Solving Factored MDPs Using Non-homogeneous Partitions

Kee-Eung Kim
IT Research Center
Samsung SDS
159-9 Gumi-Dong Bundang-Gu
Seongnam-Si Gyeonggi-Do, 463-810
Korea
kekim@samsung.co.kr

Thomas Dean
Department of Computer Science
Brown University
Providence, RI 02912-1910
U.S.A.
tld@cs.brown.edu

August 4, 2002

Abstract

We present an algorithm for aggregating states in solving large MDPs (represented as *factored* MDPs) using search by successive refinement in the space of non-homogeneous partitions. Homogeneity is defined in terms of stochastic 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. In addition to investigating more exhaustive local search methods we explore the use of techniques derived from research on discretizing continuous state spaces. Finally, we compare the results from our algorithms which search in the space of non-homogeneous partitions with exact and approximate algorithms which represent homogeneous and approximately homogeneous partitions as decision trees or algebraic decision diagrams.

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.* [4]). Since the number of states in these factored MDPs (FMDPs) is exponential in the number of fluents, traditional iterative methods such as value iteration and policy iteration (Puterman [29]), which require explicit enumeration of states, are not effective.

One approach to solving MDPs that have a very large number of states involves aggregating — in the sense of "combining together" — states that behave similarly with respect to those aspects of MDPs that affect the value of plans, namely, rewards and action dynamics. If the operative notion of similarity constitutes an equivalence relation, then the relation induces a partition of the set of states and the blocks of the partition correspond to the equivalence classes of the equivalence relation. Under certain circumstances, these blocks define a set of *aggregate states* that form the state space for a smaller MDP which is equivalent to the original MDP in the sense that optimal plans — called *policies* in the parlance of the theory of MDPs — defined on the smaller MDP can be trivially extended to optimal policies defined with respect to the original MDP.

Once we have partitioned the state space according to an appropriate notion of equivalence and thereby defined an equivalent MDP, we can solve it using more traditional methods if the number of blocks is small enough. However, theoretical results show that even if we assume that such a small (polynomial in the size of the description of the original MDP) equivalent MDP exists, the problem of finding it is NP-hard. To make matters worse, we can produce an MDP such that the size of the smallest equivalent MDP is exponential in the size of the description length of the original MDP, and remains of exponential size even if we relax our requirements by adopting a suitable notion of "approximately equivalent". So, faced with the possibility that either the original MDP has no small equivalent MDP or that finding such an MDP is intractable, advocates of state aggregation techniques resort to heuristic search.

In this paper, we propose an algorithm that solves FMDPs by constructing and refining *non-homogeneous* partitions of the state space. Before we explain what a non-homogeneous partition is, it is useful to describe first what a *homogeneous* partition is. In a homogeneous partition, any two states in a block have the same reward and the same (block-to-block) transition probabilities with respect to other blocks. A precise definition will follow, but roughly speaking, the notion of homogeneity is similar to the state equivalence relation used in the classical FSA minimization algorithm (Hopcroft and Ullman [19]). A homogeneous partition induces a reduced-state-space MDP that is equivalent to the original FMDP. In practice, however, we observed that methods using homogeneity as a guiding principle (whether explicitly or implicitly) for aggregating states often built partitions with a number of blocks exponential in the number of fluents. This observation led us to consider other methods for aggregating states.

In contrast to homogeneous partitions, non-homogeneous partitions allow variation among the states in a block with regard to block-to-block transition probabilities. The method of aggregation is not driven directly by state equivalence. From a non-homogeneous partition, we can construct an aggregate MDP by averaging the transition probabilities and the rewards within blocks. The aggregate MDP so constructed may not be equivalent to the original FMDP and one contribution of this paper is to provide some mathematical insights into how optimal policies with respect to the aggregate MDPs induced from non-homogeneous partitions compare with optimal policies for the orginal FMDP.

The algorithm described in this paper solves FMDPs by successive refinement in the space of non-homogeneous partitions. The algorithm is initialized with a partition consisting of relatively few blocks which is iteratively refined by splitting blocks to obtain successively finer and finer 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 FMDP 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 FMDP. 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 Representation of MDPs

For decades, the MDP has been an effective mathematical framework for modeling stochastic planning problems (Puterman [29]). The MDP framework assumes that all the information in the environment *directly* relevant to decision making is captured in the state space — the agent directly observes the state of the environment. Furthermore, the dynamics of the environment is assumed Markovian — the agent makes a series of decisions at discrete time steps as the environment evolves, and the state of the environment at the next time step is independent of the past history given that the agent knows the state at the current time step. Although there are variants of MDPs that enable us to formulate continuous time decision making, we concentrate on the discrete case since it is more applicable to modeling classical planning problems. The following is the formal definition of Markov Decision Processes:

Definition 1 (MDP) *An* MDP *is four-tuple* $M = \{S, A, T, R\}$ *in which*

- S *is the set of states,*
- A *is the set of actions,*
- T *is the set of transition probabilities:*

$$T(s_t, a, s_{t+1}) = P(s_{t+1}|s_t, a)$$

where s_t *and* s_{t+1} *denote the state of the environment at time* t *and* $t+1$*, respectively, and*

- $R : S \times A \to \Re$ *is the reward function.*

In terms of computational complexity, solving MDPs is known to be in the class P-complete (Papadimitriou *et al.* [28]), so we know that there exists a polynomial time algorithm that outputs the optimal policy. However, the traditional tabular and matrix representations for MDPs store the probabilities and the rewards explicitly by enumerating all of the possible states and actions. It is often infeasible to do this sort of exhaustive enumeration in real world problems where we are confronted by a large number of states. As a result, researchers have been looking into representations that encode the reward function and transition probabilities as a composite of simpler, more compact functions (or *factors*) to achieve economy in representation. In this section, following the definition of Boutilier *et al.* [6], we formally describe the Factored MDP (FMDP), which is widely used as a general-purpose factored representation for an MDP with a discrete, n-dimensional state space:

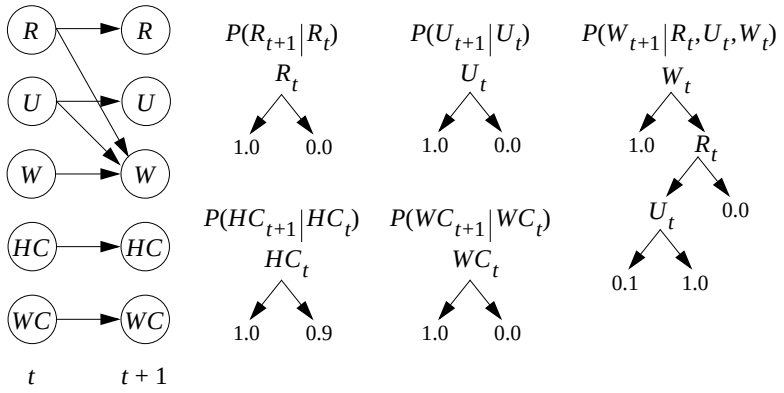

Figure 1: Graphical representation of an FMDP describing a toy problem of a coffee robot domain.

Definition 2 (FMDP) *An* FMDP *is a four-tuple,* $M = \{\vec{X}, A, T, R\}$*, whose components are defined follows:*

- $\vec{X} = [X_1, \ldots, X_n]$ *is the set of fluents that are used to describe the individual states. An assignment of values to each and every fluent defines the state. We use* Ω_{X_i} *to denote the set of values that fluent* X_i *can take —* Ω_{X_i} *is the sample space of* X_i *when* X_i *is treated as a random variable. Thus, the state space is* $\Omega_{\vec{X}} = \prod_i \Omega_{X_i}$*. We use the lowercase letter* $\vec{x} = [x_1, \ldots, x_n]$ *to denote a particular instantiation of the fluents.*

- 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:*

$$T(\vec{x}_t, a, \vec{x}_{t+1}) = \prod_{i=1}^{n} P(x_{i,t+1} | \mathrm{pa}(x_{i,t+1}), a) \tag{1}$$

  where $\mathrm{pa}(X_{i,t+1})$ *denotes the set of* parent *fluents that directly influence the value of* $X_{i,t+1}$*. Note that* $\forall i, \mathrm{pa}(X_{i,t+1}) \subseteq \{X_{1,t}, \ldots, X_{n,t}\}$*.*

- $R : \vec{X} \times A \to \Re$ *is the reward function.*

The FMDP representation is said to be a factored representation mainly due to Equation 1. The transition probabilities are factored into a product of probabilities of individual fluents conditioned on subsets of the fluents at the previous time step.

