Solving Factored MDPs via Non-Homogeneous Partitioning

Kee-Eung Kim and **Thomas Dean**
Department of Computer Science
Brown University
Providence, RI 02912-1910
{kek,tld}@cs.brown.edu

Abstract

This paper describes an algorithm for solving large state-space MDPs (represented as *factored* MDPs) using search by successive refinement in the space of non-homogeneous partitions. Homogeneity is defined in terms of bisimulation and reward equivalence within blocks of a partition. Since homogeneous partitions that define equivalent reduced state-space MDPs can have a large number of blocks, we relax the requirement of homogeneity. The algorithm constructs approximate aggregate MDPs from non-homogeneous partitions, solves the aggregate MDPs exactly, and then uses the resulting value functions as part of a heuristic in refining the current best non-homogeneous partition. We outline the theory motivating the use of this heuristic and present empirical results and comparisons.

1 Introduction

Markov Decision Processes (MDPs) employing representations that factor states, actions and their associated transition and reward functions in terms of component functions of state variables (fluents) have surfaced as plausible models for planning under uncertainty [Boutilier *et al.*, 1999]. Since the number of states in these factored MDPs (FMDPs) is exponential in the number of fluents, traditional iterative methods that require enumerating states are typically not effective. In this paper, we solve FMDPs by constructing and refining non-homogeneous partitions of the state space. In a homogeneous partition, any two states in a block have the same reward and the same distribution with respect to transitions to other blocks. Non-homogeneous partitions allow variation among the states in a block with regard to rewards and block transition distributions. From a non-homogeneous partition, we construct an aggregate MDP by averaging the transition probabilities and the rewards within blocks.

The algorithm described in this paper solves FMDPs by successive refinement in the space of non-homogeneous partitions. We use factored representations to encode the partitions and the aggregate MDPs we construct from them. We can solve the aggregate MDPs in time polynomial in the number of blocks in the partition. The resulting optimal policies (optimal for the aggregate MDPs) also serve as policies for the original MDP and the value function for the optimal policy in the aggregate MDP serves as an estimate for the value of the policy in the original MDP. Given this estimate and the current partition, we choose the refinement that yields the greatest improvement and iterate. In the remainder of this paper, we present some background, provide a theoretical motivation for our refinement heuristic, and describe the results of a series of experiments.

2 Factored MDPs (FMDPs)

Definition 1 (FMDP) *An* FMDP *is defined as a tuple* $M = \{\vec{X}, A, T, R\}$ *where*

- $\vec{X} = [X_1, \ldots, X_n]$ *is a vector of fluents which collectively define the state. We use* Ω_{X_i} *to denote the set of values for* X_i. Ω_{X_i} *is the sample space of* X_i *when* X_i *is considered as a random variable. Thus, the state space for* M *is* $\Omega_{\vec{X}} = \prod_i \Omega_{X_i}$. *We use lower case letters* $\vec{x} = [x_1, \ldots, x_n]$ *to denote a particular instantiation of the fluents, and* $X_{i,t}$ *and* $x_{i,t}$ *denote, respectively, a fluent and its value at a particular time* t.
- A *is the set of actions.*
- T *is the set of transition probabilities, represented as the set of conditional probability distributions, one for each action and fluent:*

$$T(\vec{X}_t, a, \vec{X}_{t+1}) = \prod_{i=1}^{n} P(X_{i,t+1} | \text{pa}(X_{i,t+1}), a)$$

  where $\text{pa}(X_{i,t+1})$ *denotes the set of parents of* $X_{i,t+1}$ *in the graphical model (see below). Note that* $\forall i, \text{pa}(X_{i,t+1}) \subseteq \{X_{1,t}, \ldots, X_{n,t}\}$.
- $R : \vec{X} \rightarrow \Re$ *is the reward function. Without loss of generality, we define the reward to be determined by both state and action (*$R : \vec{X} \times A \rightarrow \Re$*).*

Figure 1 shows an example of a graphical model for the dynamics governing an action in a toy robot domain with five boolean variables. The robot has to deliver a cup of coffee whenever requested. Unfortunately, the coffee bar is outside the building and the robot receives a small amount of punishment if it gets wet. Each fluent denotes a particular aspect of

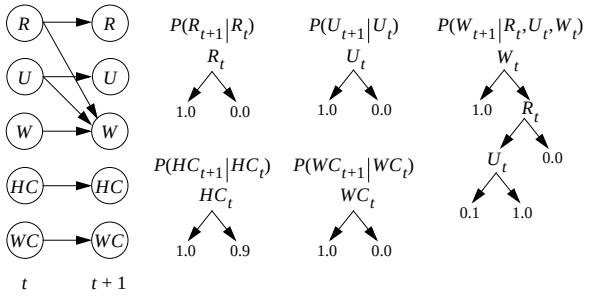

Figure 1: Graphical representation of an FMDP describing a toy robot domain.

