Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Planning in Factored Action Spaces with Symbolic Dynamic Programming

Aswin Raghavan, Saket Joshi, Alan Fern, Prasad Tadepalli[a], Roni Khardon[b]
[a]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA
[b]Department of Computer Science, Tufts University, Medford, MA 02155, USA

Abstract

We consider symbolic dynamic programming (SDP) for solving Markov Decision Processes (MDP) with factored state and action spaces, where both states and actions are described by sets of discrete variables. Prior work on SDP has considered only the case of factored states and ignored structure in the action space, causing them to scale poorly in terms of the number of action variables. Our main contribution is to present the first SDP-based planning algorithm for leveraging both state and action space structure in order to compute compactly represented value functions and policies. Since our new algorithm can potentially require more space than when action structure is ignored, our second contribution is to describe an approach for smoothly trading-off space versus time via recursive conditioning. Finally, our third contribution is to introduce a novel SDP approximation that often significantly reduces planning time with little loss in quality by exploiting action structure in weakly coupled MDPs. We present empirical results in three domains with factored action spaces that show that our algorithms scale much better with the number of action variables as compared to state-of-the-art SDP algorithms.

1 Introduction

Many planning domains involve exponentially large action spaces, where actions can be described in terms of joint assignments to action variables. This can occur when agents have multiple actuators that can act simultaneously, e.g., a robot that navigates and operates equipment in parallel, or when a single actuator inherently decomposes into component factors, e.g. playing a set of three cards. This raises the need for planning algorithms that can leverage the structure of such factored actions in order to scale better than treating each of the exponentially many joint actions as atomic. Our contribution is to address this issue by introducing the first symbolic dynamic programming (SDP) approach for planning in MDPs with factored action spaces.

SDP planners attempt to uncover regularities in a planning domain in order to compute a compact representation (e.g. using decision diagrams) of the optimal value function and policy. Such planners have been shown to be competitive in calculating exact solutions to a variety of large MDPs. However, all previous SDP planners (e.g. (Hoey et al. 1999; Sanner, Uther, and Delgado 2010)) have focused on the case of factored states and have assumed a small set of atomic actions. Thus, these algorithms ignore any structure in the action space, causing both planning and execution times to scale at least linearly in the number of actions, which is exponential in the number of action factors.

The bottleneck for existing SDP planners applied to factored actions is that they work by iteratively computing the result of decision-theoretic regression (DTR) over each joint action. One of our main contributions is the factored-action regression (FAR) algorithm, which exploits a compact factored-action MDP representation in order to compute value functions in the form of algebraic decision diagrams over state and action variables. The result is a symbolic value iteration algorithm for factored states and actions.

Our factored action regression method comes at the cost of potentially higher memory usage compared to prior methods, but with the benefit of potentially much faster running times. Thus, our second contribution is the memory-bounded factored-action regression (MBFAR) algorithm, which is parameterized by a memory bound, and adaptively performs regression over subsets of actions, exploiting the structure within each subset. MBFAR provides a space-time trade-off, where at one extreme (minimal space) we get standard SDP methods, and at the other extreme (unbounded space) we get full factored regression.

Finally, to further improve efficiency, our third contribution is an approximation called "sequential hindsight", that can exploit the action structure in weakly coupled MDPs and significantly improves planning time. Our approximation takes advantage of the factorization over actions and employs hindsight sequentially over the outcomes of actions chosen by other subMDPs. The approximation is exact when the MDP can be decomposed into fully independent subMDPs. Our evaluation in three factored-action domains shows that FAR and MBFAR can be significantly faster than current approaches while producing exactly the same solution, and that the policies produced by the sequential hindsight heuristic perform near optimally.

2 Related Work

There are several approaches addressing factored action MDPs, but unlike our work most of these works do not attempt to compute an exact optimal policy or value function over all states. For example, prior work for goal-based problems with specified initial states (Little and Thiebaux 2006; Younes and Simmons 2004), does not deal with arbitrary rewards, and does not compute the policy over all states, or guarantee optimality. The work of Mausam and Weld (2004) extends to general rewards but still assumes an initial state and does not compute a policy over the entire space or guarantee optimality. The approach of Guestrin, Koller, and Parr (2001) considers general MDPs and computes policies over the entire space, but is based on function approximation relative to a set of hand-engineered features, which requires additional knowledge and does not guarantee optimality.