A common way of representing an FMDP is using a *graphical model*. Figure 1 shows the graphical model representing the dynamics of an FMDP example in which we execute an action in a domain with 5 boolean variables. The coffee robot has to deliver a cup of coffee to the robot's owner 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 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 the robot's owner wanting coffee (WC). Note that the probabilistic effect of each fluent at time

$t+1$ is conditioned on, or in other words influenced by, a small number of fluents at time t. We call them the parent fluents of a given fluent, and use the notation $\mathrm{pa}(X_{t+1})$ for the parents of fluent X_{t+1}. For example, the set of parent fluents for W_{t+1} for the above example is $\mathrm{pa}(W_{t+1}) = \{R_t, U_t, W_t\}$.

If we use tables for the conditional probabilities, the size of a table is exponential in the number of parent fluents. In the FMDPs we consider, the conditional probabilities are stored as decision trees, which are called conditional probability trees (CPTs) (Boutilier *et al.* [6]), rather than tables for the sake of savings in the size of the representation. CPTs exploit the regularity in the conditional probabilities, namely, some of the parent fluents become independent of the child fluent given a partial set of assignments for the parent fluents. We complete the description of the state dynamics by specifying CPTs for each fluent and action. The reward function R and initial probabilities I are also represented in terms of decision trees. Concerning a detailed discussion on using decision trees for representing conditional probabilities, Boutilier *et al.* [7] show how CPTs can be used to exploit a particular form of conditional independence, which they call *context-specific independence*, often present in Bayesian networks. Similar representations have been used to model Markovian processes with factored transition probabilities, such as the Two-stage Temporal Bayesian Network (2TBN) (Dean and Kanazawa [13]) and the Dynamic Bayesian Network (DBN) (Forbes *et al.* [14]).

In this paper, we concentrate on finding an optimal policy for an FMDP that maximizes expected total discounted reward under infinite horizon. A policy $\pi : \Omega_{\vec{X}} \rightarrow A$ is a mapping from the states to the actions. Given a policy, the expected total discounted reward under infinite horizon with discount factor γ is defined by

$$V^{\pi}(\vec{x}) \equiv E\Big[\sum_{t=0}^{\infty} \gamma^t R(\vec{x}_t) | \vec{X}_0 = \vec{x}, \pi\Big],$$

where $\vec{X}_0$ specifies the state at time 0, and the expectation is taken by following the state dynamics induced by the policy π. By solving an FMDP, we mean finding a policy π that maximizes $V^{\pi}(\vec{x})$ for all $\vec{x} \in \Omega_{\vec{X}}$.

3 Homogeneous Partitioning Algorithms

In this section, we review some of the existing algorithms for partitioning the state space in FMDPs. We categorize these algorithms under homogeneous partitioning algorithms since the states are aggregated based on some "equality" criterion. Note that the number of states in an FMDP is exponential in the number of fluents. These algorithms use a variety of representations to achieve their economy of representation.

Boutilier *et al.* [6] present their structured value iteration algorithm which proceeds by aggregating states that have the same n-stages-to-go values. The n-stages-to-go value of state $\vec{x}$, which we denote $V^{(n)}(\vec{x})$, is obtained by dynamic programming:

$$V^{(n)}(\vec{x}) = \max_a \Big[R(\vec{x}, a) + \gamma \sum_{\vec{x}'} T(\vec{x}, a, \vec{x}') V^{(n-1)}(\vec{x}')\Big]$$

Structured value iteration uses decision trees to compactly represent $V^{(n)}$ for all n. They provide a method that constructs the decision tree for $V^{(n)}$ from the decision tree for $V^{(n-1)}$

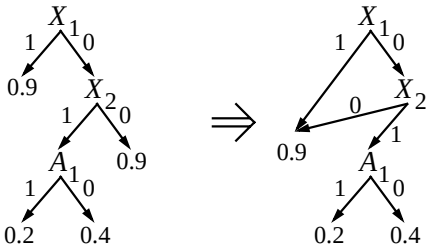

Figure 2: An example of the tree and the ADD representations of a value function.

without enumerating all the states in the FMDP by combining decision trees (grafting copies of decision trees to the leaves of other decisions trees) and simplifying the result. From the analysis of the classical value iteration algorithm (Puterman [29]), we know that the structured value iteration algorithm eventually converges to the optimal value function. Note that in some problems, the size of the decision tree for $V^{(n)}$ can still explode. The approximate structured value iteration algorithm by Boutilier and Dearden [5] uses a decision tree pruning technique to reduce the size of decision trees representing value functions. Pruning the decision tree can be seen as relaxing the condition for aggregation. Instead of aggregating the states that have exactly the same n-stages-to-go values, we now aggregate the states whose n-stages-to-go values differ by at most, say, ϵ. Later work on the SPUDD (Hoey *et al.* [18]) and APRICODD (St-Aubin *et al.* [32]) algorithms extends the structured value iteration and approximate structured value iteration algorithms by employing algebraic decision diagrams (ADDs) (Bahar *et al.* [1]) to represent value functions. Figure 2 shows an example of a value function represented as a tree and as an ADD.

The FMDP minimization algorithm by Dean and Givan [11] takes a more direct approach in terms of aggregating states that behave the same but less direct in terms of computing optimal value functions. The algorithm first finds a small equivalent MDP and then solves it using classical methods. Structured value iteration will generally find smaller partitions because the guiding notion of state equivalence at the n-th iteration is defined in terms of a subset of the set of all actions possible in each state, namely, the actions specified by the optimal n-stages-to-go policy.

The FMDP minimization algorithm uses the notion of *stochastic bisimulation equivalence* for FMDPs. It is an extension of the state equivalence relation for minimizing Finite State Automata (FSA) to that of probabilistic FSAs. The FMDP minimization algorithm partitions the state space into *stable* blocks. 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.

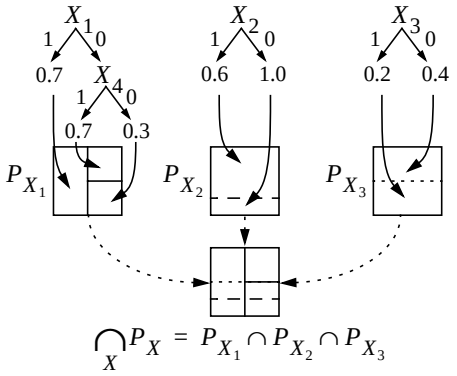

Figure 3: An illustration of the SPLIT operator.

The FMDP minimization algorithm uses the SPLIT operator to generate a refinement at each iteration. The SPLIT operator is defined as follows: for each pair of blocks (B, C) in the partition, check whether C is stable with respect to B for each action. If C is not stable with respect to B for some action a, replace C by a set, $\{C_i | 1 \leq i \leq n\}$, of sub-blocks such that each sub-block, C_i, is stable with respect to B and a and $\{C_i | 1 \leq i \leq n\}$ partitions C. We denote the resulting partition of C by SPLIT(B, C, P, a) and say that C is *refined*. Depending on how SPLIT is defined it may be possible to merge some blocks of SPLIT(B, C, P, a) that, due to various numerical coincidences, can be combined while maintaining the homogeneity with respect to block B. This merging step can be relaxed or omitted; however, if we want to compute the minimal model of a given FMDP, we must merge every block that can be merged without violating the homogeneity.

Figure 3 shows how SPLIT(B, C, P, a) operator works (without the merging step). We first collect CPTs for the fluents used in the description for block B. In the figure, we assume that B is labeled as $(X_1 \wedge X_2 \wedge X_3)$. CPTs refine block C according to their contribution to the conditional probability, and the figure shows how X_1, X_2 and X_3 differently refine C. The refined blocks are shown as P_{X_1}, P_{X_2} and P_{X_3}. The combination of all the refined blocks is shown at the bottom, which is $\bigcap_X P_X$.

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.

Dean *et al.* [12] 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

$$\exists c \in [0, 1] \text{ such that } \forall \vec{x}_t \in C, |T(\vec{x}_t, a, B) - c| \leq \epsilon. \tag{2}$$

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. By relaxing the requirement to be *approximately equal with error within* ϵ, in some cases we obtain a smaller partition (fewer blocks) than if we were to require strict homogeneity.