the world: the weather outside being rainy (R), the robot having an umbrella (U), the robot being wet (W), the robot holding coffee (HC), and someone wanting coffee (WC). Note that the probabilistic effect of each fluent at time $t+1$ is conditioned on a small number of fluents (the set of parent fluents, *e.g.*, pa$(W_{t+1}) = \{R_t, U_t, W_t\}$) at time t. The conditional probabilities are stored as trees rather than tables for further savings in the size of the representation. Thus, we complete the description of the state dynamics by specifying conditional probability trees (CPTs) [Boutilier *et al.*, 1995] for each fluent and action. Similar representations have been used to model Markovian processes with factored state spaces, *e.g.*, the Two-stage Temporal Bayesian Network (2TBN) [Dean and Kanazawa, 1989] and the Dynamic Bayesian Network (DBN) [Forbes *et al.*, 1995]. The reward function R is also designated in terms of trees.

FMDPs exploit conditional independence among the fluents given their parents in the graphical model to achieve economy of representation. However, the underlying structure in the domain that makes compact representations possible does not lead necessarily to an efficient FMDP algorithm. There are algorithms modeled after classical MDP algorithms that use dynamic programming to iteratively update a value function represented in a factored form; however, the size of the value functions so represented can easily explode. There are also algorithms that compress the value functions in an effort to prevent an explosion, *e.g.*, using a tree representation for value functions with pruning techniques [Boutilier and Dearden, 1996], or using Algebraic Decision Diagrams (ADDs) for further reduction [Hoey *et al.*, 1999]. These algorithms are examples of Value Function Approximation (VFA) algorithms applied to FMDPs [Gordon, 1995; Tsitsiklis and Van Roy, 1996; Koller and Parr, 1999].

3 Stochastic Bisimulation Equivalence

Dean and Givan [1997] introduce the notion of *stochastic bisimulation homogeneity* for FMDPs. It is an extension of the state equivalence relation for minimizing Finite State Automata (FSA) to that of probabilistic FSAs. Dean and Givan's model minimization algorithm for FMDPs partitions the state space into *stable* blocks.

Definition 2 (Stable Block and Homogeneous Partition) *A block C of a partition P is said to be* stable with respect to a block B of P and an action a *if and only if every state in C has the same transition probability of ending up in block B by action a. Mathematically,*

$$\exists c \in [0,1] \text{ such that } \forall \vec{x}_t \in C, T(\vec{x}_t, a, B) = c$$

where

$$T(\vec{x}_t, a, B) = \sum_{\vec{x}_{t+1} \in B} T(\vec{x}_t, a, \vec{x}_{t+1}).$$

We say that C is stable *if C is stable with respect to every block of P and action in A. P is said to be* homogeneous *if and only if every block is stable.*

Given a homogeneous partition of an FMDP, we can define an aggregate MDP that is *equivalent* to the original FMDP. Every state within a block of a homogeneous partition has the same state dynamics with respect to any policy, *i.e.*, the transition probability of moving into a block is the same for every state in the same block. Thus, if every state within a block of a homogeneous partition has the same reward, the partition yields a reduced model of the original FMDP.

Givan *et al.* [2000] introduce *ϵ-homogeneity* for finding an approximately homogeneous partition with few blocks. We say that a block C is *ϵ-stable* with respect to B and a if

$$\exists c \in [0,1] \text{ such that } \forall \vec{x}_t \in C, |T(\vec{x}_t, a, B) - c| \leq \epsilon.$$

If every block of partition P is ϵ-stable with respect to every other block of P and every action, we say P is an ϵ-homogeneous partition. Qualitatively speaking, by relaxing the equality to be *approximately equal with error within ϵ*, we get a smaller partition than if we required strict homogeneity. The error in the optimal value function computed from an ϵ-homogeneous partition is discussed in the work of Singh and Yee [1994] and White and Eldeib [1994]. Most of the previous work on computing approximate solutions of FMDPs follow similar analyses [Boutilier and Dearden, 1996; St-Aubin *et al.*, 2000].

