In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99),* Stockholm, Sweden, August 1999

Computing factored value functions for policies in structured MDPs

Daphne Koller
Computer Science Department
Stanford University
koller@cs.stanford.edu

Ronald Parr
Computer Science Department
Stanford University
parr@cs.stanford.edu

Abstract

Many large Markov decision processes (MDPs) can be represented compactly using a structured representation such as a dynamic Bayesian network. Unfortunately, the compact representation does not help standard MDP algorithms, because the value function for the MDP does not retain the structure of the process description. We argue that in many such MDPs, structure is approximately retained. That is, the value functions are nearly *additive*: closely approximated by a linear function over factors associated with small subsets of problem features. Based on this idea, we present a convergent, approximate value determination algorithm for structured MDPs. The algorithm maintains an additive value function, alternating dynamic programming steps with steps that project the result back into the restricted space of additive functions. We show that both the dynamic programming and the projection steps can be computed efficiently, despite the fact that the number of states is exponential in the number of state variables.

1 Introduction

Over the past few years, there has been a growing interest in the problem of planning under uncertainty. *Markov decision processes (MDPs)* have received much attention as a basic semantics for this problem. An MDP represents the domain via a set of states, with actions inducing stochastic transitions from one state to another. The key problem with this type of representation is that, in virtually any real-life domain, the state space is quite large. However, many large MDPs have significant internal structure, and can be modeled very compactly if that structure is exploited by the representation. In *factored MDPs*, a state is described implicitly as an assignment of values to some set of *state variables*. A *dynamic Bayesian network* can then allow a compact representation of the transition model by exploiting the fact that the transition of a variable usually depends only on a small number of other variables. In a simple robotics example, the location of the robot at time $t+1$ may depend on its position, velocity, and orientation at time t, but not on what it is carrying. The momentary rewards can often also be decomposed: as a sum of rewards related to individual variables or small clusters of variables. In our robot example, our reward might be a sum of two subrewards: one associated with location (for getting too close to a wall) and one associated with the printer status (for letting paper run out).

While these representations allow very large, complex MDPs to be represented compactly, they do not help address the planning problem. Standard algorithms for solving MDPs require the representation and manipulation of *value functions* — functions from states to values. Since a state is a full instantiation of all state variables, the representation of the full value function is exponential in the number of state variables. Unfortunately, structure in a factored MDP does not, in general, guarantee any type of structure in the value function. Even some very simple MDPs with very compact transition models and fully decomposed rewards lead to value functions that have no usable structure whatsoever (see Section 2). Thus, algorithms that represent the value function exactly will be impractical for dealing with many large, structured domains.

One approach to controlling the size of the value function representation is to use a truncated value function that ignores some state variables. Several methods exist for detecting irrelevant state variables. A closely related approach uses approximate or abstract model that eliminates some state variables. (See Boutilier *et al.* [1999] for a survey of these approaches.) These methods have the common limitation that they can force different states to have the same value, producing a coarse-grained approximation to the value function with large plateaus of indistinguishable states.

Our work is based on the intuition that, while value functions might not be structured, there are many domains where they are "close" to having exploitable additive structure. Consider, for example, a stochastic version of a traditional planning task, where several subgoals can contribute to the overall success of the plan. Here, it is quite plausible that the value of a state is approximately linear in the set of subgoals achieved at that state (with more important subgoals having, perhaps, a higher weight). Clearly, this is only an approximation, as subgoals might interact in various ways; however, it may be a fairly good approximation. Furthermore, if some subgoals interact strongly, we can have value function components that depend on the status of several subgoals. Such value functions have a long history in *multi-attribute utility* *theory* [Keeney and Raiffa, 1976]; they also play a central role in influence diagrams [Howard and Matheson, 1984].

Based on this intuition, we propose a new approach to computing approximate value functions for structured MDPs. We restrict attention to value functions that are a linear combination of *local basis functions*, each of which depends only on a small subset of the state variables. As discussed above, we believe that, in many cases, such a value function can be a good approximation to the correct value function. In particular, unlike the coarse-grained approximations discussed above, this approximation can assign a distinct value to every state. However, it can still be represented compactly using a small number of parameters (the coefficients of the basis functions). We propose to use algorithms whose entire computation is restricted to value functions that are compactly represented in this form.

Our focus in this paper will be on the *value determination* task: computing the value function associated with a particular policy. This task can be solved using a simple algorithm that iterates through a *dynamic programming* step, gradually converging to the correct value function. Our algorithm augments this basic iteration with an additional projection step, where the value function resulting from each dynamic programming step is projected back down into the "allowable" space of value functions.

There are several key issues that arise in such an algorithm. First, we must show how the operations required by such an algorithm — dynamic programming and projection — can be executed efficiently. One of our main contributions is that we provide, for both of these steps, an efficient algorithm that depends upon the size and structure of the model and basis function representations, *not* upon the exponentially sized state space. We note that we can accomplish these operations efficiently despite the fact that each of the exponentially many states may have a distinct value. Second, we must show that the algorithm behaves reasonably: that it converges to a stable fixed point, hopefully one which is close to the true value function. By a careful choice of distance metric, we can show conditions under which the algorithm will converge to a solution which is not too far away from the best value function describable in the restricted space. Finally, we must show how to apply this algorithm to the task of evaluating policies in a factored MDP. This application is far from trivial, as policies in a large MDP may be very complex. We show that our algorithm allows effective policy evaluation for policies represented as a small set of conjunctive decision rules (e.g., decision trees); this class is arguably the most widely used class of compactly represented policies [Boutilier and Dearden, 1996; Dean *et al.*, 1997].

2 Preliminaries

We begin by introducing some of the basic concepts that we will be using. Our primary focus is on the value determination problem. Thus, for the time being, we will restrict attention to processes without a decision component. A *Markov Process (MP)* is defined as a triple (S, R, P) where: S is a set of states; R is a *reward function* $R : S \mapsto I\!R$, such that $R(s)$ represents the reward obtained by the agent in state s; P is a *transition model*, so that $P(s' \mid s)$ represents the probability of going from state s to state s'.

