revision amount, and revising as in the previous paragraph. We however wish to note that this particular case can be handled as well using usual action-values only (in both cases, simply add $\phi \times r$ to the old value function).

Revising probabilities We now consider the case when the probabilities of some effects are revised, namely, when (`prob` p $e \ldots$) is revised into (`prob` p' $e \ldots$). We restrict ourselves to the case where this clause occurs at the highest level in the description of the action, namely, when a is of the form (`and` e_1 $\ldots e_k$ (`when` ϕ (`prob` p $e \ldots$))) (wlog, since `and` is commutative). We also assume that e is a conjunction of simple effects (literals), as in PSO representations. However, we place no restriction on other e_i's.

For clarity, we start with the case when there is only one probabilistic effect, namely, when we want to revise $a =$ (`and` $e_1 \ldots e_k$ (`when` ϕ (`prob` p e))) to $a' =$ (`and` $e_1 \ldots e_k$ (`when` ϕ (`prob` p' e))). The next proposition shows that again, revision can be performed with the sole knowledge of F_V^a. We omit the proof because it is a special case of Proposition 2 (with $f = \emptyset$), but the intuition in this particular case is that revising p to p' amounts to add the effect (`when` ϕ (`prob` δ e)) to a, because this means for e *not* to occur with probability $(1-p)(1-\delta) = (1-p')$ (Proposition 2 shows correctness for negative δ).

Proposition 1 *Let a be an action of the form* (`and` $e_1 \ldots e_k$ (`when` ϕ (`prob` p e)))*, where e is a conjunction of simple effects and $p \neq 1$. Let V be a value function, let $p' \neq p$, and let a' be the action* (`and` $e_1 \ldots e_k$ (`when` ϕ (`prob` p' e)))*. Then, with $\delta = 1 - \frac{1-p'}{1-p}$, $F_V^{a'}$ can be computed from F_V^a as*

$$\text{ITE}(\phi, \quad \delta \times B_{F_V^a}(e) + (1-\delta)F_V^a, \quad F_V^a)$$

We now turn to the more complex case where a clause of the form (`prob` p e $(1-p)$ f) is revised. Again, we assume that it occurs at the highest level of a and that e, f are conjunctions of simple effects. Moreover, we assume that e, f are mutually consistent (contain no opposite literals).

So assume p is revised to p'. We cannot simply "add" new clauses, say (`prob` δ_e e) and (`prob` δ_f f), as in the case of a single outcome, since this would incorrectly introduce possible outcomes with e and f both occurring. Once again however, it turns out that the F-value of a can be revised without recomputing it from scratch.

Proposition 2 *Let a be an action of the form* (`and` $e_1 \ldots$ e_k (`when` ϕ (`prob` p e $(1-p)$ f)))*, where e, f are mutually consistent conjunctions of simple effects and $p, 1-p \neq 0$. Let V be a value function, let $p' \neq p$, and let a' be the action* (`and` $e_1 \ldots e_k$ (`when` ϕ (`prob` p' e $(1-p')$ f)))*. Then $F_V^{a'}$ can be computed from F_V^a as*

$$\text{ITE}(\phi, \quad \frac{p'}{p} \times F_{[e']} + \frac{1-p'}{1-p} \times F_{[f']} - \alpha \times F_{[e' \cup f']}, \quad F_V^a)$$

with $\alpha = \frac{p'(1-p)}{p} + \frac{p(1-p')}{1-p}$, $F_{[e']} = (F_V^a)_{[e']}$ with e' the primed version of e, and similarly for $F_{[f']}, F_{[e' \cup f']}$.

Proof. Write W for $B_{V'}$(`and` $\mathtt{e_1} \ldots \mathtt{e_k}$). By Theorem 1 and because e, f are conjunctions of simple effects, the correct value is given by

$$F_V^{a'} = \text{ITE}(\phi, \quad p' \times W_{[e']} + (1-p') \times W_{[f']}, \quad W) \tag{4}$$

Moreover, for the same reasons we have

$$F_V^a = \text{ITE}(\phi, \quad p \times W_{[e']} + (1-p) \times W_{[f']}, \quad W) \tag{5}$$

Clearly the result holds for states not satisfying ϕ. Now for states satisfying ϕ we claim that

$$\frac{p'}{p} \times F_{[e']} + \frac{1-p'}{1-p} \times F_{[f']} - \alpha \times F_{[e' \cup f']} \tag{6}$$

is the correct function.

Indeed, observing $(W_{[e']})_{[e']} = W_{[e']}$, $(W_{[f']})_{[f']} = W_{[f']}$, and $(W_{[e']})_{[f']} = (W_{[f']})_{[e']} = W_{[e' \cup f']}$ (since e', f' are consistent together), we get from Equation 5:

$$\begin{aligned}
\frac{p'}{p} \times F_{[e']} &= p' \times W_{[e']} + \frac{p'(1-p)}{p} \times W_{[e' \cup f']} \\
\frac{1-p'}{1-p} \times F_{[f']} &= \frac{p(1-p')}{(1-p)} \times W_{[e' \cup f']} + (1-p') \times W_{[f']} \\
\alpha \times F_{[e' \cup f']} &= \left(\frac{p'(1-p)}{p} + \frac{p(1-p')}{(1-p)} \right) W_{[e' \cup f']}
\end{aligned}$$

Substituting these values in Equation 6 shows equivalence with the "then" argument in Equation 4, as desired. □

When there are more than 2 effects or effects are not consistent together, it can be shown that probabilities still can be revised from old F-values alone, albeit with a potential combinatorial explosion (depending on how effects interact). For space reasons however, we do not elaborate on this here.

It can also be easily seen that this contruction extends to any *deterministic* effect e (possibly with nesting), albeit with corrective terms proportional to the reward associated to e (which is 0 for conjunctions of simple effects).