Our work is most closely related to work on model-minimization for factored action spaces (Kim and Dean 2002). Like our work, that approach leverages decision diagrams in order to compactly represent policies and value functions, but in a very different way. The approach preprocesses the MDP description in order to compute a "homogeneous" partition of the state-action space that makes all distinctions necessary for representing the value function of any policy. These partitions are then provided to an explicit MDP solver to find a policy. In contrast, SDP interleaves partitioning with steps of planning (i.e., regression) and thus only needs to make distinctions necessary for the value functions encountered during planning. For this reason, the model-minimization approach, by its very nature, is less efficient than SDP in general (Givan, Dean, and Greig 2003).

3 Background

3.1 Factored Markov Decision Processes

MDPs (Puterman 1994) and factored MDPs (Boutilier, Dean, and Hanks 1995) have been used successfully to solve sequential decision problems under uncertainty. An MDP is a tuple $(\mathbb{S}, \mathbb{A}, T, R, \gamma)$ where $\mathbb{S}$ is a finite state-space, $\mathbb{A}$ is a finite action space, $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0, 1]$ denotes the transition function such that $T(s, a, s') = Pr(s'|s, a)$, $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ denotes the immediate reward of taking action a in state s, and the parameter $\gamma \leq 1$ is used to discount future rewards.

A policy $\pi : \mathbb{S} \to \mathbb{A}$ indicates the action to choose in a state. The value function $V^\pi : \mathbb{S} \to \mathbb{R}$ is the expected discounted accumulated reward $E[\sum_{i=0}^{n} \gamma^i r(s_i, \pi(s_i)) \mid \pi]$ where s_i is the i'th state visited when following π. An optimal policy π^* is a policy that maximizes the value for all states simultaneously. The action-value function $Q^\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ indicates the expected reward accumulated by executing action a in state s and following policy π thereafter. For every MDP, there is a deterministic optimal policy π^* and unique optimal value function V^* such that $V^*(s) = \max_a Q^*(s, a)$ and $\pi^*(s) = \arg\max_a Q^*(s, a)$. The Value Iteration (VI) algorithm identifies V^* and π^* by the fixed point of iteration:

$$V_{n+1}(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} Pr(s'|s, a) V_n(s') \right]. \quad (1)$$

In a factored MDP, the state space $\mathbb{S}$ is specified by a finite set of binary variables $X = (X_1, \ldots, X_l)$ so that $|\mathbb{S}| = 2^l$. In contrast with previous work on SDP where $|\mathbb{A}|$ is assumed to be small, in this work the action space $\mathbb{A}$ is specified by a finite number of binary variables $A_1, \ldots, A_m$ and $|\mathbb{A}| = 2^m$. In what follows we use $a \in \mathbb{A}$ to represent a ground action where action variables $A_1, \ldots, A_m$ are instantiated to particular binary values. T is specified using a Dynamic Bayesian Network (DBN) or Influence Diagram (ID) representation for each action such that the next state variables X' depend on a small subset of the current state variables X denoted by parents(X'), and hence the model can be described compactly. In addition, as in previous work, we assume that there are no "synchronic arcs", i.e. the X' variables are conditionally independent and $Pr(s'|s, a) = \prod_i Pr(X'_i|\text{parents}(X'_i))$ and parents$(X'_i) \subseteq X$. This assumption is easy to remove to handle the general case.

The conditional probability tables (CPTs) of the DBN or ID are captured using a structured representation, such as a decision tree or algebraic decision diagrams (ADDs) (Bahar et al. 1997). Figure 1 shows an example influence diagram for an MDP in the SysAdmin domain (Section 7) with factored actions. The influence diagram captures the fact that the computers `c1`, `c2` and `c3` are arranged in a directed ring so that the status of each computer is influenced by its reboot action and the status of its predecessor in the ring. The right part of Figure 1 shows the ADD that implements the CPT of $running'(c1)$ representing the probability that $running'(c1)$ is true after the current action.

3.2 Symbolic Dynamic Programming

The VI algorithm specified by Eq (1) requires that the value of every state be updated in every iteration and this is prohibitive with 2^l states. SDP (Boutilier, Dean, and Hanks 1995) uses state aggregation to avoid this cost. The structured canonical representation of CPTs and reward makes it possible to perform value function updates efficiently by employing symbolic operations. Consider arbitrary functions over a set of propositional variables $X_1, \ldots, X_n$. A binary operation between two functions (denoted by $f_1 \textcircled{\text{op}} f_2$ or $f_1 \boxed{\text{op}} f_2$) is defined as the pointwise application of the operator op on the two functions. For example, $f_1 \oplus f_2 = f$ such that for all $\mathbf{z} \in \mathbb{R}^n$, $f(\mathbf{z}) = f_1(\mathbf{z}) + f_2(\mathbf{z})$, where $\mathbf{z}$ is an instantiation of $X_1, \ldots, X_n$. The restriction of f to a particular value $x_i \in \{0, 1\}$ of the variable X_i, fixes the value $X_i = x_i$, and removes X_i from the domain of f. We denote this by $f^{X_i = x_i}$. For example $(x + y)^{x=0} = y$ and $(x + y)^{x=1} = 1 + y$. Similarly, a marginalization operation (e.g., using sum or max) is defined as $\textcircled{op}_x f = op_{x_i \in \{0,1\}} f^{x=x_i}$. For example $\textcircled{\Sigma}_x (x + y) = y + (1 + y) = 1 + 2y$. These operations suffice so that dynamic programming operations over all states can be performed efficiently by specifying them as symbolic operations. The SPUDD algorithm is described in Figure (2a) where the main loop implements Eq (1) over the factored space. This algorithm implements the following equations