4 Non-Homogeneous Partitions for FMDPs

There are two important questions remaining to be answered concerning partition refinement techniques for FMDPs. First, what class of functions should we allow to represent block formulae? If we restrict the representational power of the formulae too much, we may end up with large partitions but with easily manipulable block formulae. If we allow arbitrary functions, manipulating block formulae is NP-hard (see [Goldsmith and Sloan, 2000] for recent analysis.) though we can compute the coarsest homogeneous partition which may be small. Second, what if the coarsest homogeneous partition is still exponentially large compared to the description size of the FMDP? Calculating ϵ-homogeneous partition helps, but note that we don't have a fine control over the size of the partition — all we know is that increasing ϵ will generally reduce the size of the partition and decreasing it will generally do the opposite.

Based on the above observation, we describe a new state aggregation algorithm that does not search for the coarsest homogeneous partition. In fact, the technique is not driven

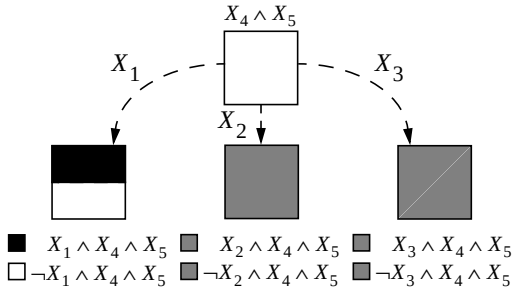

Figure 2: Chopping the block $X_4 \wedge X_5$ with respect to X_1, X_2 or X_3. Note that the blocks resulting from the CHOP operator are not stable in general.

by the notion of stochastic bisimulation homogeneity. Given an initial non-homogeneous partition of the state space, the algorithm iteratively decides which block to refine so that it produces at the end an approximately optimal policy without generating a huge partition. Thus, given a partition, the algorithm has to determine which block to refine with respect to which fluent. Note that the partitions we consider are not ϵ-homogeneous given that the transition probabilities can widely differ.

We show that the optimal value functions calculated on non-homogeneous partitions provide the basis for an effective heuristic for selecting the block-fluent pair to chop. First, we define how we construct the MDP from a non-homogeneous partition.

Definition 3 (MDP from Non-Homogeneous Partition) *Given an* FMDP $M = (\vec{X}, A, T, R)$ *and a non-homogeneous partition P of the state space $\Omega_{\vec{X}}$, the aggregate* MDP *induced by P, denoted as $M_P = (P, A, T_P, R_P)$, is defined as*

$$T_P(C, a, B) = \frac{1}{|C|} \sum_{\vec{x} \in C, \vec{x}' \in B} T(\vec{x}, a, \vec{x}')$$

$$R_P(C, a) = \frac{1}{|C|} \sum_{\vec{x} \in C} R(\vec{x}, a)$$

for all $B, C \in P$ and $a \in A$. V_P^* denotes the optimal value function of M_P, which is a mapping from $\Omega_{\vec{X}}$ to $\Re$. Note that for any $\vec{x}$ and $\vec{x}'$ in the same block of P, $V_P^*(\vec{x}) = V_P^*(\vec{x}')$ and $\pi^*(\vec{x}) = \pi^*(\vec{x}')$.

Given a partition P, the algorithm selects block C and fluent X for generating a refined partition P'. P' has the same blocks as P except for C which is replaced by CHOP(P, C, X). Figure 2 illustrates how the CHOP operator generates refined blocks of C. Recalling that n is the number of fluents in the domain, we note that there are at most $n|P|$ different refined partitions that can be obtained by the CHOP operator. For each refined partition P' generated by the CHOP operator, the algorithm constructs $M_{P'}$ and calculates the optimal value function $V_{P'}^*$.

We will present the algorithm shortly. But first, we would like to know how good it is to use M_P to solve the original FMDP. We prove that given any non-homogeneous partition

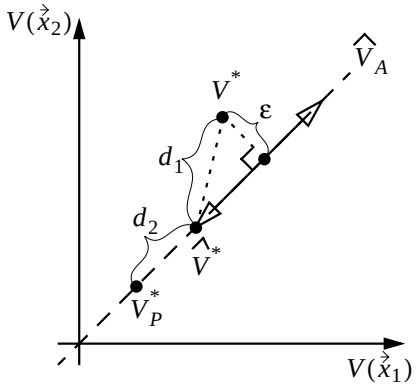

Figure 3: Value functions for an MDP constructed from a non-homogeneous partition. The figure illustrates an example where the MDP has two states and the partition has one block. The dashed line is the set of representable approximate value functions. $d_1 \leq 2(1 + \frac{\gamma}{1-\gamma})\epsilon$ and $d_2 \leq \frac{\|LV_P^* - V_P^*\|_\infty}{1-\gamma}$.

P, the optimal value function of M_P, which we define as V_P^*, is within a bounded distance from the true value function V^*. To this end, we introduce a definition and a theorem from Gordon [1995] that are used in proving Theorem 2.