A Markov process is associated with a *value function* $V : S \mapsto I\!R$, where $V(s)$ is the total cumulative value that the agent gets if it starts at state s. We will be assuming that the MP has an infinite horizon and that future rewards are discounted exponentially with a discount factor γ. Thus, V is defined using the fixed point equation $V(s) = R(s) + \gamma \sum_{s'} P(s' \mid s)V(s')$. A simple iterative process can be used to perform *value determination*. We choose $V^0 \equiv 0$.[1] We then repeatedly execute a *dynamic programming (DP)* step:

$$V^{(t+1)}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s) V^{(t)}(s'). \qquad (1)$$

We use $\mathcal{T}$ to denote the operator that takes a value function $V^{(t)}$ and returns $V^{(t+1)}$. We call $V^{(t+1)}$ the *backprojection* of $V^{(t)}$. Repeated applications of $\mathcal{T}$ are known to converge to the true value function V^*.

In a factored MP, the set of states is described via a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $\mathrm{Val}(X_i)$. A state $\mathbf{x}$ defines a value $x_i \in \mathrm{Val}(X_i)$ for each variable X_i. The transition model is described as a *dynamic Bayesian network (DBN)*. Let X_i denote the variable X_i at the current time and X_i' the variable at the next step. The *transition graph* is a two-layer directed acyclic graph whose nodes are $\{X_1, \ldots, X_n, X_1', \ldots, X_n'\}$. In this paper, we will assume that all edges in this graph are directed from nodes in $\{X_1, \ldots, X_n\}$ to nodes in $\{X_1', \ldots, X_n'\}$. (This assumption can be relaxed, but the resulting algorithm becomes somewhat more complex. We defer details to a longer paper.) We denote the parents of X_i' in the graph by $\mathrm{Pa}(X_i')$. Each node X_i' is associated with a *conditional probability table (CPT)* $P(X_i' \mid \mathrm{Pa}(X_i'))$. The transition probability $P(\mathbf{x}' \mid \mathbf{x})$ is then defined to be $\prod_i P(x_i' \mid \mathbf{u}_i)$, where $\mathbf{u}_i$ is the value in $\mathbf{x}$ of the variables in $\mathrm{Pa}(X_i')$.

We also need to provide a compact representation of the reward function. As discussed in the introduction, we assume that the reward function is factored additively into a set of *localized* reward functions, each of which only depends on a small set of variables. In general, we say that a function f is *restricted* to a set of variables $\mathbf{C}$ if $f : \mathrm{Val}(\mathbf{C}) \mapsto I\!R$. Let $R_1, \ldots, R_r$ be a set of functions, where each R_i is restricted to a cluster of variables $\mathbf{W}_i \subset \{X_1, \ldots, X_n\}$, such that $R_i : \mathrm{Val}(\mathbf{W}_i) \mapsto I\!R$. The reward function associated with the state $\mathbf{x}$ is then defined to be $\sum_{i=1}^r R_i(\mathbf{x}) \in I\!R$. We note that, here and in the future, we use the following shorthand: if f is a function over some set of variables $\mathbf{Y}$, and $\mathbf{Y} \subset \mathbf{Z}$, we will use $f(\mathbf{z})$ to denote the value that f takes over the part of the vector $\mathbf{z}$ that corresponds to variables in $\mathbf{Y}$.

One might be led to believe that factored transition dynamics and rewards would result in a structured value function. Unfortunately, this usually is not the case. As we execute DP steps, the value function typically becomes more complex.

[1] We begin with $V^0 \equiv 0$ by convention. Any choice for V^0 will converge to the same fixed point.

Example 2.1: Assume that we have only a single reward function R that depends only on X_1; $V^{(1)} \equiv R$, so it also depends only on X_1. Consider the summation in the DP step for $V^{(2)}$:

$$\begin{aligned}
&\sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}) V^{(1)}(\mathbf{x}') \\
&= \sum_{x_1' \in \mathrm{Val}(X_1)} V^{(1)}(x_1') \sum_{\mathbf{w}' \in \mathrm{Val}(X_2', \ldots, X_n')} P(x_1', \mathbf{w}' \mid \mathbf{x}) \\
&= \sum_{x_1' \in \mathrm{Val}(X_1)} V^{(1)}(x_1') P(x_1' \mid \mathbf{x}) \\
&= \sum_{x_1' \in \mathrm{Val}(X_1)} V^{(1)}(x_1') P(x_1' \mid \mathbf{u}),
\end{aligned}$$

where $\mathbf{u}$ is the instantiation of $\mathrm{Pa}(X_1')$ in $\mathbf{x}$. Thus, $V^{(2)}$ depends on X_1 *and* on the entire parent set of X_1. Similarly, $V^{(3)}$ depends on the union of all of their parents, etc. ∎

In general, the value function will eventually depend on all of the variables that have any influence whatsoever, direct or indirect, on a reward. In practice, this set will typically be all of the state variables, as it is somewhat superfluous to introduce into the process description variables whose value never matters to the agent's utility.

3 Constrained value determination

As we described in the introduction, the key idea behind our approach is the restriction of our algorithms to the use of value functions in a limited class. This idea is best known under the name *value function approximation*, which is used frequently in the context or reinforcement learning [Tadepalli and Ok, 1996; Van Roy, 1998]. We use this idea in the context of maintaining full value functions and propagating them through the DP equation (1) [Gordon, 1995; Tsitsiklis and Van Roy, 1996]. However, unlike other methods, which deal with large state spaces by considering only a restricted set of "representative" states, our method efficiently finds a least squares approximation for the *entire* state space.

More precisely, let $\mathcal{V} \subseteq I\!R^S$ be a restricted set of value functions. We will define $\mathcal{V}$ via a set of *basis* functions $H = \{h_1, \ldots, h_m\}$. That is, we have that H is a basis for $\mathcal{V}[H] \subseteq I\!R^S$ if every function $V \in \mathcal{V}[H]$ can be written as $V = \sum_{j=1}^{m} w_j h_j$ for some weights $w_1, \ldots, w_j$.