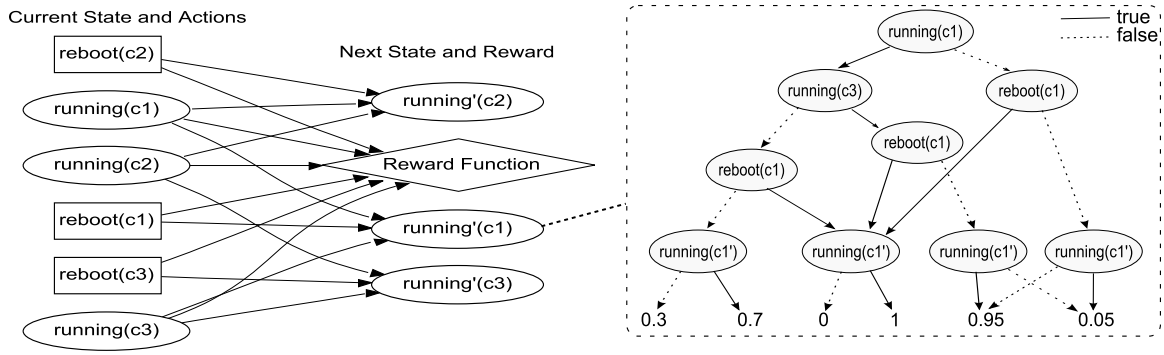

Figure 1: Left: Influence Diagram for an unidirectional ring with three computers in the SysAdmin domain; Right: CPT for $running'(c1)$ as an ADD.

$$Q^a_{n+1} = R^a \oplus \gamma \Big(\sum\Big)_{X_1'} Pr^a(X_1^{'} \mid X) \cdots \Big(\sum\Big)_{X_l'} Pr^a(X_l^{'} \mid X) \otimes (V'_n)$$
$$V_{n+1} = \max_a \{Q^a_{n+1}\}.$$

In these equations V'_n is the value function V_n expressed over next state variables X', $Pr^a(X'_i|X)$ specifies the probabilistic dependence of X'_i on X under a, and the summation leaves us with a value function over the current state variables X. Note that R^a and Pr^a represent the immediate reward and transition dynamics of the ground action a, and Q^a represents the Q-value of action a as a function of the state. The equations use the conditional independence of X' variables to push the sum over next state variables into the product. Partitions of the state space with similar values can be represented compactly and they capture state aggregation as in (Givan, Dean, and Greig 2003).

The current standard SDP planner, SPUDD (Hoey et al. 1999), uses ADDs as the representation. This has been recently extended by using Affine ADDs (AADDs) to compactly represent additive and multiplicative structure (Sanner, Uther, and Delgado 2010). The algorithms we present are independent of the representation used and hence the details of ADDs and AADDs are omitted here.

To conclude this section, we note that the symbolic representation of the value function allows us to represent constraints over state variables, and over state and action variables. Constraints can be applied symbolically by simply masking the value function to have a large negative value for states that violate a constraint.

4 Factored Action Regression

As mentioned in the previous section, while SPUDD takes advantage of structure in the state space, it enumerates the ground actions a, leading to 2^m iterations of regression. We address this high time complexity by modifying the representation of the domain dynamics which in turn allows for faster value iteration. Instead of specifying the transition model for each ground action separately as in SPUDD, we specify the transition model jointly as a function $Pr(X'|X, A)$ that depends on state variables $\{X_1, \ldots, X_n\}$ and action variables $\{A_1, \ldots, A_m\}$. If the action space has structure then this yields some space saving even in model specification. More importantly it allows us to perform VI in a factored manner over actions as well. Our factored-action regression (FAR) algorithm is presented in Figure (2b). In the algorithm, lines 3-7 include representations of V and Q that explicitly refer to action variables, and lines 8 and 9 explicitly marginalize action variables one at a time and in this way compute the optimal policy and value function. Thus we have replaced the expensive maximization step of SPUDD with a faster step of variable elimination. A side effect of factored action regression is that the Conditional Probability Tables (CPTs) of exogenous events are not repeated across action models. Expectation over exogenous next state variables is computed only once in FAR, whereas SPUDD recomputes it for each action. This yields additional savings in run time.