Definition 4 (Averager) *An approximation function is an* averager *if, given the target vector U, the approximation function generates $\hat{U}$ defined as*

$$\hat{U}(i) = \beta_i c_i + \sum_j \beta_{ij} U(j)$$

where c_i is a constant and β_i and β_{ij} are non-negative constants such that $\forall i, \beta_i + \sum_j \beta_{ij} = 1$.

Theorem 1 (Gordon) *Let V^* be the optimal value function for an* MDP *M, L the update (Bellman backup) operator for value iteration,*

$$LV(\vec{x}) = \max_a \big[R(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \Omega_{\vec{X}}} T(\vec{x}, a, \vec{x}') V(\vec{x}')\big]$$

and A the mapping for an averager. Let V^A be any fixed point of A. Given $\|V^* - V^A\|_\infty = \epsilon$, the iteration of $L \circ A$ converges to a value function $V^{L \circ A}$ so that $\|V^* - V^{L \circ A}\| \leq 2\gamma\epsilon/(1-\gamma)$. Approximate value iteration using A returns $AV^{L \circ A}$ which satisfies $\|V^* - AV^{L \circ A}\| \leq 2\epsilon + 2\gamma\epsilon/(1-\gamma)$.

Theorem 2 (Bounded Distance between V^* and V_P^*) *Given an* FMDP *$M = (\vec{X}, A, T, I, R)$ and a partition P of the state space $\Omega_{\vec{X}}$, the optimal value function of M given as V^* and the optimal value function of M_P given as V_P^* satisfy the bound on the distance*

$$\|V^* - V_P^*\|_\infty \leq 2\Big(1 + \frac{\gamma}{1-\gamma}\Big)\epsilon + \frac{\|LV_P^* - V_P^*\|_\infty}{1-\gamma} \quad (1)$$

where $\epsilon = \min_{V_P} \|V^* - V_P\|_\infty$ (V_P being any value function such that for any block $B \in P$ and any $\vec{x}$ and $\vec{x}'$ in B, $V_P(\vec{x}) = V_P(\vec{x}')$).

Proof Recall from the definition that $\forall \vec{x} \in C$,

$$V^*(\vec{x}) = \max_a \big[R(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \vec{X}} T(\vec{x}, a, \vec{x}') V^*(\vec{x}')\big]$$

and

$$V_P^*(\vec{x}) = \max_a \big[\frac{1}{|C|} \sum_{\vec{x} \in C} R(\vec{x}, a) + \gamma \frac{1}{|C|} \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big].$$

We define another value function

$$\widehat{V}^*(\vec{x}) = \frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') \widehat{V}^*(B)\big].$$

To calculate the maximum distance between V^* and $\widehat{V}^*$, we define an averager A_P as follows:

$$\widehat{V}(\vec{x}) = (A_P V)(\vec{x}) = \sum_{\vec{x}' \in C} \frac{1}{|C|} V(\vec{x}') \text{ for } \forall C \in P, \forall \vec{x} \in C.$$

Note that each fixed point of A_P assigns the same value to states in the same block. For the example shown in Figure 3, the straight dotted line satisfying $\widehat{V}_A(\vec{x}_1) = \widehat{V}_A(\vec{x}_2)$ is the set of fixed points for A_P since $\vec{x}_1$ and $\vec{x}_2$ belong to the same block. Since P is a refinement of the reward partition, $\widehat{V}^*$ is the fixed point of the mapping $L \circ A_P$, and applying the result from Gordon [1995], we have that

$$\|\widehat{V}^* - V^*\|_\infty \leq 2(1 + \frac{\gamma}{1-\gamma})\epsilon \tag{2}$$

where ϵ is the distance between V^* and the closest fixed point of A_P. The distance between $\widehat{V}^*$ and V_P^* is calculated as follows: For any $\vec{x} \in C$,

$$\begin{aligned}
&|\widehat{V}^*(\vec{x}) - V_P^*(\vec{x})| \\
&= \Big|\frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') \widehat{V}^*(B)\big] \\
&- \frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big] \\
&+ \frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big] \\
&- \max_a \big[\frac{1}{|C|} \sum_{\vec{x} \in C} R(\vec{x}, a) + \gamma \frac{1}{|C|} \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big]\Big| \\
&\leq \gamma \|\widehat{V}^* - V_P^*\|_\infty + \|(A_P \circ L) V_P^* - V_P^*\|_\infty \\
&= \gamma \|\widehat{V}^* - V_P^*\|_\infty + \|(A_P \circ L) V_P^* - A_P V_P^*\|_\infty \\
&\leq \gamma \|\widehat{V}^* - V_P^*\|_\infty + \|L V_P^* - V_P^*\|_\infty
\end{aligned}$$