Our algorithm repeatedly executes the following steps. It begins with a value function $\hat{V}^{(t)} \in \mathcal{V}$. It then backprojects it via (1), resulting in a value function $\tilde{V}^{(t+1)}$. In general, the DP step does not maintain the property of being in a restricted class $\mathcal{V}$. Therefore, we typically have that $\tilde{V}^{(t+1)} \notin \mathcal{V}$. We then project $\tilde{V}^{(t+1)}$ into $\mathcal{V}$, i.e., find the "closest" value function in $\mathcal{V}$. The result is our new value function $\hat{V}^{(t+1)}$.

We must decide what it means for a value function $\hat{V} \in \mathcal{V}$ to be "close" to some other value function V by choosing a suitable distance measure. Several constraints combine to make this choice difficult. On the one hand, we need a *contraction* property: exact value determination converges because (1) is a contraction — each iteration decreases the error of $V^{(t)}$ by a constant factor. In order to get convergence for approximate value determination, we need a similar contraction property. We also need an effective algorithm for projecting into $\mathcal{V}$, relative to this distance. Finally, we need this projection operation to be a *non-expansion* under the same distance, otherwise we are not guaranteed the desired convergence property. Tsitsiklis and Van Roy [1996] underscore the importance of this point by demonstrating a two-state MDP for which this type of approximate value determination diverges if we use standard least-squares approximation. Unfortunately, it turns out to be nontrivial to find a distance metric that satisfies all criteria. For example, (1) is not a contraction in Euclidean distance. It is a contraction in L_∞ and L_1 norms[2], but we have no efficient projection algorithm for these distances.

One choice that turns out to satisfy both desiderata is the *weighted Euclidean distance*. Let $\rho(s)$ be the *occupancy probability* of the state s in the *stationary (steady-state) distribution* ρ.[3] We define the *weighted Euclidean distance* $d_\rho(V, \hat{V})$ as $\sum_s \rho(s)(V(s) - \hat{V}(s))^2$. We say that $\hat{V}$ is the *least d_ρ projection* of V into $\mathcal{V}$ if $\hat{V} \in \mathcal{V}$ is the function in $\mathcal{V}$ that is closest to V in terms of d_ρ. Intuitively, unweighted Euclidean distance places equal emphasis on getting each $\hat{V}(s)$ as close as possible to $V(s)$, whereas the weighted distance places more emphasis on getting correct values for states that are visited more often.

Fortunately, there exists a fairly straightforward algorithm for doing projection relative to weighted Euclidean distance. The key operation is the *weighted dot product*. For two functions $f, g \in I\!R^S$, we define $(f \bullet g)_\rho = \sum_s \rho(s) f(s) g(s)$. We define the *length* $\ell_\rho(h_j) = (h_j \bullet h_j)_\rho$. To project a function V into $\mathcal{V}[H]$, we first compute each coefficient $w_j' = (h_j \bullet V)_\rho / \ell_\rho(h_j)$. This intermediate weight vector $\mathbf{w}' = (w_1', \ldots, w_m')^t$ has to be transformed, in order to accommodate for any (linear) dependencies between our basis vectors. We define $a_{ij} = (h_i \bullet h_j)_\rho$, and define A to be the matrix (a_{ij}). If we now define a weight vector $\mathbf{w} = A^{-1}\mathbf{w}'$, we know that $\sum_{j=1}^{m} w_j h_j$ is the least d_ρ projection of V into $\mathcal{V}$ [Strang, 1980]. We define $\mathcal{P}_\rho$ to be the operator that projects into $\mathcal{V}$.

The contraction of the $\mathcal{T}$ operator in d_ρ distance is established in Nelson [1958] and is combined with projection in Van Roy [1998], yielding:

Theorem 3.1: *(a) The operator $\mathcal{T}$ is a contraction in d_ρ distance with rate $\gamma < 1$; hence, $\mathcal{T}$ has a unique fixed point V^*. (b) Let $\hat{\mathcal{T}} = \mathcal{P}_\rho \mathcal{T}$. Then $\hat{\mathcal{T}}$ is a contraction in d_ρ distance with rate $\kappa \leq \gamma$; hence, $\hat{\mathcal{T}}$ has a unique fixed point $\hat{V}^*$. Furthermore, $\hat{V}^*$ satisfies:*

$$d_\rho(V^*, \hat{V}^*) \leq \frac{1}{\sqrt{1-\kappa^2}} d_\rho(V^*, \mathcal{P}_\rho V^*).$$

In other words, the alternating DP and projection algorithm converges to a fixed point; the value function at that point is

[2] The L^1 norm contraction requires stronger assumptions than we have made here. For example, positive rewards and a pessimistic V^0 will suffice.

[3] We assume, for simplicity, that the process is mixing.

at most a constant factor worse than the *optimal* value function within $\mathcal{V}$, i.e., the one closest to the true optimal value function, V^*. Thus, if we assume that our true value function is well-approximated by some function within $\mathcal{V}$, we have strong bounds on the performance of the algorithm.

4 Factored value functions

The ideas described in the previous section are well-known. They allow a compact representation of very complex value functions: we only need to maintain the coefficients (the w_i's) for the limited number of basis functions that we choose to use. Why have these methods not already been used to address the problem of value determination in factored Markov processes? Unfortunately, a compact representation of the value function is not enough. The representation has to support efficient execution of our two main computational steps: the DP step, $\mathcal{T}$, and the dot product step required for $\mathcal{P}_\rho$.

We thus turn our attention to the specific properties of factored processes. As we discussed in the introduction, we propose restricting our algorithm to *factored* value functions, ones that are linear in functions over small clusters of variables. More precisely, we define a *cluster* $\mathbf{C}$ of variables to be a subset of $\mathbf{X}$. The key idea behind our result is that we can efficiently perform approximate value determination with any basis whose functions are restricted to small clusters. In decision-analytic terminology, we say that a function is *generalized additive* [Bacchus and Grove, 1995] over a set of clusters $\mathcal{C}$ if it can be written as a $\sum_{i=1}^m U_i$, where each U_i is restricted to some $\mathbf{C} \in \mathcal{C}$. The function is said to be *additive* if the clusters in $\mathcal{C}$ are disjoint. Now, let $H = \{h_1, \ldots, h_m\}$, and assume that each h_j is restricted to some small cluster $\mathbf{C}_j$. Clearly, the functions in $\mathcal{V}[H]$ are generalized additive in $\{\mathbf{C}_1, \ldots, \mathbf{C}_m\}$. In fact, it is easy to construct a basis that allows us to have $\mathcal{V}[H]$ represent *exactly* the set of all generalized additive functions over a particular set of clusters. In the remainder of the analysis, we assume that we have restricted attention to a specific basis H, and use $\mathcal{V}$ to denote $\mathcal{V}[H]$.