The explicit use of action variables allows us to take advantage of structure in state-action space and leads to efficiency. On the other hand, the intermediate diagrams capturing the Q-function depend on all actions simultaneously and can be more complex and require more space. Basically, we have traded the exponential time complexity of SPUDD for higher memory usage. This cost, too, can become prohibitive. In the next section we show how one can strike a more refined trade-off between SPUDD and FAR.

5 Memory-Bounded FAR

SPUDD and FAR are two extremes of handling factored actions in SDP - we either enumerate all actions or none at all, yielding extreme points w.r.t. time and space complexity. Roughly speaking we expect FAR to be faster unless it exceeds the space available on the computer. In such cases we can obtain a more refined trade-off by controlling the space explicitly. We develop this idea by analogy to recursive conditioning in Bayesian Networks (BN) (Darwiche 2001). There, a set of cutset variables are chosen and instantiated in all possible ways, giving a set of simpler BNs. Each simpler BN can be solved exactly and the results can be combined. The total time cost is exponential in the size of the cutset. In our context, the relevant variables are action variables and they need not form a cutset of any graph. The variables are grounded in all possible ways, to form restrictions of the value function for the corresponding action choices. The remaining variables are handled symbolically as in FAR. The grounded restrictions of the value function are maximized

Require: DBN action representation Pr^a, R^a for all actions $a \in \mathbb{A}$
$V' \leftarrow$Swap each X_i variable in V_n with X_i'
for each action $a \in \mathbb{A}$ **do**
3: $Q^a \leftarrow$Apply constraints to V' under a
for each X_i' **do**
$Q^a \leftarrow Q^a \otimes Pr^a(X_i'|X)$
6: $Q^a \leftarrow \sum_{X_i'} Q^a$
end for
$Q^a \leftarrow R^a \oplus \gamma Q^a$
9: $V_{n+1} \leftarrow \max(V_{n+1}, Q^a)$
update π_{n+1} to a in states where $Q^a = V_{n+1}$
end for
12: **return** (V_{n+1}, π_{n+1})

(a) SPUDD

Require: Pr, R as ADDs of state and action variables
$V' \leftarrow$Swap each X_i variable in V_n with X_i'
Apply constraints to V'
3: **for** each X_i' **do**
$V' \leftarrow V' \otimes Pr(X_i'|X, A)$
$V' \leftarrow \sum_{X_i'} V'$
6: **end for**
$Q_{n+1} \leftarrow R \oplus \gamma V'$
$\pi_{n+1} \leftarrow \arg\boxed{\max}_{A_1,\ldots,A_m}(Q_{n+1})$
9: $V_{n+1} \leftarrow \boxed{\max}_{A_1,\ldots,A_m} Q_{n+1}$
return (V_{n+1}, π_{n+1})

(b) Factored-Action Regression (FAR)

Figure 2: Comparison of FAR and SPUDD

MBFAR (Value function ADD V', Constraints on action variables Z)

Apply action variable constraints Z to V'
for each X_i' in V' **do**
3: **if** $size(V') > C$ **then**
$a_p \leftarrow$ pick action variable
$Q^{a_p=T}, \pi^{a_p=T} \leftarrow$ MBFAR$(V', Z \cup \{a_p = T\})$
6: $Q^{a_p=F}, \pi^{a_p=F} \leftarrow$MBFAR$(V', Z \cup \{a_p = F\})$
$Q^Z \leftarrow \max(Q^{a_p=T}, Q^{a_p=F})$
$\pi^Z \leftarrow \pi^{a_p=T}$ in states where $Q^{a_p=T} \geq Q^{a_p=F}$
9: $\leftarrow \pi^{a_p=F}$ in others
return (Q^Z, π^Z)
else
12: $V' \leftarrow V' \otimes Pr^Z(X_i'|X, A - Z)$
$V' \leftarrow \sum_{X_i'} V'$
end if
15: **end for**
$Q^Z \leftarrow \boxed{\max}_{A_1 \ldots A_m}(R \oplus \gamma V')$
$\pi^Z \leftarrow \arg\boxed{\max}_{A_1,\ldots,A_m} Q^Z$
18: **return** (Q^Z, π^Z)

Figure 3: The MBFAR Algorithm

over explicitly. Importantly, the set of variables to ground are decided dynamically in a recursive manner. The MBFAR algorithm is shown in Figure 3. The algorithm takes as inputs the current value function V' in the form of an ADD and Z specifying the set of constraints on action variables i.e. a partially instantiated joint action. Line 1 restricts action variables in V' to their assignments in Z. Then, as long as the value functions calculated are small our algorithm behaves exactly as FAR (lines 11-14). If any intermediate stage of regression exceeds the prespecified space bound C, the algorithm picks an action variable a_p to ground, and calls the function recursively twice by adding the constraints $\{a_p = T\}$ and $\{a_p = F\}$ respectively. It then finds the max of the two ADDs returned by the recursive calls and updates the policy (lines 3-11).