In theory, model minimization followed by traditional value iteration on the minimal

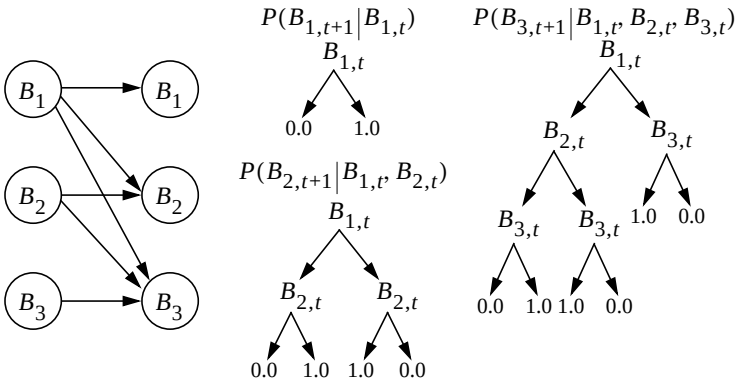

Figure 4: An FMDP with one action representing a 3-bit counter. Note that CPTs are still of size polynomial in the number of fluents since they are skewed trees. There is only one action in the FMDP. The size of the coarsest homogeneous partition is $2^3 = 8$. We can generalize this example to an n-bit counter. See Boutilier *et al.* [6] for a similar example with n actions.

model could outperform structured value iteration if the cost of computing the minimal model could be amortized over several iterations of structured value iteration; in practice, however, structured value iteration appears to be more efficient due to its strategy for adapting the partition implicit in the n-stages-to-go value function to derive the $(n+1)$-stages-to-go value function.

Bounding the error in the optimal value function computed from an ϵ-homogeneous partition is discussed in the work of Singh and Yee [30], White and Eldeib [33] and Givan *et al.* [15]. The approximate structured value iteration and APRICODD algorithms follow similar analyses.

Solving FMDPs is computationally difficult. The corresponding decision problems have been shown to be EXP-hard for the finite-horizon case (Littman *et al.* [24]) and undecidable for the infinite-horizon case (Madani *et al.* [25]). The computational intractability persists even for problems involving ϵ-homogeneity, and it is known, for example, that computing an ϵ-homogeneous partition is coNP^{PP}-hard (Goldsmith and Sloan [16]).

4 Non-homogeneous Partitioning Algorithms

There are two main questions remaining to be answered concerning the homogeneous partitioning algorithms described in the previous section. First, what class of functions (in the form of logical formulas) should we use to describe blocks and partitions? The formulas for blocks, which we call block formulas, implicitly describe the set of all (primitive) states in a block without explicitly enumerating these states. A restricted or less expressive language for block formulas generally results in simpler operations for splitting and merging blocks. However, if we restrict the representational power of block formulas too much, we may end up with partitions consisting of large numbers of blocks, perhaps even the degenerate partition consisting of all singleton sets. If on the other hand we adopt a more expressive language such as general boolean functions, management of block formulas for SPLIT operations is

NP-hard while we are guaranteed to be able to represent the coarsest homogeneous partition which may, if we're lucky, be small.

Second, what if the coarsest homogeneous partition is still exponentially large compared to the description size of the FMDP? We can indeed build a simple example showing that this is possible. Figure 4 shows a case in which each block of the coarsest homogeneous partition contains only one atomic state. Using the definition of ϵ-stability and ϵ-homogeneous partition (Equation 2) helps, but it is not hard to show that there are problems with threshold δ where the size of the partition is large for $\epsilon < \delta$ and the partition collapses into a small number of blocks for $\epsilon = \delta$. Note that Figure 4 also shows this kind of behavior since it is a deterministic domain in which all the transition probabilities are either 0 or 1, and any $\epsilon \in [0.5, 1]$ collapses the partition into three blocks and any $\epsilon \in [0, 0.5)$ does not result in any change.

Based on the above observation, we present an aggregation algorithm for FMDPs that is not based on the notion of stochastic bisimulation equivalence. We are not interested in finding a minimized MDP that is equivalent to the original FMDP. We are only interested in finding a reasonably sized partition of the state space so that it leads us to a policy close to the optimum.

The algorithm iteratively refines the current partition as the FMDP minimization algorithm does. However, as mentioned above, our criterion for splitting a block is different from making the block stable or obtaining a homogeneous partition. Since we discard the notion of stochastic bisimulation equivalence, a block can be split in many ways. At each iteration, the algorithm makes the decision of *which* block to split and *how* to split the block. Note that if we allow a general propositional formula for representing blocks in a partition, there are exponentially many ways to split a block. In this work, we assume that the partition is represented as a decision tree. In other words, the block formulas are simple conjunctions. We also restrict our split to be dependent on only one fluent in the FMDP. The number of ways we can split a block B in the current partition now becomes $n - |\vec{X}_B|$ where n is the total number of fluents in the FMDP and $\vec{X}_B$ is the set of fluents that are used to describe block B, which we refer to as the relevant fluents. In other words, a block can be split as many ways as the number of fluents not relevant to the block.

Now the crucial question becomes, how do we select the best block-fluent pair to split? In attempting to answer this question, we first define how we construct an MDP from a non-homogeneous partition. Simply put, the resulting MDP is an aggregated version of the original MDP with rewards and transition probabilities obtained by averaging over blocks. The following definition makes this notion a bit more precise.

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*

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

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

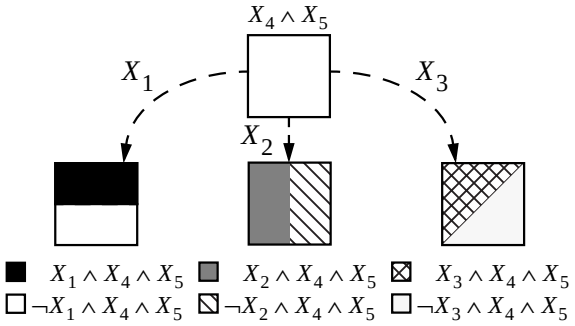

Figure 5: Illustration of the CHOP operator on block C given as $X_4 \wedge X_5$ with respect to fluents X_1 (left), X_2 (center) and X_3 (right). The block C is split by the possible values of the fluent. When split by fluent X_1 for example, we obtain two blocks, colored black and white respectively as shown on the left. Note that the blocks resulting from the CHOP operator are not stable in general.

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* $\pi^*(\vec{x}) = \pi^*(\vec{x}')$.

In some sense, this is a good model of the original MDP given an arbitrary partition. Since the partition is non-homogeneous, if we examine each state in a block, we make contradictory observations of reward and transition probabilities. The best model that explains the original MDP is the average of these observations not given any prior knowledge of the model.

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 $\mathrm{CHOP}(P, C, X)$. Figure 5 shows how the CHOP operator splits up the block C. Letting n be 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 $M_{P'}$ according to Definition 3 and calculates the optimal value function $V_{P'}^*$ of $M_{P'}$. How should we find the best refined partition? In Theorem 2, we show that for any partition P, the optimal value function of M_P, which we define as V_P^*, is within a bounded distance from the best approximation to the true value function V^*. First, we introduce a definition and a theorem from Gordon [17] that we will use in the proof. Theorem 1 shows that when the value function is approximated through an averager (Definition 4), we can bound the approximation error. In the following, we make use of the fact that a value function can be represented as a vector whose components map states to real numbers.

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* $\forall i, \beta_i + \sum_j \beta_{ij} = 1$.

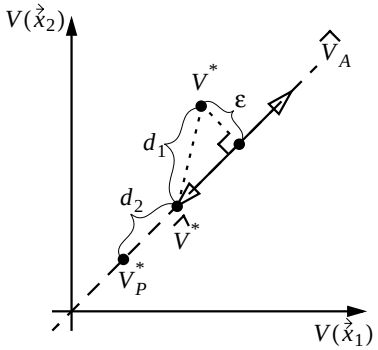

Figure 6: Value functions for an MDP constructed from a non-homogeneous partition. The figure illustrates an example in which the MDP has two states and the partition has one block. The dotted line is the set of representable approximate value functions. $d_1 \leq 2(1 + \frac{\gamma}{1-\gamma}\epsilon)$ and $d_2 \leq \frac{\|LV_P^* - V_P^*\|_\infty}{1-\gamma}$.