As we now show, restricted basis functions allow an efficient implementation of the DP and projection steps. Assume that we are restricting to value functions in $\mathcal{V}$, so that (as a result of the algorithm so far) $\hat{V}^{(t)}$ is in this space. Thus, as we argued, $\hat{V}^{(t)}$ can be represented as $\sum_{i=1}^m U_i$, for some set of functions U_i, each restricted to some small cluster $\mathbf{C}_i$. (Specifically, we have $U_i = w_i h_i$.) Let us examine the result of the DP step:

$$\begin{aligned}\tilde{V}^{(t+1)}(\mathbf{x}) &= R(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}) \hat{V}^{(t)}(\mathbf{x}') \\ &= \sum_{i=1}^r R_i(\mathbf{x}) + \gamma \sum_{i=1}^m \sum_{\mathbf{x}'} U_i(\mathbf{x}') P(\mathbf{x}' \mid \mathbf{x}) \qquad (2)\end{aligned}$$

Consider one of the terms in (2) that corresponds to some function U_i. As we saw in Example 2.1, if the domain of a value function is restricted to a single variable X_1, its back-projection through (1) depends on all of X_1's parents. The same principle holds here. Thus, let $\mathbf{C}_i^-$ be the variables in $\cup_{X_k \in \mathbf{C}_i} \mathrm{Pa}(X_k)$; let $\mathbf{c}_i^-$ be the instantiation of these variables in the state $\mathbf{x}$. It is easy to show that

$$\sum_{\mathbf{x}'} U_i(\mathbf{x}') P(\mathbf{x}' \mid \mathbf{x}) = \sum_{\mathbf{c}' \in \mathrm{Val}(\mathbf{C}_i)} U_i(\mathbf{c}') P(\mathbf{c}' \mid \mathbf{c}_i^-) \qquad (3)$$

Thus, the term for U_i in (2) simplifies to an expression in (3) that depends only on the value of the variables in $\mathbf{C}_i^-$ in the preceding state $\mathbf{x}$. We denote the function in (3) by U_i^-. To compute it, we must (at worst) generate the entire conditional probability $P(\mathbf{c}' \mid \mathbf{c}_i^-)$; this can take $O(|\mathbf{C}_i| \cdot |\mathrm{Val}(\mathbf{C}_i)| \cdot |\mathrm{Val}(\mathbf{C}_i^-)|)$ operations. The locality of influence that is characteristic of DBNs typically implies that $\mathbf{C}_i^-$ is small, allowing this computation to be done efficiently.

To compute the unweighted projection of $\tilde{V}^{(t+1)}(\mathbf{x})$, we simply need to compute its dot product with each basis function h_j. As the dot product is a linear operation, it can be decomposed into a separate dot product of h_j with each of the terms in (2): R_i for $i = 1 \ldots r$ and U_i^- for $i = 1 \ldots m$. It is straightforward to verify that computing the dot product of h_i with a function restricted to a set $\mathbf{Y}$ requires time which is linear in $|\mathrm{Val}(\mathbf{C}_i \cup \mathbf{Y})|$.

Unfortunately, the unweighted projection is not the operation we need; rather, we need to compute the weighted projection in order to get a least d_ρ projection. There are two major obstacles preventing us from using a weighted dot product in our context. First, the stationary distribution of our current policy is not known. More importantly, the stationary distribution has no structure that can be exploited to allow a compact representation [Boyen and Koller, 1998]. In other words, we can represent this distribution only explicitly, as an exponentially-sized probability function. This problem makes it impractical to compute the stationary distribution; it also prevents us from doing the projection step efficiently, as discussed above.

We solve both problems at once by building on the work of Boyen and Koller [1998] (BK from here on). They show how a process very similar to our iterated DP & project algorithm can be used to maintain and propagate a compactly represented approximate *belief state* — a distribution over the states at a given point in time. Most simply, they partition $\mathbf{X}$ into a set of disjoint clusters $\mathbf{Q}_1, \ldots, \mathbf{Q}_q$. The distribution over the states $\hat{\mu}^{(t)}$ is represented implicitly as a product of distributions $\rho_1, \ldots, \rho_k$ over these clusters. The step of propagating $\hat{\mu}^{(t)}$ to the next time step results in a distribution $\tilde{\mu}^{(t+1)}$ that is not of the right form; the algorithm generates $\hat{\mu}^{(t+1)}$ by computing the marginal distributions over each $\mathbf{Q}_k$. BK show that, if the marginalization introduces an error of $\leq \epsilon$, then, for all t, $\hat{\mu}^{(t)}$ is $\leq \epsilon/\delta$ away (in relative entropy) from the distribution that would have been maintained by a similar process without the approximation. Here, δ is the *mixing rate* of the process, which can be shown to depend on the sizes of the CPTs in the DBN, and not on the size of the process. Thus, we have that $d_{KL}(\rho \| \hat{\mu}^{(t)}) = \epsilon/\delta + O((1-\delta)^t)$. Hence, the number of iterations required to get close to the stationary distribution is very small — logarithmic in $1 - \delta$. The benefits of this approximate computation are twofold: (1) The complexity of the algorithm is not prohibitive; its complexity depends on the sizes of the clusters $\mathbf{Q}_1, \ldots, \mathbf{Q}_q$ and on the interconnectivity of the DBN. (2) The result is a factored distribution, which can be used in an efficient weighted projection algorithm.