With some abuse of notation, $Pr^Z(X_i|X, A - Z)$ represents the conditional probability of X_i as a function of the state variables X and action variables A except those that are grounded in Z. Q^Z denotes the generalized Q-function over all actions that satisfy the constraints in Z. These functions are similar to the generalized X-value functions over subsets of actions introduced by Pazis and Parr (2011). However, unlike in their work where these subsets are hand-crafted, we provide a method for constructing the subsets incrementally. In particular, the grounded variable a_p is dynamically chosen to be the action variable with the highest out-degree in the influence diagram. Hence the action subsets can be different for different groundings of previous variables which makes the approach more flexible.

The top level function is called with the current value function ADD V_n where all the variables X are replaced by their primed variants X' having already applied the state-action constraints, and with the empty set of constraints. MBFAR returns the next level value function V_{n+1} and the corresponding policy ADDs.

6 Sequential Hindsight Approximation

Our final contribution is a new approximate SDP approach we call "sequential hindsight" that yields a significant speedup when the states and actions can be decomposed into several weakly coupled subMDPs. The method yields an exact value function for *Compositional MDPs* where the dynamics are a cross product composition of independent subMDPs, and the reward function is an additive combination of the subMDPs. This is the situation for a system with full concurrency and no interaction, where we can solve each subMDP separately. We illustrate the idea with a simple example with two subMDPs. In this case, regression requires us to calculate

$$V_{n+1} = \boxed{\max_{A_1,A_2}} R(X_1, X_2, A_1, A_2) \oplus \left(E_{X_1', X_2'|X_1,A_1,X_2,A_2} \gamma V_n'\right)$$

but because of the decomposition this is equivalent to

$$V_{n+1} = \boxed{\max}_{A_1} R(X_1, A_1) \oplus E_{X_1'|X_1,A_1} \left(\boxed{\max}_{A_2} R(X_2, A_2) \oplus E_{X_2'|X_2,A_2} \gamma V_n'\right).$$

The key idea is to interchange the expectation $E_{X'_1}$ and maximization $\max_{A_2}$ which is correct for compositional MDPs, and yields computational savings. For non-compositional MDPs, the resulting value function is an upper bound on the optimal value function. Our algorithm applies the same idea at the level of individual action variables instead of subMDPs. We next describe the modifications to the FAR algorithm (space constraints preclude inclusion of the pseudocode). Comparing to the pseudocode for FAR the maximization at line 9 is moved inside the summation loop after line 5. The approximation algorithm checks for any action variables a_p such that all X'_j variables associated with a_p have already been summed out (cf. A_2 when X'_2 has been summed out). The heuristic assumes that these action variables are "independent of" X' variables in V' and therefore eliminates them (taking the max over their values) at this stage. When the assumption does not hold then the choice of a_2 in $\max_{a_2 \in A_2} E_{X'_2|X_2,X_1,X'_1,a_2,a_1} V'_n$ is done with "hindsight" of the value of a_1 and X'_1 (which is the outcome of action A_1) for every a_1, X'_1 pair, and hence the name sequential hindsight. For the example shown in Figure 1, $reboot(c1)$ is maximized for each value of $running'(c2)$ after expection is computed over $running'(c1)$, . In contrast with previous work on hindsight optimization (Yoon et al. 2008) we do not use sampling, nor do we assume the knowledge of the complete next state having only instantiated some of the variables (X'_1 but not X'_2 in our example).

Next we give an example that this algorithm can fail if the transition or reward functions of subMDPs are correlated. Consider a lottery ticket domain, with two tickets and where at most one ticket wins, each with a low probability. Assume that buying a ticket has a small cost and the optimal action is to buy both tickets. Under the assumption that the first ticket does not win ($X'_1 = 0$), $\max_{a_2 \in A_2} E_{X'_2|X_2,X_1,X'_1,a_2,a_1} V'_n$ picks the action of buying the second ticket. Under the assumption that the first ticket wins ($X'_1 = 1$) and $a_1 = 1$, the second ticket is not bought. Thus the value calculated by this method never chooses to buy both tickets, and yet assumes that the reward can be collected, and is therefore an overestimate.