Since the above inequality holds for $\forall C \in P$, we have

$$\|\widehat{V}^* - V_P^*\|_\infty \leq \frac{\|L V_P^* - V_P^*\|_\infty}{1 - \gamma} \tag{3}$$

1. The number of iterations N, initial partition P of the state space (reward partition), and the optimal value function V_P^* for M_P.
2. For each block $C \in P$ and fluent $X_i, 1 \leq i \leq n$, compute $M_{P'}$ where P' is the same as P except C is replaced by CHOP(P, C, X_i).
3. For each P', compute $V_{P'}^*$ and then select $P^* = \arg\max_{P'} \|V_{P'}^* - V_P^*\|$.
4. If Step 2∼3 has been run N times, halt and output $\pi_{P^*}^*$.
5. Set $P = P^*$ and go to Step 2.

Figure 4: The algorithm for finding a non-homogeneous partition for an approximately optimal policy

By combining Equation 2 and Equation 3, we have shown that

$$\begin{aligned}
\|V^* - V_P^*\|_\infty &\leq \|V^* - \widehat{V}^*\|_\infty + \|\widehat{V}^* - V_P^*\|_\infty \\
&\leq 2\Big(1 + \frac{\gamma}{1-\gamma}\Big)\epsilon + \frac{\|L V_P^* - V_P^*\|_\infty}{1-\gamma}
\end{aligned}$$

□

There are two remarks on the bound in the above theorem. First, $\epsilon \equiv \min_{V_P} \|V^* - V_P\|_\infty$ that appears in the first additive term in Equation 1 measures how fine is the partition P. ϵ is the minimum error that we can achieve by assigning values to the components of V_P, with the restriction that the values should be the same for the components that belong to the same block in P. Thus, we naturally expect that a finer partition will achieve a smaller ϵ, although it depends on how the state space is partitioned. At least, given a partition P and its refinement $P' \subseteq P$, and letting ϵ_P and $\epsilon_{P'}$ be the ϵ of P and P' respectively, we can say that $\epsilon_P \geq \epsilon_{P'}$. Note also that ϵ becomes 0 for a homogeneous partition. Second, $\|L V_P^* - V_P^*\|_\infty$ in the second additive term also vanishes as the partition becomes finer. Although there is no guarantee that the error is monotonically smaller for a finer partition and larger for a coarser partition, at least this error serves as an upper bound on the distance.

The algorithm that we describe in the following selects the *best* refined partition P' from the current partition P given by the formula

$$\arg\max_{P'} \|V_{P'}^* - V_P^*\|.$$

Theorem 2 indicates that by choosing a refinement P' of P maximizing the distance between $V_{P'}^*$ and V_P^* there is reason to believe that we are making significant progress in approaching V^*. This is the same sort of argument used to justify the use of discretization strategies for continuous state-space MDPs.

The algorithm runs iteratively by refining the partition for a pre-determined number of iterations so that we end up with a fixed-size policy. Figure 4 shows the algorithm. Note that constructing $M_{P'}$ can be done efficiently. The new transition probability matrix $T_{P'}$ typically has many components

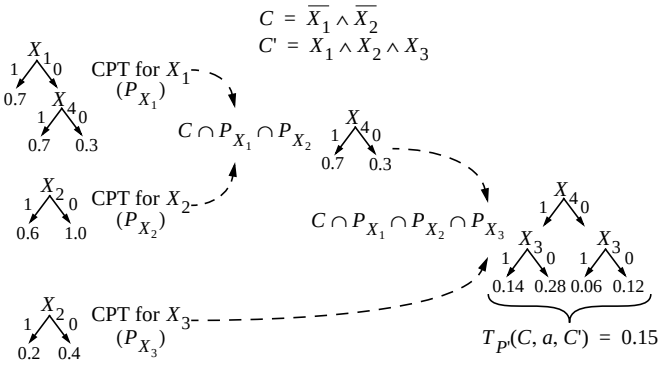

Figure 5: Calculating $T_{P'}(C, a, C')$ with decision trees.

the same as T_P. By reusing the components already calculated, we can construct the $M_{P'}$s without computing transition probabilities and rewards for all blocks. For example, if the partition P' was obtained by chopping a block $B \in P$ with respect to the fluent X_i so that $B = \{B_1, \ldots, B_{|\Omega_{X_i}|}\}$, we have

$$T_{P'}(C, a, C') = T_P(C, a, C'), \ \ \forall a \in A, \forall C \notin B, \forall C' \notin B.$$

The remaining components are computed without explicitly enumerating all of the states in the blocks. Figure 5 shows how we calculate the component $T_{P'}(C, a, C')$ using decision trees when $C \in B$ or $C' \in B$. To compute the component, we first graft CPTs and multiply the probabilities in the terminal nodes. Since we are only interested in the transition probability related to blocks C and C', we can eliminate the branches of the constructed tree that do not intersect with block C. This greatly reduces the size of the tree. The final step is to take the average of the items in the terminal nodes in the tree, weighted by the sizes of the blocks that the terminal nodes represent.