Let us see how a factored distribution can be used in the context of a weighted projection. The key procedure used in the algorithm is that of computing $(g \bullet \rho) = \sum_{\mathbf{x}} \rho(\mathbf{x})g(\mathbf{x})$ for some function g restricted to a set $\mathbf{Y}$. It is easily verified that $(g \bullet \rho) = \sum_{\mathbf{y} \in \mathrm{Val}(\mathbf{Y})} g(\mathbf{y})\rho(\mathbf{y})$. We note that we can easily compute $\rho(\mathbf{y})$. Let $\mathbf{y}_k$ denote the part of $\mathbf{y}$ that overlaps with variables in the cluster $\mathbf{Q}_k$. We compute $\rho(\mathbf{y}_k)$ by a simple marginalization process over the ρ_k component of ρ, and then multiply the results for all $k = 1, \ldots, q$. Note that this operation requires time which is linear in $|\mathrm{Val}(\mathbf{Y})|$ and in $|\mathrm{Val}(\mathbf{Q}_k)|$ for the relevant clusters $\mathbf{Q}_k$. We can use this subroutine for doing the weighted projection, simply by noting that $(f \bullet g)_\rho = ((f \cdot g) \bullet \rho)$, and that if f is restricted to $\mathbf{Y}$ and g to $\mathbf{Z}$, then $f \cdot g$ is restricted to $\mathbf{Y} \cup \mathbf{Z}$.

The full algorithm is as follows:

1. Compute a factored approximation ρ to the stationary distribution.
2. Precompute for each $i, j = 1, \ldots, m$, $a_{ij} = (h_i \bullet h_j)_\rho$ and set $\ell_\rho(h_j) = a_{jj}$. Invert the matrix A.
3. Precompute for each $i = 1, \ldots, m$ and $j = 1, \ldots, r$ $r_{ij} = (R_j \bullet h_i)_\rho$.
4. Set all $w_i^{(0)} = 0$, which defines $\hat{V}^{(0)}$. Iterate the following computation over t:
   (a) For each component U_j of $\hat{V}^{(t)}$ compute U_j^-.
   (b) $u_{ij} = (U_j^- \bullet h_i)_\rho$;
   (c) $\hat{w}_i = \sum_j r_{ij} + \gamma \sum_j u_{ij}$;
   (d) $\mathbf{w}^{(t+1)} = A^{-1}\hat{\mathbf{w}}$, which in turn defines $\hat{V}^{(t+1)}$.

The number of operations required by an iteration of this algorithm grows linearly with the number of basis functions, and exponentially with the sizes of the clusters and their backprojections. It also depends (in step 4b) on the extent to which backprojected clusters $\mathbf{C}_i^-$ overlap with other clusters $\mathbf{C}_j$. Thus, the complexity is determined by the interconnectivity of the DBN, by the amount of interaction in our restricted set of value functions, and by the extent to which the structure of the value functions matches the structure of the process.

The main remaining question involves the performance of this algorithm. Ideally, we would like to state a theorem along the lines of Theorem 3.1. The problem is that our projection step no longer uses the exact stationary distribution as weights. However, one would hope that if the approximation is a good one (as suggested by the results of BK), the error introduced by using projection relative to the approximate weights would not prevent the process from converging. Indeed, we can show that such a result holds (albeit only for relative error approximation, a slightly stronger notion than the one guaranteed by BK):

Theorem 4.1: *Assume that ρ is an ϵ-factor relative approximation to the true stationary distribution ρ^*; i.e., for all $\mathbf{x}$, $(1-\epsilon)\rho(\mathbf{x}) \leq \rho^*(\mathbf{x}) \leq (1+\epsilon)\rho(\mathbf{x})$. Let κ be the contraction rate of $\mathcal{P}_{\rho^*}\mathcal{T}$. If $\hat{\kappa} = \kappa(1+\epsilon)/(1-\epsilon) < 1$, then our approximate value determination algorithm $\mathcal{P}_\rho\mathcal{T}$ is a contraction in d_ρ distance with rate $\hat{\kappa}$, and therefore has a unique fixed point $\hat{V}^*$. Furthermore, $\hat{V}^*$ satisfies:*

$$d_{\rho^*}(V^*, \hat{V}^*) \leq \frac{1}{\sqrt{1-\hat{\kappa}^2}} d_{\rho^*}(V^*, \mathcal{P}_{\rho^*}V^*).$$

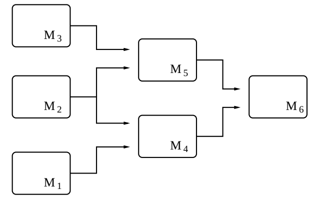

Figure 1: A factory with six machines. A reward of 1.0 is given for each time step in which machine M_6 is functioning.

Added basis functions	Unweighted Projection SSE_μ	Unweighted Projection SSE_{ρ^*}	Weighted Projection SSE_μ	Weighted Projection SSE_{ρ^*}
$1, h(M_6)$	0.368	0.326	0.169	0.160
$h(M_5), h(M_4)$	2.283	2.757	0.098	0.124
$h(M_3), h(M_2), h(M_1)$	0.632	0.683	0.130	0.111
$h(M_5, M_4)$	0.104	0.132	0.026	0.020
$h(M_3, M_2)$	0.075	0.092	0.023	0.019
$h(M_2, M_1)$	0.051	0.060	0.020	0.018

Table 1: The change in the sum of squared errors (SSE) weighted by the uniform distribution, μ, and by the true stationary distribution, ρ^*.

5 Experimental Results

We performed some simple experiments to demonstrate the convergent behavior of our algorithm and to verify our intuitions about the additive structure of MDP value functions. Figure 1 shows an abstracted factory with six machines, $M_1 \ldots M_6$. In this problem, a machine can be in two states, *working* and *not-working*. If a machine and all of its predecessors were working in the previous time step, then the machine will work in the current time step with probability 0.8. There is a stochastic startup lag, which reduces this probability to 0.5 if the machine was not working in the previous time step. No machine can operate if its predecessors were not both working in the previous time step. Note that a failure in any of $M_1 \ldots M_3$ will have a cascade effect, ultimately causing M_6 to fail. It will take several time steps for the system to recover. A reward of 1.0 is received whenever machine M_6 is working.