None of the domains in our experiments is completely compositional although they are all weakly coupled. Nonetheless our experiments demonstrate that sequential hindsight approximation is very effective in terms of the quality of the policies generated.

7 Experiments

In this section we investigate the performance of the methods proposed and compare them to the de facto standard SDP algorithm, SPUDD, in multiple domains. As our focus is on factored action spaces, in most experiments we fix the size of the state space and increase the number of factors in the action space. This shows scaling purely as a function of the size of action space.

We use the Relational Dynamic Influence Diagram Language (RDDL) introduced in the International Planning Competition (IPC) 2011 (Sanner 2011) for input specification of our domains. RDDL allows for easy specification of parametrized models, provides a compact influence diagram specification for problems with exogenous events, and allows to specify state and action constraints (e.g., action preconditions, or inconsistent combinations of state or action variables). For our algorithms, we translate the RDDL model into a compact propositional model using algebraic decision diagrams (ADD) as used in SDP.

To complete the details of the algorithms we must specify the ordering of variables in ADDs and the ordering of sum and max operations. In our experiments the ordering of variables within ADDs puts $parents(X'_i)$ above X'_i, where the X'_is are ordered by the number of parents that are action variables. The order of expectations is done top-down with respect to variable ordering (we regress exogenous events first), and maximization operations are done in a bottom-up order with respect to the variable ordering. Unless a specific iteration bound is given below, in all experiments we stop VI upon convergence (bellman error smaller than 0.01) or after 40 iterations. Our experiments are performed on a single core of a Intel Core 2 Quad 2.83GHz with a memory limit of 4GB.

Inventory Control: We first consider a simple inventory control domain. The domain consists of n independent shops each being full or empty, where all shops are empty in the initial state. Each shop has an associated deterministic action of filling the shop (in one time step) and total number of shops that can be filled in one time step is bounded by the number of available "trucks" m. Thus we can take at most m parallel actions among the n possible parallel actions. The cost of filling a shop is -0.1. The shops become empty via exogenous events: a nonempty shop that is not being filled becomes empty with a probability $p = 0.15$. The reward is additive, and at every time step a reward of -1 is given per empty shop. Note that if $m \geq n$ then the domain (for the above parameters) is very simple because all empty shops can be filled at every time step. When $m < n$ the complexity of the policy increases from $m = 1$ to $n/2$ and decreases for larger values for the same reason.

The results are shown in Figure 4 and they confirm the expected dependence on the number of parallel actions. MBFAR is parameterized by C that represents the total number of nodes in its ADDs. We see that SPUDD and MBFAR(200) are significantly slower than the other algorithms. MBFAR's run time varies, and in intermediate settings of C it is faster than FAR demonstrating the need for space-time trade-off. The sequential hindsight method V_H is the fastest. To evaluate the quality of the policies produced by this method we measure the accumulated discounted reward for 30 random initial states using 30 episodes from each state. We tested the hypothesis that V_H and the optimal policy (obtained using FAR) have the same mean reward using a welch's test with a one sided significance level of 0.10. The mean rewards are statistically indistinguishable and for all states the returns are within one standard deviation.

Figure 5 further explores the space-time trade-off. In this case FAR has similar run time to MBFAR on the first two iterations (the lines are overlapping in the figure) but it exceeds memory limit on the third iteration. SPUDD is slower than MBFAR and it exceeds the 120 minute time limit in the 6th iteration. MBFAR scales to run more iterations.