The construction of Proposition 2 is illustrated on the right part of Figure 2 for revising (`prob` 0.6 x 0.4 y) to (`prob` 0.8 x 0.2 y).

Using F-values for control As we already insisted, F-values represent the possible post-action combinations of variables, taking into account explicit effects, but not enforcing any frame assumption on other variables. We now briefly discuss an application to control.

Assume you want to embark a policy for some MDP, and the evolution of some variables is stochastic, independent of your actions, but might be known one step in advance. To make things concrete, assume you know the probability for rain to fall (*rain*) on each day, but you also have the opportunity, at execution time, to know whether or not it will rain in the next state (by watching weather forecast, say).

Precisely, assume that you have two deterministic actions, namely taking an umbrella (`take-umb`), with cost 1 in all states, and doing nothing (`noop`), with cost 0 in all states. Assume moreover that in any state, rain falls with probability 0.2, and does not with probability 0.8, independently of whether it was falling before. Finally, in any state, if it rains and you do not carry your umbrella ($\overline{umb}$) you incur a penalty of -10, while in other cases you incur no penalty.

Now let $s = \overline{rain}\ \overline{umb}$. Under the usual semantics, action `take-umb` has a (1 step-to-go) value of $-1 + 0.2 \times 0 + 0.8 \times 0 = -1$ in s, and `noop` has a value of $0 + 0.2 \times -10 + 0.8 \times 0 = -5$. Hence the best action in s is to take the umbrella.

Now assume that at execution time, in state s you learn that it will not rain in the next state (say, tomorrow), that is, you learn that $\overline{rain'}$ is false. Then action-values will not tell you what the best action is with this new information, since they do not make explicit the role of rain in expected values. Constrastingly, assume you have F-values available, namely, values F_V^a ($a \in \{$`take-umb`, `noop`$\}$), where V is an optimal value function and F_V^a is computed wrt V without taking into account the probability of rain. Then you can simply evaluate them in s with $\ell' = \overline{rain} \triangleright s$. This will tell you exactly the expected value of actions *knowing that it will not rain tomorrow* (and using probabilities $0.2/0.8$ from tomorrow on). Precisely, in this case, the F-value for `take-umb` will still be -1, but that for `noop` will become 0, making `noop` now the best action.

We insist that F-values embed action-values. In this case, if you miss the weather forecast at execution time, you can simply add the effect (`prob` 0.2 *rain* 0.8 $\overline{rain}$) to both actions, update their F-values, and retrieve the action-values.

Another case where such control is particularly relevant is in multi-agent systems. Here you may know at execution time whether some other agents will help you by setting the values of some variables which you do not control. We leave a detailed investigation of such applications to multi-agent systems and cooperation for future work, but we think that they are very promising.

As a final note, we remark that this problem can alternatively be solved by introducing specific fluents with an epistemic flavor, say, two fluents coding whether you know the forecast, and what it is in case you know. Then action descriptions can be conditional on their values, and here usual action-values are enough. Nevertheless, it is clear that this makes the state space blow up, in particular if there are many such "short-term predictable" fluents.

Conclusion

We presented RBAB, an algorithm for computing optimal policies for MDPs described in PPDDL. To that end, we introduced the notion of a frameless action-value, and showed that using them for action backup in a value iteration scheme yields an algorithm which is complementary to previous approaches, and interesting in particular for correlated action effects and compact PPDDL descriptions. We also discussed some promising applications of this notion to some policy revision and control problems.

Our future work will focus on goal-oriented problems, especially using reachability analysis for computing partial policies over the set of reachable states, as in the LAO* family of algorithms. Also promising is the adaptation of our techniques to PPDDL at the relational level, for which there has been a lot of excitement recently (Boutilier and Sanner 2009; Lang and Toussaint 2010). To that end we plan to use First-Order ADDs (Wang, Joshi, and Khardon 2008). On the short term, we also plan to enrich our approach with the use of Affine ADDs (Sanner and McAllester 2005), which we believe will improve the behaviour of our algorithm just as it does for SPUDD and other solvers.

References

Bahar, R.; Frohm, E.; Gaona, C.; Hachtel, G.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebric decision diagrams and their applications. *Formal methods in system design* 10:171–206.

Boutilier, C., and Sanner, S. 2009. Practical Solution Techniques for First-Order MDPs. *Artificial Intelligence* 173:748–788.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *J. Artificial Intelligence Research* 11:1–94.

Feng, Z., and Hansen, E. 2002. Symbolic heuristic search for factored Markov decision processes. *Proc. AAAI 2002* 455–460.

Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *Proc. UAI 2003*, 209–216.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Effcient Solution Algorithms for Factored MDPs. *J. Artificial Intelligence Research* 19:399–468.

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

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–286.

Lang, T., and Toussaint, M. 2010. Planning with Noisy Probabilistic Relational Rules. *J. Artificial Intelligence Research* 39:1–49.

Rintanen, J. 2003. Expressive Equivalence of Formalism for Planning with Sensing. In *Proc. ICAPS 2003*, 185–194.

Sanner, S., and McAllester, D. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proc. IJCAI 2005*, 1384–1390.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2001. APRICODD: Approximate policy construction using decision diagrams. *NIPS 2000* 1089–1095.

Strehl, A. L.; Li, L.; and Littman, M. L. 2006. Incremental model-based learners with formal learning-time guarantees. In *Proc. UAI 2006*.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational MDPs. *J. Artificial Intelligence Research* 31:431–472.

Younes, H., and Littman, M. 2004. PPDDL1.0: An Extension to PDDL for Expressiong Planning Domains with Probabilistic Effects. *Tech. Rep. CMU-CS-04-167*.