We evaluated this 64-state system using our constrained value determination algorithm with a discount factor of $\gamma = 0.90$ and a series of six different bases. The basis functions are binary so that, for example, $h(M_2, M_1)$ has value 1 when both M_2 and M_1 are working, and value 0 otherwise. We performed both unweighted projection, which is not guaranteed to converge, and our weighted projection algorithm. For this problem, we achieved convergence in all cases. The second and third columns of Table 1 show the reduction in error as new basis functions are added. The first three additions allow the value function to depend on the status of more individual machines. The last three allow it to depend in a correlated way on the status of pairs of machines. In each case, we ap-

Added basis functions	Unweighted Projection SSE_μ	Unweighted Projection SSE_{ρ^*}	Weighted Projection SSE_μ	Weighted Projection SSE_{ρ^*}
$1, h(M_6)$	1.449	1.663	0.260	0.303
$h(M_5), h(M_4)$	0.319	0.385	0.207	0.174
$h(M_3), h(M_2), h(M_1)$	0.175	0.225	0.132	0.121
$h(M_5, M_4)$	0.175	0.225	0.132	0.121
$h(M_3, M_2)$	0.210	0.277	0.120	0.115
$h(M_2, M_1)$	0.210	0.277	0.120	0.115

Table 2: Results with M_6 modified to work if *either* of M_4 or M_5 are working.

proximated the stationary distribution using a product of distributions over the individual machine states. Notice that the weighted projection outperforms unweighted projection in all cases, even in the unweighted norm. Also, the ρ^* weighted error for the weighted projection is monotonically decreasing, while the other errors fluctuate. The final value function for the weighted projection with all ten basis functions corresponds to an unweighted RMS error of less than 0.141. The true value function for this problem assigns values between 0.48 and 2.78, so that 0.141 is a very reasonable error bound. Hence, our final value function is a good 10 parameter approximation to the true, 64 parameter value function.

To further verify our hypothesis about additive structure of value functions, we changed machine M_6 so that it would work if *either* of M_4 or M_5 were functioning. In this case, we would expect the value function to be less sensitive to correlations between the status of M_4 and M_5. The results, shown in the Table 2, support this hypothesis. We see that introducing the basis function $h(M_5, M_4)$ provides no significant reduction in the error. (The overall increase in error compared to the original problem is probably due to higher overall values in this model — between 2.30 and 5.74.) We observe a similar pattern in the comparison of weighted and unweighted projection: weighted projection provides more reliable approximations.

Finally, we demonstrate how the value functions generated by our algorithm can be useful in decision making. Suppose that there are two types of workers, *reliable* and *resilient*. When a reliable worker is operating a working machine, it will continue to work properly with probability 0.9, but if the machine fails, the worker will restart the machine successfully with probability 0.4. A resilient worker will keep a working machine functioning with a probability of only 0.8, but recovers from failures with probability 0.5. If the manager of our 6-machine factory has three of each type of worker, he may wish to compare different strategies for allocating workers to machines. One possibility would be a *reliable-first* strategy that places the reliable workers on machines $M_1 \ldots M_3$ to avoid cascade failures. A *resilient-first* strategy would place the resilient workers on machines $M_1 \ldots M_3$ to recover quickly when failures occur. We consider the original machine configuration, which requires both M_4 and M_5 to be working for machine M_6 to work properly. For this problem the reliable-first strategy completely dominates the the resiliant-first strategy.

Figure 2 shows a graph of the true value function for the resilient-first strategy versus the approximation with all 10 basis functions. The states are numbered $0 \ldots 63$. The status

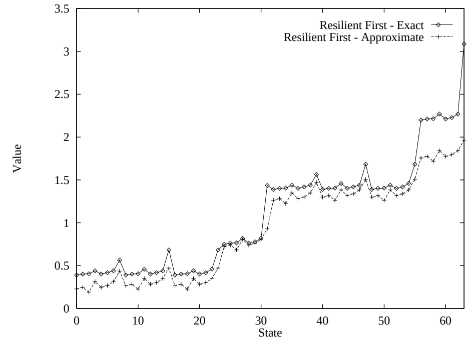

Figure 2: The resilient-first strategy.

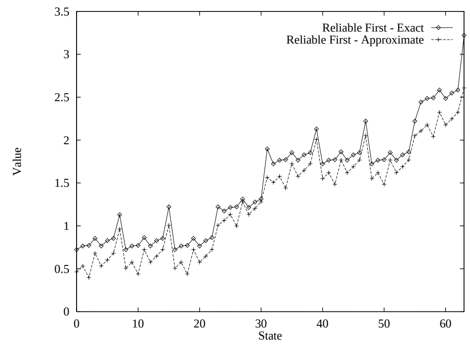

Figure 3: The reliable-first strategy.

of the machines is used to identify the states, e.g., when machine M_6 is working, the highest order bit of the state number has value 1; when machine M_1 is working the lowest order bit of the state number has value 1. Figure 3 shows both value functions for the reliable-first strategy. The qualitative closeness of the these approximations is apparent from inspection — the shape of the approximations are quite close to the shape of the exact value functions. Moreover the approximate value functions maintain the dominance of the reliable-first strategy over the resiliant-first strategy. Thus, we see that our approximation closely matches the qualitative structure of the exact value functions for these domains.

6 Policy evaluation

We now address the issue of using our value determination algorithm to evaluate a policy in an MDP, where we extend the definition of our process to incorporate actions. Most importantly, the transition probability P and the reward R now also depend on a. In the context of factored MDPs, this extension is usually done by defining a separate transition graph and associated set of CPTs for each action a; the resulting transition probability $P_a(\mathbf{X}' \mid \mathbf{X})$ is thereby factored (perhaps in a different way) for every action a. The reward function can also vary over actions in a similar way.