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

$$LV(\vec{x}) \equiv \max_a \big[R(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \Omega_{\vec{X}}} T(\vec{x}, a, \vec{x}')V(\vec{x}')\big] \tag{3}$$

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 $M = (\vec{X}, A, T, R)$ 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*

$$\|V^* - V_P^*\|_\infty \leq 2\Big(1 + \frac{\gamma}{1-\gamma}\Big)\epsilon + \frac{\|LV_P^* - V_P^*\|_\infty}{1-\gamma} \tag{4}$$

where $\epsilon \equiv \min_{V_P} \|V^* - V_P\|_\infty$ and LV_P^* is the Bellman backup (Equation 3) of V_P^*.

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

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

and

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

We define another value function

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

To calculate the maximum distance between V^* and $\widehat{V}^*$, we define an averager A_P (Definition 4) as follows:

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

Note that the fixed points of A_P are vectors with the constraint that the values of two components are the same if these two components belong to the same block. For the example shown in Figure 6, the straight dashed line satisfying $\widehat{V}_A(\vec{x}_1) = \widehat{V}_A(\vec{x}_2)$ is the set of fixed points for A_P if we assume that $\vec{x}_1$ and $\vec{x}_2$ belong to the same block. More formally, any fixed point $\widehat{V}_A$ satisfies

$$\widehat{V}_A(\vec{x}) = \widehat{V}_A(\vec{x}') \text{ for } C \in P \text{ such that } \vec{x} \in C \text{ and } \vec{x}' \in C.$$

Since P is a refinement of the reward partition, $\widehat{V}^*$ is the fixed point of the mapping $L \circ A_P$, and from Theorem 1, we have that

$$\|\widehat{V}^* - V^*\|_\infty \leq 2(1 + \frac{\gamma}{1-\gamma})\epsilon \tag{5}$$

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 $\vec{x} \in C$,

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

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

$$\|\widehat{V}^* - V_P^*\|_\infty \leq \frac{\|L V_P^* - V_P^*\|_\infty}{1-\gamma} \tag{6}$$

By combining Equation 5 and Equation 6, we have shown that

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

□

1. Input: the number of iterations N, the 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 $M_{P'}$ in which P' is the same as P except C is replaced by $\mathrm{CHOP}(P, C, X_i)$.
3. For each P', compute $V_{P'}^*$ and then select $P^* \equiv \arg\max_{P'} \|V_{P'}^* - V_P^*\|$.
4. If Steps 2 and 3 have been run N times, halt and output $\pi_{P^*}^*$.
5. Set $P = P^*$ and go to Step 2.

Figure 7: The algorithm for finding a non-homogeneous partition for an approximately optimal policy

There are a couple of observations worth making about the bound in the above theorem: First, the quantity $\epsilon \equiv \min_{V_P} \|V^* - V_P\|_\infty$ that appears in the first additive term in Equation 4 measures how fine the partition P is. ϵ is the minimum error that we can achieve by arbitrarily 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.

The second observation is that the quantity $\|LV_P^* - V_P^*\|_\infty$ 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 term serves as a finite contribution to the upper bound on the distance.

Given that the calculated value function V_P^* is a good approximation to the original value function V^* (in the sense that the distance is bounded), the algorithm selects the best refined partition given by the formula

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

The algorithm runs by iteratively refining the partition for a pre-determined number of iterations so that we end up with a fixed-size policy. Figure 7 lists the pseudo-code for the algorithm. Note that we can accelerate the algorithm if we can find a good heuristic for selecting the best block-fluent pair in Steps 2 and 3. In Section 6, we show some experimental results using a heuristic approach for choosing a fluent to split on rather than constructing and solving $M_{P'}$ for each block-fluent pair.

Note that constructing $M_{P'}$ from the refined partition P' can be done efficiently by reusing the parameters from M_P. The new transition probability matrix $T_{P'}$ has a lot of components that are the same as those of T_P. By reusing the components already calculated, we can construct $M_{P'}$ without recomputing the transition probabilities and rewards for all blocks. Assume that we perform $\mathrm{CHOP}(P, C, X_i)$ so that the block C splits into $|\Omega_{X_i}|$ blocks,

1. Input: the original partition P and the refined partition P' such that P' is the same as P except the block $C \in P$ is replaced by $\text{CHOP}(P, C, X_i) = \{C_1, \ldots, C_{|\Omega_{X_i}|}\}$.

2. For each action a, and each pair of blocks B and B' in partition P', we calculate the transition probability $T_{P'}(B, a, B')$ as follows:

   (a) If $B \notin \{C_1, \ldots, C_{|\Omega_{X_i}|}\}$ and $B' \notin \{C_1, \ldots, C_{|\Omega_{X_i}|}\}$, the transition probability is the same as T_P:

   $$T_{P'}(B, a, B') = T_P(B, a, B')$$

   (b) If $B \notin \{C_1, \ldots, C_{|\Omega_{X_i}|}\}$ and $B' = C_k$ for some k,

      i. Calculate $B \cap \bigcap_{X \in \vec{X}_{C_k}} P_X$ where $\vec{X}_{C_k}$ is the set of relevant fluents for block C_k and P_X is the partition induced by the CPT of fluent X. The set of relevant fluents for a block means the fluents used to describe the block.

      ii. From the result obtained in the previous step, average the probabilities contained in the split blocks weighted by the sizes of the blocks.

   (c) If $B = C_j$ for some j and $B' \notin \{C_1, \ldots, C_{|\Omega_{X_i}|}\}$,

      i. Calculate $C_j \cap \bigcap_{X \in \vec{X}_{B'}} P_X$.

      ii. From the result obtained in the previous step, average the probabilities contained in the split blocks weighted by the sizes of the blocks.

   (d) If $B = C_j$ for some j and $B' = C_k$ for some k,

      i. Calculate $C_j \cap \bigcap_{X \in \vec{X}_{C_k}} P_X$.

      ii. From the result obtained in the previous step, average the probabilities contained in the split blocks weighted by the sizes of the blocks.

3. Output $T_{P'}$.

Figure 8: The algorithm for calculating $T_{P'}$

$\{C_1, \ldots, C_{|\Omega_{X_i}|}\}$. Then we have

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

All we have to do is to recompute some of the transition probabilities in P' which are

$$T_{P'}(B, a, B') \quad \forall a \in A, \text{ and for all } B \in \{C_1, \ldots, C_{|\Omega_{X_i}|}\} \text{ or } B' \in \{C_1, \ldots, C_{|\Omega_{X_i}|}\}$$

Using either decision trees or ADDs, we can calculate new transition probabilities without explicitly enumerating all the states in B or B'. For example in calculating $T_{P'}(B, a, B')$, let us suppose that $B' = X_1 \wedge X_2 \wedge X_3$. First, note the CPT P_X^a for a fluent X and an action a induces a partition of the state space, so that every state in a block has the same contribution towards the transition probability with respect to the fluent X. Thus, by taking the intersection of CPTs for all the fluents used in describing block B', which in this case are

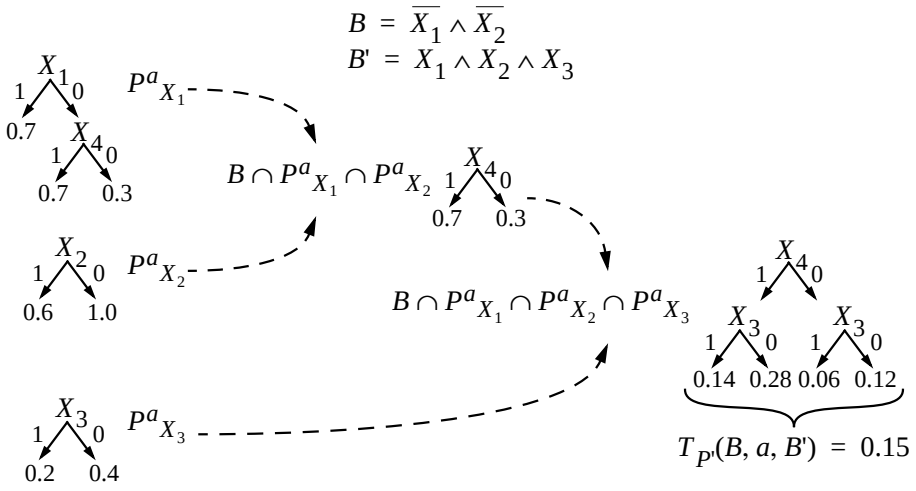

Figure 9: Calculating $T_{P'}(B, a, B')$ with decision trees.