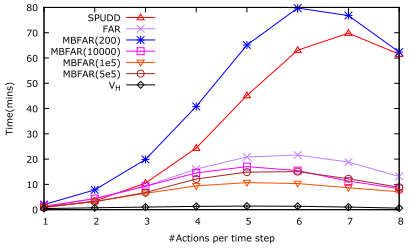

Figure 4: Time for solution using ADDs - 11 shops

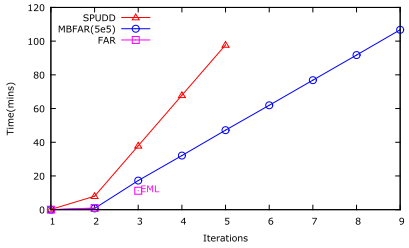

Figure 5: Time for solution - 15 shops and 3 trucks

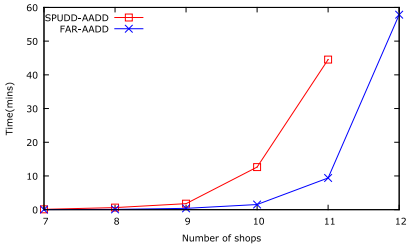

Figure 6: Time for solution using AADDs - trucks/shops = 0.5

The linear plots show that the structure of the value function has converged in a small number of iterations while the bellman error has not. In fact, this turns out to be the case in all our domains, which shows the power of SDP. However, we have also observed that the ADD size grows exponentially with n in this domain mainly because the additive reward and value function are not easily captured by ADDs. This was the motivation for the introduction of Affine ADDs (AADDs) (Sanner and McAllester 2005). Figure 6 shows the scaling with size of state space using AADDs. We see that in this case too FAR scales better than SPUDD.

In our experiments we have observed that although AADDs are more compact, their normalization operations introduce numerical rounding errors that compromise the quality of the policies. Hence we report only results with ADDs on the rest of the domains. However, we note that this issue is orthogonal to our approach which can work with any compact representation that can support SDP operations.

System Administration: The "SysAdmin" domain was part of the IPC 2011 benchmark and was introduced in earlier work (Guestrin, Koller, and Parr 2001; Sanner and McAllester 2005). It consists of a network of n computers connected in a given topology. Each computer is either running or failed so that $|S| = 2^n$ where all computers are running in the initial state. Each computer has an associated deterministic action of rebooting the computer and there is an action cost of -0.75 for each reboot action. In our experiments we allow for reboot of a bounded number of computers in one time step. Unlike the previous domain, here the exogenous events for computers are not independent. A running computer that is not being rebooted is running in the next state with probability p proportional to the number of its running neighbors, where $p = 0.45 + 0.5 * \frac{1+n_r}{1+n_c}$, n_r is the number of running neighbors and n_c is the number of neighbors. The reward is additive and a reward of +1 is given for each running computer.

We test this domain on three topologies, a unidirectional ring (Figure 7), a bidirectional ring (Figure 8) and a star topology (Figure 9). In the unidirectional ring, we fix the number of computers and split them into independent rings. Action parallelism is controlled by allowing only one action per ring. In the bidirectional ring, we use a fixed number of computers and increase the number of allowed simultaneous reboots. Concurrent actions are important in this network since it is better to reboot pairs of adjacent computers as they influence each other. In the star network computers are connected in a breadth first fashion, with each computer connected to up to three other computers. Computers near the center of the star are more important. Here too we fix the number of computers and increase the number of simultaneous reboots.

The figures show that SPUDD requires exponential time across network topology and fails to scale beyond the smallest concurrent problems. In spite of its high memory usage FAR scales well with the concurrency. MBFAR lines show the interpolation between SPUDD and FAR and MBFAR with an intermediate bound is sometimes faster than FAR. The sequential hindsight method is the fastest on this domain. Using the same method to evaluate V_H as in IC, we found that the mean return is not distinguishable from the return of the optimal policy.

Elevator Control: The elevators domain is a challenging domain for probabilistic planners, and was part of the IPC 2011 benchmark. Here we simplify the domain to be solvable by SDP. A building with n floors is serviced by m elevators, where each elevator is assigned a subset of the floors. The floors are divided uniformly between elevators with some common floors. For each floor a fluent denotes whether a person is waiting. People only wait to go up. For each elevator the fluents denote whether a person is in the elevator and the current direction of the elevator. For each elevator-floor pair a fluent denotes whether the elevator is at that floor. Therefore the size of the state space is $|\mathbb{S}| = 2^{n+2m+mn}$. Each elevator moves independently and has 3 actions: move-up, move-down and change direction, so that $|\mathbb{A}| = 3^m$. The dynamics implicitly imply that a person waiting at a floor moves into the elevator if the direction of the elevator is also up. A person in an elevator exits if the elevator cannot move up anymore. In this domain the exogenous events describe arrival of people, where a person arrives at a floor with probability $p = 0.15$. The reward function is additive. At each time step, each person waiting at a floor gives a reward of -1. Each person in an elevator gives a reward of -0.75 if the elevator is going in her desired direction and -3.0 otherwise. Note that unlike previous domains, in this domain increasing the action space, i.e., the number of elevators, also increases the state space.