Our goal now is to compute an approximate value function for a particular policy π. Intuitively, there seems to be no problem. A policy π in an MDP specifies a transition model $P_\pi(\mathbf{x}' \mid \mathbf{x})$ and a reward function $R_\pi(\mathbf{x})$. We might simply apply our algorithm to these. Unfortunately, the problem is somewhat more subtle. The most obvious difficulty is that the naive representation of a policy in an MDP with exponentially many states is also exponentially large. Therefore, we might not even be able to construct a compact transition model and reward function for π. Clearly, we must restrict the set of policies that we consider to ones that can be efficiently manipulated by our algorithm.

To understand this issue, assume that we have constructed a value function $\hat{V}_\pi^{(t)}$, and are now trying to perform our DP and projection steps. We do not have a single transition model to use for the next step of backprojection, because the action depends on the state. However, we can use our algorithm to compute a factored approximation $\hat{Q}_a^{(t+1)}$ to the Q function $Q_a(\mathbf{x}) = R_a(\mathbf{x}) + \gamma \sum_{\mathbf{x}'} P_a(\mathbf{x}' \mid \mathbf{x}) \hat{V}_\pi^{(t)}$: we simply backproject $\hat{V}_\pi^{(t)}$ through the transition model P_a, and then project into $\mathcal{V}$. We now have to combine the different $\hat{Q}$ functions to get a complete value function. For each action a, let S_a be the subset of states in which a is taken, and let I_{S_a} be the indicator function which takes value 1 at $\mathbf{x}$ when $\mathbf{x} \in S_a$ and value 0 otherwise. Then,

$$\tilde{V}^{(t+1)}(\mathbf{x}) = \sum_a I_{S_a}(\mathbf{x}) \hat{Q}^{(t+1)}(\mathbf{x}).$$

We see that, depending upon the policy, $\tilde{V}^{(t+1)}$ can be composed of an arbitrary combination of pieces of the $\hat{Q}^{(t+1)}$ functions. In particular, if the indicator functions I_{S_a} depend on variables in many different clusters, the combination will no longer be in $\mathcal{V}$, even if each of the $\hat{Q}^{(t+1)}$ is in $\mathcal{V}$. Thus, we need to project $\tilde{V}^{(t+1)}$ back into $\mathcal{V}$.

As before, we need to compute $((I_{S_a} \cdot \tilde{V}^{(t+1)}) \bullet h_j)_\rho$ to perform the projection. This reduces to the problem of computing $((I_{S_a} \cdot \hat{Q}^{(t+1)}) \bullet h_j)_\rho$ for each action and summing the results. Since $\hat{Q}^{(t+1)}$ is in $\mathcal{V}$, it can be represented as $\sum_i U_i$, where each U_i is restricted to some small cluster $\mathbf{C}_i$. Again, we need only compute $((I_{S_a} \cdot U_i) \bullet h_j)_\rho$. Let $\mathbf{Y}$ be the set of variables appearing in $\mathbf{C}_j \cup \mathbf{C}_i$. Then:

$$((I_{S_a} \cdot U_i) \bullet h_j)_\rho = \sum_{\mathbf{y}} U_i(\mathbf{y}) h_j(\mathbf{y}) \sum_{\mathbf{x} \in S_a[\mathbf{y}]} \rho(\mathbf{x}) \quad (4)$$

where $S_a[\mathbf{y}] = \{\mathbf{x} \in S_a \ : \ \mathbf{x} \text{ is consistent with } \mathbf{y}\}$. Thus, in order to perform this weighted dot product operation, we must be able to compute the total probability, according to the approximate stationary distribution, of $S_a[\mathbf{y}]$. In many cases, this computation is far from trivial, even if the set S_a has a simple and compact description (e.g., as a linear separator defined over the variables $X_1, \ldots, X_n$).

However, for what is arguably the most often-used class of compactly represented policies, this computation can be done easily.[4] A *region-based policy* is one represented as a set $\{(\mathbf{z}_k, a_k)\}_{\ell=1}^p$, where each $\mathbf{z}_\ell$ is an assignment of values to some subset of the variables in $\mathbf{X}$. (Different $\mathbf{z}_\ell$'s can refer to different sets of variables.) Each $\mathbf{z}_\ell$ thereby specifies a region in the state space, such that action a_ℓ is taken in this region. The assumption is that the regions corresponding to the different $\mathbf{z}_\ell$ are mutually exclusive and exhaustive. Note that a decision tree representation of the policy [Boutilier and Dearden, 1996], falls into this category, as do the policies for the aggregated state spaces [Dean *et al.*, 1997]. Given the factored representation of ρ, $\rho(\mathbf{z}_\ell, \mathbf{y})$ can be computed simply and efficiently, in the obvious way.

It remains only to extend the algorithm of BK to the task of computing the approximate stationary distribution, ρ, for a region-based policy, π. Recall that their algorithm was based on repeatedly computing a factored belief state $\hat{\mu}^{(t+1)}$ from a factored belief state $\hat{\mu}^{(t)}$, where

$$\hat{\mu}^{(t+1)}(\mathbf{Q}_k) = \sum_{\mathbf{x}} \hat{\mu}^{(t)}(\mathbf{x}) P(\mathbf{Q}_k \mid \mathbf{x}).$$

Here, we have a potentially different transition model P_{a_ℓ} for each a_ℓ. Therefore, we must do a separate computation for each of them:

$$\begin{aligned} \hat{\mu}^{(t+1)}(\mathbf{Q}_k) &= \sum_\ell \sum_{\mathbf{x} \in S_{\mathbf{z}_\ell}} \hat{\mu}^{(t)}(\mathbf{x}) P_{a_\ell}(\mathbf{Q}_k \mid \mathbf{x}) \\ &= \sum_\ell \hat{\mu}^{(t)}(\mathbf{z}_\ell) \sum_{\mathbf{x}} \hat{\mu}^{(t)}(\mathbf{x} \mid \mathbf{z}_\ell) P_{a_\ell}(\mathbf{Q}_k \mid \mathbf{x}). \end{aligned}$$

Fortunately, the algorithm of BK can easily be adapted to this task. We simply instantiate $\mathbf{z}_\ell$ as evidence into the Bayesian network that they use for doing the one step propagation, and then compute the internal summation. The probabilities $\hat{\mu}^{(t)}(\mathbf{z}_\ell)$ can be computed easily, as described above. This algorithm provides a factored representation of the stationary distribution, which can be used in Equation 4. Thus, we can efficiently compute an approximate value function for any region-based policy.