X_1, X_2 and X_3, we make all of the distinctions necessary to compute the probability that we end up in block B'. When using decision trees, we calculate the intersection by repeatedly grafting CPTs and then simplifying to obtain the intersection.

Figure 8 shows the algorithm for calculating the new transition probabilities of $M_{P'}$. The core operation is the intersection of the partitions. Note that, when calculating $T_{P'}(B, a, B')$, all the intersections of the partitions are also intersected with the block B. When we use decision trees and graft CPTs to calculate the intersection, we can eliminate the subtrees in CPTs that do not intersect with the block B. For example, in Figure 9, we can eliminate the left subtree of X_1 in $P^a_{X_1}$, and the left subtree of X_2 in $P^a_{X_2}$ since the block representation of B tells us that X_1 and X_2 should be false. Assuming that the block being split is B, the size of a block represented by a terminal node v in the resulting decision tree is determined by

$$\frac{\prod_{X \notin \vec{X}_B} |\Omega_X|}{\prod_{Y \in \vec{X}_v} |\Omega_Y|}$$

where $\vec{X}_B$ is the set of relevant fluents used to describe the block B and $\vec{X}_v$ is the set of fluents that appear on the path from the root of the tree to the terminal node v. In our example, since all the terminal nodes are at the same level, we take the simple average, which yields 0.15. When using ADDs, the intersection is done by *multiplying* ADDs. The size of a block represented by a terminal node in the resulting ADD is determined by first constructing the ADD so that it represents the block (1 if the assignment of the fluents belongs to the block, and 0 otherwise), and summing over all values of the fluents in the FMDP, and then dividing by the size of the state space. The reward function $R_{P'}$ for $M_{P'}$ is obtained similarly.

The non-homogeneous partitioning algorithm still faces computational intractability problems. Although finding the best non-homogeneous partition is intractable, the intermediate results from the CHOP operator can be large — the CHOP operator is similar to the SPLIT operator. However, by averaging out the intermediate results, the non-homogeneous partitioning algorithm prevents the aggregate model from exploding. This is one of the main

differences from the homogeneous algorithms.

5 Adapting Heuristics from the Work on Solving Continuous-State MDPs

The algorithm presented in the previous section examines every block-fluent pair and then constructs and solves the resulting aggregate MDPs at each iteration. To avoid this exhaustive and time consuming process, we borrow some ideas from the research on using discretization techniques to solve continuous state-MDPs. The work by Munos and Moore [27] provides us with some ideas for how we might select a good block-fluent pair without examining every pair. Although their work concerns refining the discretization of continuous-state MDPs, their heuristic for deciding which portions of the grid discretizing the continuous space to split can be applied to our problem as well. They introduce two measures that help to predict which parts of the discretized state space are most important to allocate additional computational resources for gathering data and refining the current representation, and these measures extend naturally to splitting blocks in FMDPs:

- The *influence* of a state s on another state s' is a measure how much the change in the reward $R(s)$ contributes to the change in value function $V_P^*(s')$. It is calculated by

$$I(s,s') = \left[\gamma \sum_{s'' \in S_P} T_P(s, \pi_P^*(s), s'') I(s'', s')\right] + \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases} \tag{7}$$

  where S_P and T_P are the state space and the transition probabilities of M_P, the MDP constructed from non-homogeneous partition P, and π_P^* is the optimal policy of M_P.

- The *variance* of a state s is a measure of how uncertain we are about the value function for the state s. It is calculated by

$$\sigma^2(s) = e(s) + \gamma^2 \sum_{s' \in S_P} T_P(s, \pi_P^*(s), s') \sigma^2(s') \tag{8}$$

  where

$$e(s) = \sum_{s' \in S_P} T(s, \pi_P^*(s), s')[\gamma V_P(s') - V_P(s) + R_P(s)]^2.$$

Note that the states used in the above definitions are indeed blocks. The overall measure for splitting a block is defined by

$$Stdev\_Inf(s) = \sum_{s' \in S_d} \sigma(s) I(s, s')$$

where S_d is the set of states where we have different policies depending on the exact underlying model. Originally, Munos and Moore find the set S_d by comparing two optimal policies. Once they have the discretization of the continuous state space, they calculate

1. Input: the number of iterations N, the initial partition P of the state space (reward partition), and the optimal value function V_P^* for M_P.
2. For each block $C, C' \in P$, compute $I(C, C')$, the influence of C on C' (Equation 7), and $\sigma^2(C)$, the variance of C (Equation 8).
3. Compute V_{pes}^* (Equation 9) and V_{opt}^* (Equation 10) to obtain S_d, the set of states where the pessimistic optimal policy and the optimistic optimal policy differ.
4. Compute $Stdev\_Inf$ and then select $C^* \equiv \arg\max_C Stdev\_Inf(C)$.
5. For each fluent X_i, $1 \leq i \leq n$, compute $M_{P'}$ in which P' is same as P except C^* is replaced by $\text{CHOP}(P, C^*, X_i)$.
6. For each P', compute $V_{P'}^*$ and then select $P^* \equiv \arg\max_{P'} \|V_{P'}^* - V_P^*\|$.
7. If Steps 2-6 have been run N times, halt and output $\pi_{P^*}^*$.
8. Set $P = P^*$ and go to Step 2.

Figure 10: The non-homogeneous partitioning algorithm with Munos and Moore's heuristic.

the first optimal policy for the MDP constructed from that discretization. Also, since the discretization yields an approximation to the gradient of the value function, they calculate the second optimal policy by plugging in the gradients to the partial differential equation for the optimal policy. The latter is problematic for us to implement, since the value functions of FMDPs are discontinuous. Our approach for finding the set S_d involves calculating two different policies by assuming optimistic and pessimistic scenarios in the transition probabilities, using the techniques in bounded-parameter MDPs (Givan *et al.* [15]) and MDPs with imprecise parameters (White and Eldeib [33]).

Let a pessimistic optimal policy be a policy which assumes that the actual transition probabilities are unfavorable to the decision making agent, and an optimistic optimal policy be a policy which assumes the probabilities are favorable to the agent. S_d is the set of states in which the actions taken by the pessimistic optimal policy and the optimistic optimal policy differ.

Formally, let $T_\downarrow(s, a, s')$ be the minimum among the individual transition probabilities for starting from a state in the block represented by s and arriving at a state in the block represented by s' when executing action a. Let $T_\uparrow(s, a, s')$ be the maximum among the individual transition probabilities for starting from a state in the block represented by s and arriving at a state in the block represented by s' when executing action a. The pessimistic optimal policy is calculated from the pessimistic optimal value function, which is obtained by iterating the following equation

$$V_{\text{pes}}^{(n+1)}(s) = \max_a \left[R(s) + \gamma \min_{T(s,a,\cdot)} \sum_{s'} T(s, a, s') V_{\text{pes}}^{(n)}(s') \right] \tag{9}$$

where the minimization is done by choosing probabilities $T(s, a, s')$ that are between the minimum and the maximum of the individual transition probabilities:

$$\forall s', T_{\downarrow}(s, a, s') \leq T(s, a, s') \leq T_{\uparrow}(s, a, s')$$

and

$$\sum_{s'} T(s, a, s') = 1.$$

The optimistic optimal value function and the associated optimistic optimal policy are calculated by maximizing instead of minimizing:

$$V_{\text{opt}}^{(n+1)}(s) = \max_a \left[R(s) + \gamma \max_{T(s,a,\cdot)} \sum_{s'} T(s, a, s') V_{\text{opt}}^{(n)}(s') \right]. \tag{10}$$

In summary, when we calculate the new transition probabilities for a refined partition P' at each iteration, instead of taking the average of the items in the terminal nodes as in the previous version of the algorithm, we use the minimum ($T_{\downarrow}$) and the maximum ($T_{\uparrow}$) as the bound for $T_{P'}$, and calculate the optimistic and pessimistic optimal policies. We select the block which has the largest $Stdev\_Inf$ and examine splitting the block with respect to each fluent. Once the best block is chosen, we select the best fluent by constructing the MDP and comparing the value functions as in the previous version of the algorithm. Figure 10 shows the pseudo-code for the algorithm.