The reward function $R_{P'}$ for $M_{P'}$ is obtained similarly.

5 Experiments

The test problems used in our experiments are adopted from Hoey *et al.* [1999] and involve domains with 6 to 17 binary variables (fluents). The initial probabilities are given as a uniform distribution. For each domain, we show the performance of the policy derived from the non-homogeneous partition and the cumulative elapsed time at each iteration. We use CPLEX 6.5 for calculating optimal value functions of aggregate MDPs induced by non-homogeneous partitions, and the CUDD package [Somenzi, 1998] to implement structured versions of iterative algorithms (similar to SPUDD [Hoey *et al.*, 1999]) for evaluating the policies and calculating the optimal value functions. All experiments are run on a Sun Ultra10 with 256Mb of memory.

Figure 6 through Figure 8 illustrate the performance of the algorithm in Figure 4 on three benchmark domains. We ran the algorithm for 100 iterations, except for the COFFEE domain since it consisted of only 64 states. For each figure, the graph on the left side shows the actual performance (actual value of the optimal policy calculated from the aggregated MDP) and the heuristic value (the optimal value of the aggregated MDP) after each iteration. *The values are obtained assuming uniform starting probabilities on the states.* The graphs on the right sides show the plots of cumulative elapsed time (in seconds) after each iteration. Note that in some domains, there is a small gap between the actual performance and the heuristic value even when the policy from the aggregated MDP reached the optimal performance. This is due to the fact that the evaluation is done through an iterative method with the stopping condition $\|V_{i+1} - V_i\| \leq \delta$, which we set δ to 0.01.

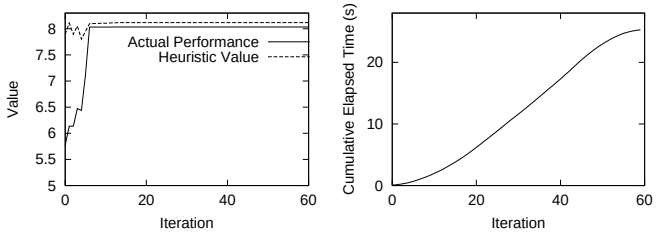

Figure 6: COFFEE domain (6 variables, 64 states). The non-homogeneous partitioning algorithm found the optimal policy after 6 chops, totaling 10 blocks in the partition. The optimal value is 8.12.

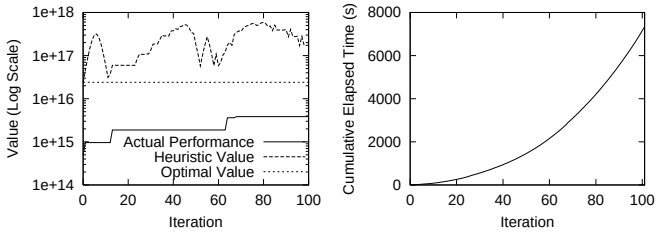

Figure 7: EXPON domain (12 variables, 4096 states). After 100 chops (113 blocks), the non-homogeneous partitioning algorithm yields an approximately optimal policy. The optimal value is 2.4×10^{16}.

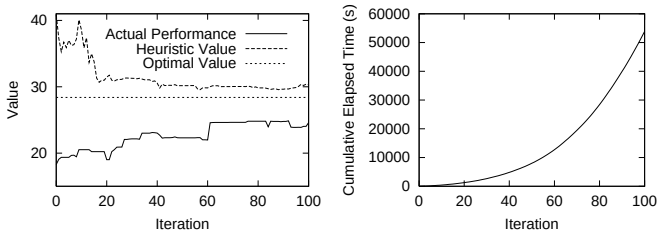

Figure 8: FACTORY domain (17 variables, 131,072 states). After 100 chops (126 blocks), the non-homogeneous partitioning algorithm yields an approximately optimal policy. The optimal value is 28.4.