7 Discussion and conclusions

We have shown that we can efficiently compute factored approximations to value functions. We have also demonstrated, both theoretically and empirically, that the result of our approximate value determination algorithm can often be quite close to the true value function. Finally, we showed how to use our algorithm for evaluating policies in a restricted but interesting class.

It is useful to compare our algorithm to other algorithms for factored MDPs [Boutilier *et al.*, 1999]. Some algorithms for factored MDPs try to construct compact value functions by exploiting structured, usually tree-based, CPTs. When the value function representation becomes too large, the standard approach is to perform a type of state aggregation [Boutilier and Dearden, 1996; Dean *et al.*, 1997]. Thus, their approximate value function is one that is piecewise constant over the state space. In other words, there are substantial blocks of states all of which take the same value. A main advantage of our approach is that it uses a richer class of value function approximations. It seems clear that, at least in some domains, a linear approximation is much more accurate. While the algorithm presented in this paper does not exploit structured CPTs, this feature can be added easily to make both the dynamic programming steps and projection steps more efficient.

[4] For other classes of policies, Monte Carlo sampling may succeed in providing good estimates for $\rho(S_a[\mathbf{y}])$. We defer discussion to a longer paper.

Other approaches [Meuleau *et al.*, 1998; Boutilier *et al.*, 1998; Singh and Cohn, 1998] decompose the process into subprocesses, and compute a separate value function for each one. They then attempt to combine the different value functions into a global one using some heuristic approximation. Generally speaking, the decoupling of the *process* is a much harsher approximation than the factorization of the value function. The latter still allows influence to flow between the different subprocesses, and thereby is likely to get more accurate results.

Of course, it is important to remember that these other approaches solve the planning problem whereas, at least for now, our algorithm is restricted to the value determination problem. The obvious question is how our value determination algorithm can be used for planning. The most obvious approach is to use our algorithm as a subroutine in an algorithm such as *policy iteration*, where we compute the value function for some policy, and use it as guidance for locally improving the policy. Unfortunately, there are barriers to this application of our algorithm. The difficulty lies in the nature of our approximation. We measure our distance from the true value function in a norm that is weighted by the probability that the different states will be visited in the stationary distribution. Hence, states that are visited infrequently can have very poor value estimates. Changes to the policy based upon these estimates will not be guaranteed to improve the overall policy as in exact policy iteration, and could actually make the policy much worse. The issue of using our value functions as guidance in policy search is an important direction for future work.

There are, however, several other important uses for a value determination algorithm. Plan evaluation is an important problem in and of itself; our algorithm allows us to take policies generated by a user or some other tool, and produce a fairly good value function for them. Furthermore, it is known that simple policies can be improved significantly by the addition of a small lookahead search that uses the value function of the policy as a heuristic value at the leaves of the search. Indeed, the recent work of Sutton [1999] shows that even very simple policies can be quite powerful if a meta-level reasoner is allowed to choose which one to apply in different contexts. Our algorithm can readily be used to compute the value functions necessary for making such a choice.

Factored MDPs provide us with a compact and natural representation for the type of complex problems that arise in real-world settings. Unfortunately, the structure is not reflected in the value function for these processes. However, we believe that the value functions that arise are often approximately factored, i.e., approximated well by the generalized additive functions that we have discussed. The fact that such functions are often used by human decision makers supports this intuition. We therefore believe that algorithms that exploit this structure can be very successful. This paper takes a first step towards this goal.

Acknowledgement

We are grateful to Craig Boutilier, Andrew Ng, Satinder Singh, and Benjamin Van Roy for helpful discussions. This work supported by ARO grant DAAH04-96-1-0341 under the MURI program "Integrated Approach to Intelligent Systems", by ONR contract N66001-97-C-8554 under DARPA's HPKB program, and by the generosity of the the Powell Foundation and the Sloan Foundation.

References

[Bacchus and Grove, 1995] F. Bacchus and A. Grove. Graphical models for preference and utility. In *Proc. UAI*, 1995.

[Boutilier and Dearden, 1996] C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. In *Proc. ICML*, pages 54–62, 1996.

[Boutilier *et al.*, 1998] C. Boutilier, R.I. Brafman, and C. Geib. Prioritized goal decomposition of Markov decision processes: Towards a synthesis of classical and decision theoretic planning. In *Proc. UAI*, pages 24–32, 1998.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999.

[Boyen and Koller, 1998] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, 1998.

[Dean *et al.*, 1997] T. Dean, R. Givan, and S. Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proc. UAI*, 1997.

[Gordon, 1995] G.J. Gordon. Stable function approximation in dynamic programming. In *Proc. ICML*, pages 261–268, 1995.

[Howard and Matheson, 1984] R.A. Howard and J.E. Matheson. Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, 1984.

[Keeney and Raiffa, 1976] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, 1976.

[Meuleau *et al.*, 1998] N. Meuleau, M. Hauskrecht, K-E. Kim, L. Peshkin, L.P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled Markov decision processes. In *Proc. AAAI*, pages 165–172, 1998.

[Nelson, 1958] E. Nelson. The adjoint Markoff process. *Duke Mathematical Journal*, 25, 1958.

[Singh and Cohn, 1998] S.P. Singh and D. Cohn. How to dynamically merge Markov decision processes. In *NIPS 10*, pages 1057–1063, 1998.

[Strang, 1980] G. Strang. *Linear Algebra and Its Applications*. Academic Press, 1980.

[Sutton *et al.*, 1999] R.S. Sutton, S. Singh, D. Precup, and B. Ravindran. Improved switching among temporally abstract actions. In *NIPS 12*, 1999. To appear.

[Tadepalli and Ok, 1996] P. Tadepalli and D. Ok. Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proc. ICML*, 1996.

[Tsitsiklis and Van Roy, 1996] J. D. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, January 1996.

[Van Roy, 1998] B. Van Roy. *Learning and Value Function Approximation in Complex Decision Problems*. PhD thesis, Massachusetts Institute of Technology, 1998.