6 Experiments

The test problems used in our experiments are adopted from Hoey *et al.* [18] and involve domains with 6 to 17 binary fluents. The initial probabilities are given as a uniform distribution. For each domain, we show the quality of the policy derived from the non-homogeneous partition and the cumulative elapsed time at each iteration. We used ILOG CPLEX 6.5 for calculating optimal value functions of aggregate MDPs constructed from non-homogeneous partitions. For evaluating the policies and calculating the optimal value functions of the original FMDP, we used the CUDD package (Somenzi [31]) to implement structured versions of iterative algorithms similar to SPUDD (Hoey *et al.* [18]).

It is important to note that the problem sizes were chosen to be near the limit of what can be calculated and analyzed exactly so as to provide a gold standard for comparison purposes. It is the hope that our methods will scale better than existing methods based on decision trees and ADDs by virtue of there being able to find aggregate MDPs that yield approximate solutions to the original MDP that are significantly smaller than the representations required by the iterative decision tree and ADD methods.

Figure 11, Figure 12, Figure 13, and Figure 14 illustrate the results from the algorithm shown in Figure 7. For each domain except the COFFEE domain, we ran the algorithm for 100 iterations starting from the reward partition. This number of iterations represented a practical tradeoff given that it took a significant amount of time to show the intermediate results after each iteration (we had to exactly evaluate the policies generated by solving

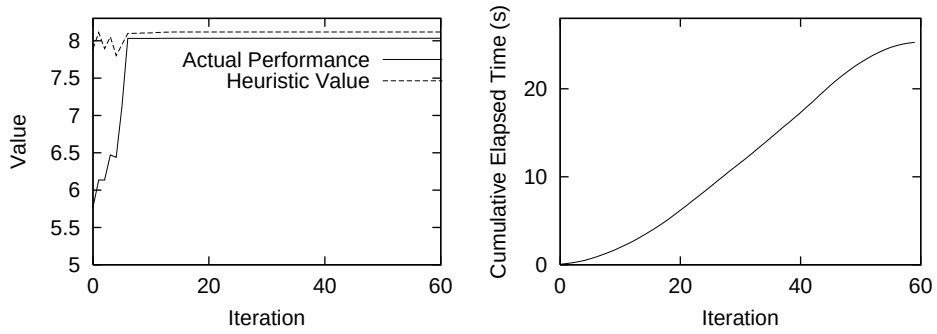

Figure 11: COFFEE domain (6 variables, 64 states). Starting from the reward partition with 4 blocks, the non-homogeneous partitioning algorithm found the optimal policy after 6 splits, totaling 10 blocks in the partition. The running time between the iterations diminishes in the end since the number of fluents we can choose for CHOP decreases. The optimal value is 8.12.

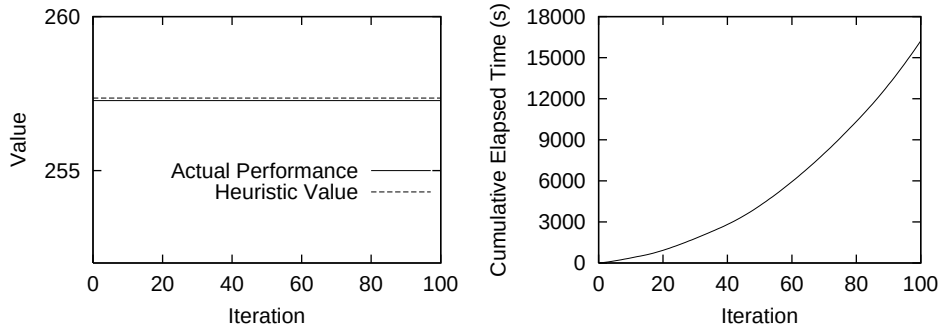

Figure 12: LINEAR domain (14 variables, 16,384 states). The optimal policy of the aggregate MDP from the reward partition (14 blocks) yields the optimal value, *i.e.*, the optimal policy was obtained without any split. The optimal value is 257.4.

the aggregate MDPs) and 100 iterations was enough that the algorithm produced policies with acceptable quality. In the case of the COFFEE domain, we ran the algorithm for 60 iterations since there were only 64 states and the reward partition had 4 blocks. For each figure, the graph on the left side shows the actual quality (the actual value of the optimal policy calculated from the aggregate MDP and then evaluated in the original MDP) and the heuristic value (the optimal value of the aggregate MDP) after each iteration. The values are obtained assuming a uniform starting distribution on the states. The graph on the right side shows the plot of cumulative elapsed time (in seconds) after each iteration. Note that in some domains, there is a small gap between the actual quality and the heuristic value even when the policy from the aggregate MDP reached the optimal quality. This is due to the fact that the evaluation is done through an iterative method with the stopping condition $\|V_{i+1} - V_i\| \leq \delta$, in which we set δ to 0.01.

Figure 12 shows the experimental results on the LINEAR domain. 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 one iteration. When the

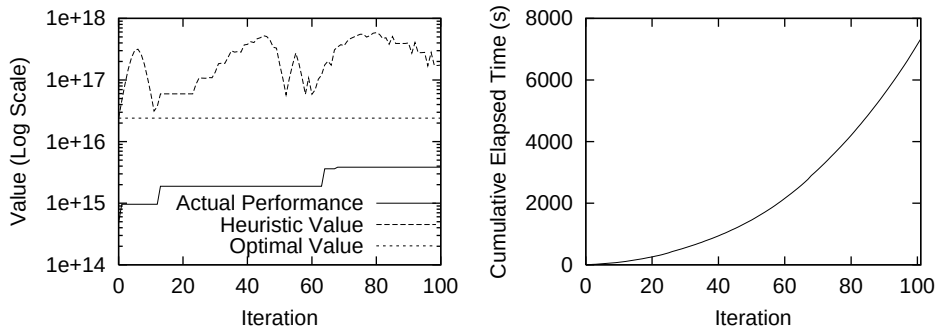

Figure 13: EXPON domain (12 variables, 4096 states). Starting from the reward partition (13 blocks), the non-homogeneous partitioning algorithm yields an approximately optimal policy after 100 splits (113 blocks). The optimal value is 2.4×10^{16}.

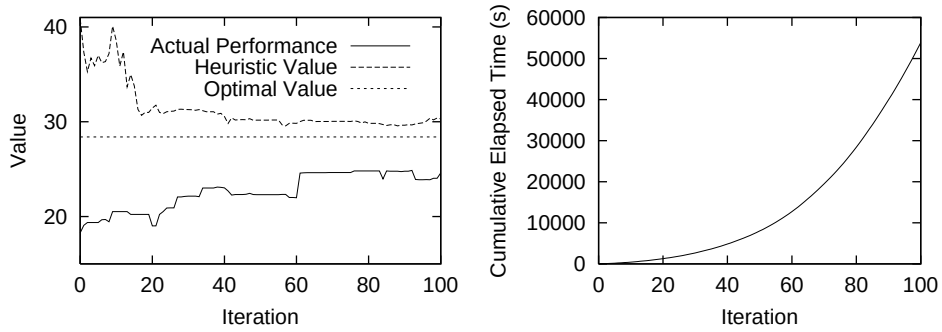

Figure 14: FACTORY domain (17 variables, 131,072 states). Starting from the reward partition (26 blocks), the non-homogeneous partitioning algorithm yields an approximately optimal policy after 100 splits (126 blocks). The optimal value is 28.4.

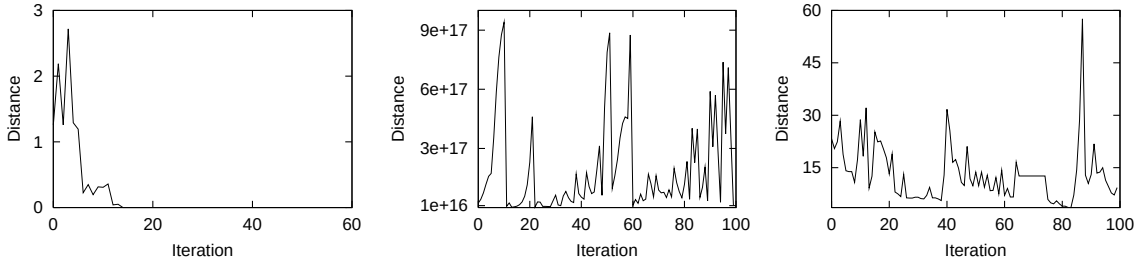

Figure 15: Distance plots between two value functions of successive aggregate MDPs ($\|V^*_{P^*} - V^*_P\|$ in Figure 7). From left to right: COFFEE, EXPON, FACTORY.