The optimal value function of the EXPON domain (Figure 7) has thousands of internal nodes even when represented as an ADD. The non-homogeneous partitioning algorithm is not able to find the optimal policy with partition size less than or equal to 113, however, the policy at the end of 100th itera-

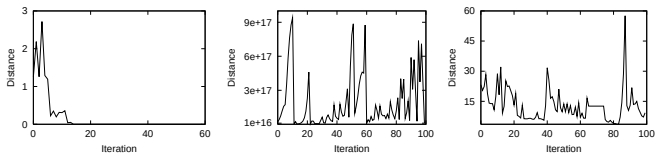

Figure 9: Distance plots between two value functions of successive aggregate MDPs ($\|V^*_{P^*} - V^*_P\|$ in Figure 4). From left to right: COFFEE, EXPON, FACTORY.

tion performs 10 times better than the initial policy from the aggregate MDP induced by the reward partition. The size of the ADD representation for the optimal value function is also quite large in the case of the FACTORY domain (Figure 8). After 100 chops, totaling 126 blocks, the policy from the aggregated MDP has a value of 24.8 which is 87% of the optimal value.

We also experimented on the LINEAR domain (14 variables, 16,384 states). The optimal policy of the aggregate MDP from the reward partition yields the optimal value, *i.e.*, the optimal policy was obtained without any split. It takes 23.28 seconds for 1 iteration. When the algorithm finds out that every refinement P' of the current partition P yields $\|V^*_{P'} - V^*_P\| = 0$, then it knows the current optimal policy from the aggregated MDP is indeed optimal. Meanwhile, our implementation of the structured value iteration using CUDD package (which is not as fully optimized as SPUDD) takes 43.58 seconds (15 leaves).

Figure 9 shows the distances between two value functions of successive aggregate MDPs for each domain. We excluded the LINEAR domain since the distances were zero from the onset. Note that in the COFFEE domain, the distance decreases sharply after the optimal policy from the aggregate MDP is actually optimal. However, the distances after that are not zero, which implies the partition is still non-homogeneous. We do not observe such behavior in the EXPON and FACTORY domain since the algorithm was not able to yield the optimal policy. Although the distances between consecutive value functions are large, we note that the actual performance does not change much in the two domains. It is natural that a big difference in value functions does not necessarily imply a big difference in the actual performance. Sometimes, a block containing states with highly varing transition probabilities is not so important to reach for optimal performance, hence chopping the block does not greatly improve the actual performance of the policy.

We also compare the performance of non-homogeneous partitioning algorithm to APRICODD [St-Aubin *et al.*, 2000]. Figure 10 and Figure 11 summarize the comparison of the two algorithms on the larger domains, EXPON and FACTORY. By trial and error, we tuned the pruning parameter of the APRICODD algorithm so that the size of the approximate value function from the APRICODD algorithm is comparable to that of the non-homogeneous partition at the end of 100 chops. We used "sliding-tolerance" pruning technique with different parameters. We allow two to three times more nodes in the decision diagrams than the number of blocks in the non-homogeneous partition. Often, increasing the pruning

	Perf.	Blocks		Time
Non-homogeneous	16 %	113		7333 s
	Perf.	Nodes	Leaves	Time
APRICODD (0.4)	48 %	320	65	630 s
APRICODD (0.5)	23 %	246	33	73 s

Figure 10: Comparison of the non-homogeneous partitioning algorithm and the APRICODD on EXPON domain. The number inside the parenthesis is the pruning parameter for the "sliding-tolerance" pruning. The Perf. is the ratio between the performance of the optimal policy and that of the approximate policy assuming uniform starting distribution on states. We also present the number of nodes and leaves in the ADD representation of approximate value function from the APRICODD.

	Perf.	Blocks		Time
Non-homogeneous	87 %	126		55402 s
	Perf.	Nodes	Leaves	Time
APRICODD (0.2)	67 %	342	73	920 s
APRICODD (0.3)	26 %	252	65	893 s

Figure 11: Comparison of the non-homogeneous partitioning algorithm and the APRICODD on FACTORY domain.

parameter so that we get smaller value functions from APRICODD results in non-converging behavior. In fact, APRICODD on the EXPON domain with pruning parameter 0.5 does not converge. In this case, we stopped the algorithm when the value functions between consecutive iterations were oscillating (500 iterations). Note that we are still allowing the APRICODD to search in the space of a much richer representation for value functions — an ADD with 246 nodes can represent a much finer partition than a partition with 113 blocks. In terms of the number of blocks, the non-homogeneous partitioning algorithm performs comparable to the APRICODD algorithm. Note also that our implementation of APRICODD is not fully optimized, and that the running times may be significantly greater than those reported by the original authors.