In the one elevator case (Figure 10), we see that MBFAR(200) is faster than SPUDD. The only difference between these algorithms is that exogenous events are regressed only once by MBFAR while they are regressed once for each joint action in SPUDD. When increasing the number of elevators, the gap between FAR and SPUDD widens and FAR is an order of magnitude faster than SPUDD for three elevators (Figure 12). The sequential hindsight method is the fastest on this domain too. Using the same method to evaluate V_H as in

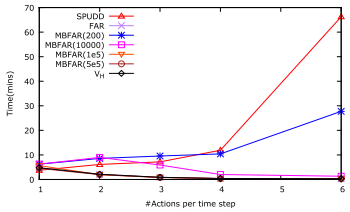

Figure 7: Time for solution - 12 computers Unidirectional ring

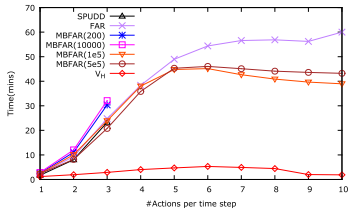

Figure 8: Time for solution - 11 computers Bidirectional ring

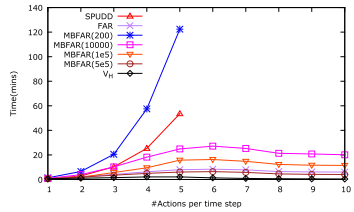

Figure 9: Time for solution - 12 computers star network

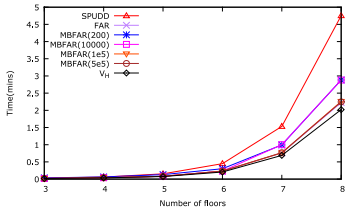

Figure 10: Time for solution - 1 elevator

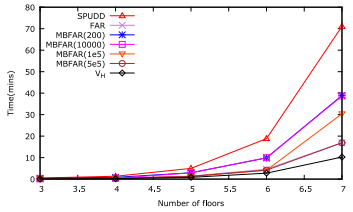

Figure 11: Time for solution - 2 elevators

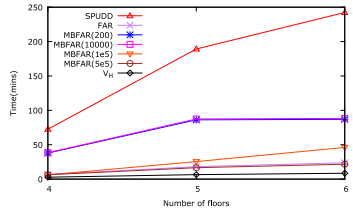

Figure 12: Time for solution - 3 elevators

IC, we found that the mean return is not distinguishable from the return of the optimal policy.

8 Discussion and future work

In this paper we presented the first symbolic dynamic programming algorithms for planning in MDPs with large action spaces. Our methods have fundamental advantages due to explicitly representing action variables within ADDs thus capturing state-action structure. Extension to existing decision diagram (DD) techniques e.g. APRICODD (St-Aubin, Hoey, and Boutilier 2001) and Multi-value DDs is straightforward. Extension to first order diagrams (Wang, Joshi, and Khardon 2008) is a possible future direction. This work can also be applied to cooperative multi-agent MDPs where the state is globally known. The MBFAR algorithm inspired by recursive conditioning provides a principled way of trading memory for time. We also presented a new heuristic SDP approach which is near optimal in the domains we tested. Analyzing the error in sequential hindsight approximation is also future work.

Acknowledgments

The authors would like to thank Scott Sanner for making available an ADDs library. This work is supported by NSF under grants IIS-0964457 and IIS-0964705, and the CI fellows award for Saket Joshi.

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(2):171–206.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, 157–171.

Darwiche, A. 2001. Recursive conditioning. *Artif. Intell.* 126:5–41.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147(12):163 – 223.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Multiagent planning with factored MDPs. *Advances in neural information processing systems* 14:1523–1530.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288.

Kim, K.-E., and Dean, T. 2002. Solving factored MDPs with large action space using algebraic decision diagrams. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*, PRICAI '02, 80–89. London, UK, UK: Springer-Verlag.

Little, I., and Thiebaux, S. 2006. Concurrent probabilistic planning in the graphplan framework. In *Proc. ICAPS*, volume 6, 263–272.

Mausam, M., and Weld, D. 2004. Solving concurrent Markov decision processes. In *Proceedings of the 19th national conference on Artifical intelligence*, 716–722. AAAI Press.

Pazis, J., and Parr, R. 2011. Generalized value functions for large action sets. In *Proceedings of the Twenty Eighth International Conference on Machine Learning(ICML)*, 1185–1193.

Puterman, M. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

Sanner, S., and McAllester, D. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *International Joint Conference on Artificial Intelligence*, volume 19, 1384.

Sanner, S.; Uther, W.; and Delgado, K. 2010. Approximate dynamic programming with affine ADDs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, 1349–1356.

Sanner, S. 2011. Relational dynamic influence diagram language (rddl): Language description. *NICTA, Australia*.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2001. Apricodd: Approximate policy construction using decision diagrams. *Advances in Neural Information Processing Systems* 1089–1096.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research* 31(1):431–472.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd national conference on Artificial intelligence*, volume 2.

Younes, H., and Simmons, R. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS04*, volume 325.