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 aggregate MDP is indeed optimal. In Figure 12 we continue splitting despite this knowledge simply to illustrate how the running time increases as a function of the number of iterations. Meanwhile, our implementation of SPUDD, which is not fully optimized, takes 43.58 seconds and produces 15 terminal nodes.

The optimal value function of the EXPON domain (Figure 13) has thousands of internal

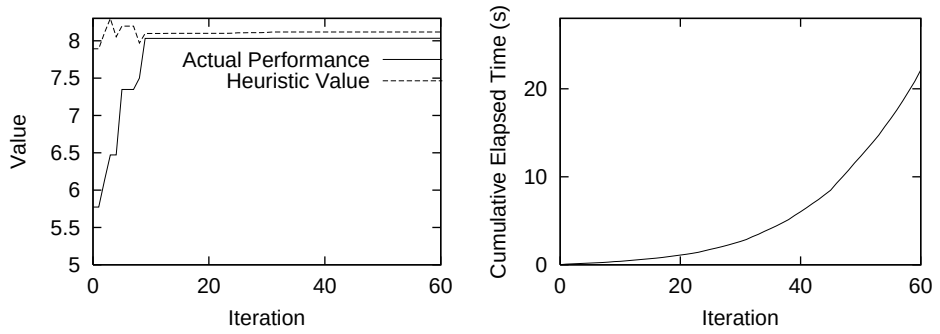

Figure 16: Munos and Moore's heuristic on the COFFEE domain (6 variables, 64 states). The optimal value is 8.12.

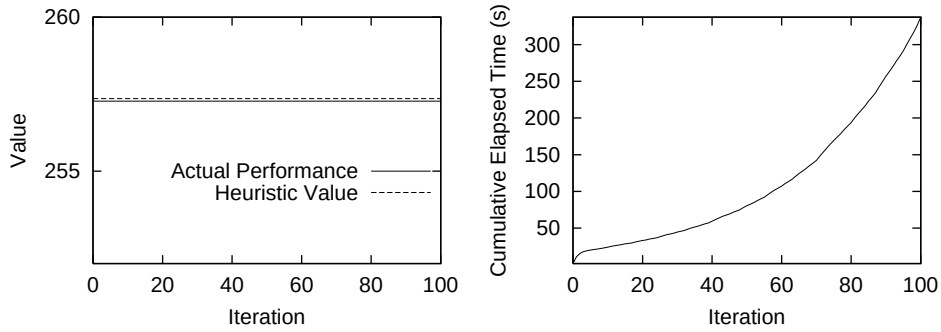

Figure 17: Munos and Moore's heuristic on the LINEAR domain (14 variables, 16,384 states). The optimal value is 257.4.

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 the 100th iteration is 10 times better than the initial policy. The size of the ADD representation for the optimal value function is also quite large in the case of the FACTORY domain (Figure 14). After 100 splits, totaling 126 blocks, the policy from the aggregate MDP has a value of 24.8 which is 87% of the optimal value.

Figure 15 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 actual quality of the aggregate MDP policy reaches the 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 find the optimal policy.

Munos and Moore's heuristic (Section 5) provides a significant speed up, but shows a small amount of loss in the quality for some domains. We show the results in Figure 16, Figure 17, Figure 18, and Figure 19.

As the basis for comparison of the heuristics we have shown, we also show the experimental results from random partitioning method. At each iteration, this method randomly selects the next refinement. Figure 21 shows the results from the random partitioning method

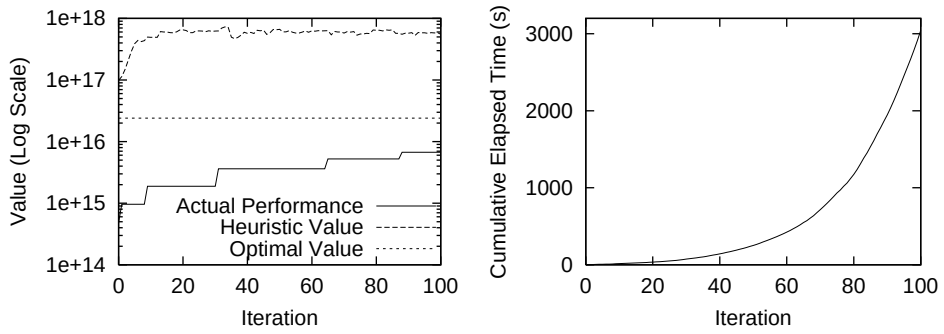

Figure 18: Munos and Moore's heuristic on the EXPON domain (12 variables, 4096 states). The optimal value is 2.4×10^{16}.

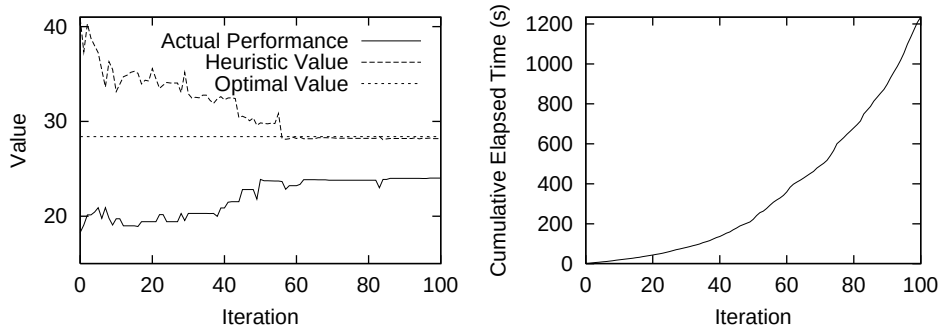

Figure 19: Munos and Moore's heuristic on the FACTORY domain (17 variables, 131,072 states). The optimal value is 28.4.

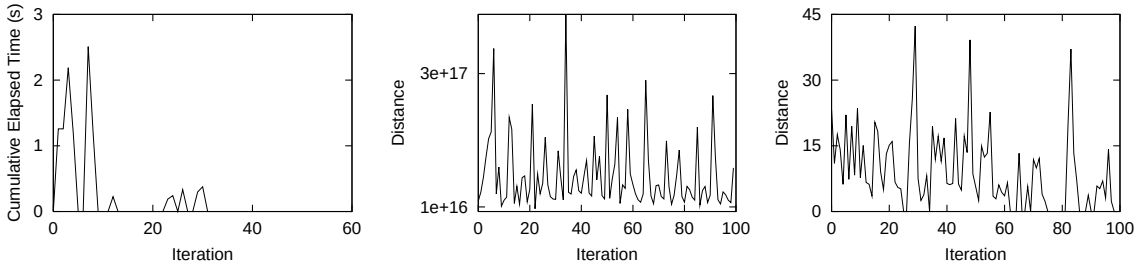

Figure 20: Distance plots between two value functions of successive aggregate MDPs ($\|V^*_{P^*} - V^*_P\|$ in Figure 7). From left to right: COFFEE, EXPON, FACTORY.

on the same domains.

Table 1 summarizes the results of three non-homogeneous aggregation methods using the best-split heuristic by examining every block-fluent pair (Figure 7), Munos and Moore's heuristic by computing the standard deviation of influences (Figure 10), and finally the random refinement at each iteration.

We also compare the quality of the policies from non-homogeneous partitioning algorithm to APRICODD [32]. Table 2 and Table 3 summarize the comparison of the two algorithms on the larger domains, EXPON and FACTORY. By trial and error, we tuned the pruning

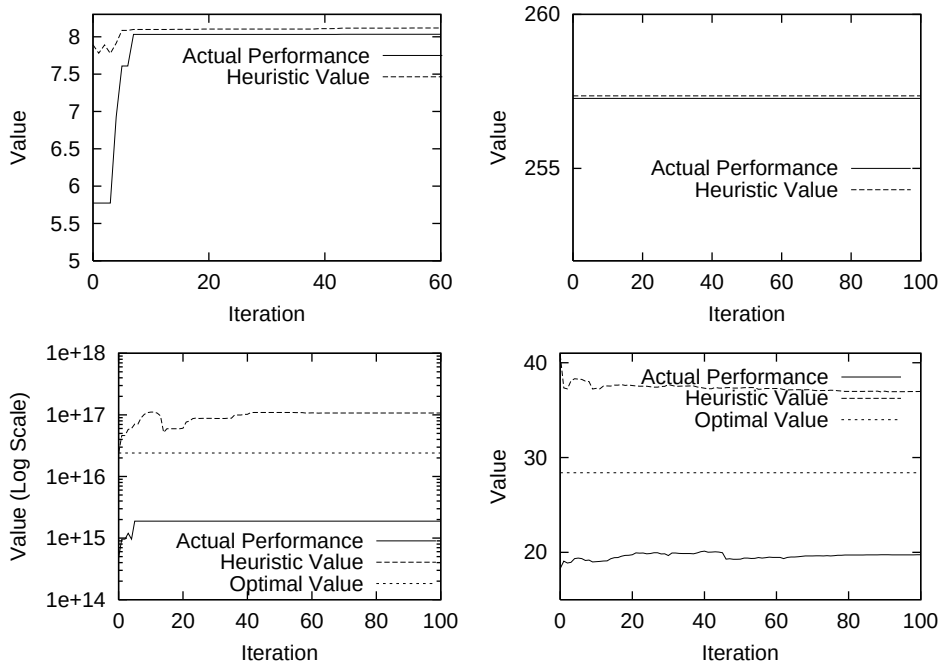

Figure 21: Random partitioning on COFFEE (upper-left), LINEAR (upper-right), EXPON (lower-left) and FACTORY (lower-right).