We are currently exploring heuristics for efficiently selecting block-fluent pairs to chop. We note that solving continuous MDPs faces a similar problem, and we are experimenting on discretization heuristics such as Munos and Moore [Munos and Moore, 1999]. A preliminary result shows that this heuristic is highly effective, yielding approximately 50-fold speed up in some domains. A preliminary report on this experiment appears in [Kim, 2001].

The above experiments show two advantages of using the non-homogeneous partitioning algorithm. First, while in some domains the coarsest homogeneous partition may be quite large, it may not be critical to compute a homogeneous partition to obtain an optimal (or near optimal) policy. Structured value iteration algorithms such as SPUDD or APRICODD may incur a large cost in representing an optimal or near optimal value function, whereas the non-homogeneous partitioning algorithm does not necessarily face such a problem. Second, the algorithm provides a new approach to finding an approximately optimal policy, which allows us to specify the desired size of the policy *a priori*.

6 Conclusion and Related Work

To achieve economy of representation many algorithms aggregate states that have the same (or roughly the same) value; they do so using a variety representations ranging from simple set representations [Bertsekas and Castañon, 1989], decision trees [Boutilier *et al.*, 1995; Boutilier and Dearden, 1996] and algebraic decision diagrams [Hoey *et al.*, 1999; St-Aubin *et al.*, 2000], to linear combinations of simple basis functions [Koller and Parr, 1999] and neural networks [Bertsekas and Tsitsiklis, 1996].

While all that is necessary is that the states that are grouped together have the same value, it is often the case that the aggregated states have other properties in common. Structured value iteration [Boutilier *et al.*, 1995] and structured model reduction [Dean and Givan, 1997] group states according to stochastic bisimulation equivalence. This is a sufficient but not necessary criterion and in some cases it can result in partitions that are larger than strictly necessary. In the case of (exact) structured value iteration the leaves of the decision tree constitute the blocks of a partition that is reward and transition homogeneous. The structured model reduction algorithm successively refines an initial partition by splitting non-homogeneous blocks and is guaranteed to terminate with the coarsest homogeneous partition; unfortunately, this partition could have a number of blocks exponential in the number of fluents. This paper provides a method for splitting blocks that provides a global perspective (rather than considering each block in isolation) and for which the objective is to find a non-homogeneous partition that yields an aggregate MDP and corresponding optimal value function that is "close" to the original MDP and its optimal value function. The preliminary experimental results in this paper indicate that our algorithm achieves good performance using a compact representation encoded in terms of a non-homogeneous partition that is a fraction of the size of the representations required by other approaches.

Finding a non-homogeneous partitioning algorithm with a richer representation for blocks (such as ADDs) seems promising, and this task remains as our future work.

References

[Bertsekas and Castañon, 1989] Dimitri P. Bertsekas and David A. Castañon. Adaptive aggregation for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, 1989.

[Bertsekas and Tsitsiklis, 1996] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Boutilier and Dearden, 1996] Craig Boutilier and Richard Dearden. Approximating value trees in structured dynamic programming. In *Proceedings ICML-96*, 1996.

[Boutilier *et al.*, 1995] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings IJCAI-95*, pages 1104–1111, 1995.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11, 1999.

[Dean and Givan, 1997] Thomas Dean and Robert Givan. Model minimization in Markov decision processes. In *Proceedings AAAI-97*, 1997.

[Dean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa. A Model for Reasoning about Persistence and Causation. *Computational Intelligence*, pages 143–150, 1989.

[Forbes *et al.*, 1995] Jeff Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proceedings IJCAI-95*, 1995.

[Givan *et al.*, 2000] Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122:71–109, 2000.

[Goldsmith and Sloan, 2000] Judy Goldsmith and Robert H. Sloan. The complexity of model aggregation. In *Proceedings AIPS-2000*, 2000.

[Gordon, 1995] Geoffrey J. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-103, School of Computer Science, Carnegie Mellon University, 1995.

[Hoey *et al.*, 1999] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings UAI-99*, 1999.

[Kim, 2001] Kee-Eung Kim. *Representations and Algorithms for Large Stochastic Planning Problems*. PhD thesis, Brown University, 2001. In preparation.

[Koller and Parr, 1999] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings IJCAI-99*, 1999.

[Munos and Moore, 1999] Rémi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings IJCAI-99*, 1999.

[Singh and Yee, 1994] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16:227–233, 1994.

[Somenzi, 1998] Fabio Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, 1998.

[St-Aubin *et al.*, 2000] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *Proceedings NIPS-2000*, 2000.

[Tsitsiklis and Van Roy, 1996] John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[White and Eldeib, 1994] Chelsea White and Hany Eldeib. Markov decision processes with impricise transition probabilities. *Operations Research*, 42(4), 1994.