	EXPON		FACTORY	
	Quality	Time	Quality	Time
Best	16.0 %	7333 s	87.3 %	55402 s
Munos & Moore	28.0 %	3169 s	84.6 %	1260 s
Random	7.9 %	38 s	69.5 %	65 s

Table 1: Summary of results after 100 iterations on the EXPON and FACTORY domains. Quality is the ratio between the value of the optimal policy and that of the approximate policy assuming uniform starting distribution on states.

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 iterations. We used a "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 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 APRICODD to search in the space of a much richer representation for value functions — when converted to an ADD, a decision tree partition with 113 blocks generally becomes much smaller than an ADD with 246 nodes. In terms of the number of blocks, the non-homogeneous partitioning algorithm produces policies whose quality is comparable to that of the APRICODD algorithm. Note

	Quality	Blocks		Time
Non-homogeneous (Best)	16 %	113		7333 s
Non-homogeneous (Munos & Moore)	28 %	113		3169 s
	Quality	Nodes	Leaves	Time
APRICODD (0.4)	48 %	320	65	630 s
APRICODD (0.5)	23 %	246	33	73 s

Table 2: Comparison of the non-homogeneous partitioning algorithm and the APRICODD algorithm on EXPON domain. The number inside the parentheses is the pruning parameter determined by "sliding-tolerance" pruning. We also present the number of nodes and leaves (terminal nodes) in the ADD representation of the approximate value function from the APRICODD algorithm.

	Quality	Blocks		Time
Non-homogeneous (Best)	87 %	126		55402 s
Non-homogeneous (Munos & Moore)	85 %	126		1260 s
	Quality	Nodes	Leaves	Time
APRICODD (0.2)	67 %	342	73	920 s
APRICODD (0.3)	26 %	252	65	893 s

Table 3: Comparison of the non-homogeneous partitioning algorithm and the APRICODD algorithm on the FACTORY domain.

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.

7 Related Work

We have already reviewed the most directly relevant related work in some detail in Sections 3 and 6. In this section, we summarize the earlier connections and supply some additional but less directly relevant work.

Many algorithms for solving MDPs try to achieve efficiency by aggregating states that have approximately the same value. The class of representations includes simple (flat) sets (Bertsekas and Castañon [3]) (in this case, the authors are not averse to enumerating the states but still wish to reduce the polynomial overhead involved in evaluating policies), decision trees (Boutilier *et al.* [6], Boutilier and Dearden [5]), algebraic decisions diagrams (Hoey *et al.* [18], St-Aubin *et al.* [32]), linear combinations of simple basis functions (Koller and Parr [22, 23]) and neural networks (Bertsekas and Tsitsiklis [2]). The FMDP minimization algorithm by Dean and Givan [11] uses the notion of stochastic bisimulation equivalence to aggregate the states that behave the same. This algorithm was later extended to handle approximate bisimulation equivalence (Dean *et al.* [12]). We classified these algorithms as homogeneous partitioning algorithms in contrast to our algorithm which does not use the notion of equivalence in aggregating the states.

In the area of reinforcement learning, G-learning by Chapman and Kaelbling [10] and U-Tree algorithm by McCallum [26] use decision trees to aggregate states. Since these

algorithms assume no prior knowledge of the transition probabilities and the rewards, they first collect samples of transitions and rewards and do statistical tests to determine whether to differentiate (split) parts of the state space. In contrast, our approach assumes prior knowledge of the exact probabilities and rewards, but examining every state is prohibitive due to the huge state space.

The use of an averaged model in reinforcement learning is called the indirect method, meaning that we first collect samples of the transitions and rewards to construct a model of the domain and then solve the domain using the model (Bradtke and Barto [9], Kearns and Singh [21], Boyan [8]). Since indirect methods also assume no initial knowledge of the domain, they acquire samples for each parameter of the model.

8 Conclusions

We have presented a new algorithm for (approximately) solving FMDPs. The previous work on aggregation techniques for FMDPs focuses on notions of reward and transition equivalence for grouping the states. This equality condition essentially guarantees that handling a group of states as one (aggregate) state does not cost us precision in calculating value functions while solving FMDPs. The non-homogeneous partitioning algorithm takes a rather different approach.

Given any partition of the set of states of a target MDP we can define an aggregate MDP with states that correspond to the blocks of the partition and rewards and transition probabilities defined by averaging rewards and transition probabilities over the states in each block. Our method uses the optimal value functions for these aggregate MDPS to figure out how to refine partitions in order to obtain better approximations to the original MDP (at least with respect to policies that matter, i.e., those that are optimal or near optimal). Our new view of aggregating states also allows us to use some of the heuristics in the area of discretizing continuous state-space MDPs.

In the experiments, we have shown 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, that allows us to specify the desired size of the policy *a priori*. We can also extend the algorithm so that, at each iteration, it splits a pre-determined number of multiple blocks that score highest in our heuristics.

There are some additional issues concerning the non-homogeneous partitioning algorithm that we should address. First, we do not have the desirable property that we obtain monotonically better quality as we refine the partitions. We could enforce this property by calculating pessimistic optimal policies at each iteration. However, in the domains we tested on, the optimal policy obtained from the MDP constructed by averaging out the transition probabilities and the rewards had much better quality. We think that the above observation is also true for many real world problems. The pessimistic optimal policies often assume a

worst-case scenario that may not arise in the original FMDP. Second, we currently allow only simple conjunctions as the representation for the blocks in the partition. Extending the algorithm to allow for a richer representation of blocks may result in a combinatorial explosion in examining every possible split in the partition; however, heuristics for searching a richer class of partitions may be possible to achieve quality closer to that of the decision tree and ADD-based algorithms. Third, we do not have an easy way to detect that the non-homogeneous partitioning algorithm produces an optimal policy. As shown in Figure 15 and Figure 20, the distances between consecutive non-homogeneous partitions in the algorithm show large variations. Again, we could compare the value function V_P^* with the pessimistic optimal value function, but this is very conservative. We might also compare the Monte Carlo estimation of the value function under the policy with the result from the Kearns *et al.* [20] algorithm. We leave these problems for future work.

References

[1] R. Iris Bahar, Erica Frohm, Charles Gaona, Gary Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the International Conference on Computer-Aided Design*, 1993.

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

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

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

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

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

[7] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings UAI-96*, 1996.

[8] Justin A. Boyan. Least-squares temporal difference learning. In *Proceedings ICML-99*, 1999.

[9] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

[10] David Chapman and Leslie Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings IJCAI-91*, 1991.

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

[12] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings UAI-97*, 1997.

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

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

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

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

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

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

[19] John Hopcroft and Jeffrey Ullman. *Introduction to Automata theory, languages, and computation.* Addison Wesley, 1979.

[20] Michael Kearns, Yishay Mansour, and Andrew Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings IJCAI-99*, 1999.

[21] Michael Kearns and Satinder Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Proceedings NIPS-98*, 1998.

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

[23] Daphne Koller and Ronald Parr. Policy iteration for factored MDPs. In *Proceedings UAI-2000*, 2000.

[24] Michael Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9, 1998.

[25] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings AAAI-1999*, 1999.

[26] Andrew McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB'96)*, 1998.

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

[28] Christos Papadimitriou and John Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[29] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

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

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

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

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