IJCAI-01 workshop on Planning under Uncertainty and Incomplete Information (PRO-2), pp. 67 - 75, Seattle, Washington, August 2001.

Solving Factored POMDPs with Linear Value Functions

Carlos Guestrin
Computer Science Dept.
Stanford University
guestrin@cs.stanford.edu

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

Ronald Parr
Computer Science Dept.
Duke University
parr@cs.duke.edu

Abstract

Partially Observable Markov Decision Processes (POMDPs) provide a coherent mathematical framework for planning under uncertainty when the state of the system cannot be fully observed. However, the problem of finding an exact POMDP solution is intractable. Computing such solution requires the manipulation of a piecewise linear convex *value function*, which specifies a value for each possible belief state. This value function can be represented by a set of vectors, each one with dimension equal to the size of the state space. In nontrivial problems, however, these vectors are too large for such a representation to be feasible, preventing the use of exact POMDP algorithms. We propose an approximation scheme where each vector is represented as a linear combination of basis functions to provide a compact approximation to the value function. We also show that this representation can be exploited to allow for efficient computations in approximate value and policy iteration algorithms in the context of *factored POMDPs*, where the transition model is specified using a dynamic Bayesian network.

1 Introduction

Over the last few years, *Partially Observable Markov Decision Processes (POMDPs)* have been used as the basic semantics for optimal planning for decision theoretic agents in stochastic environments where the state of the system cannot be fully observed. In the POMDP framework, the system is modeled via a set of states which evolve stochastically. The key problem with this 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 compactly if the structure is exploited in the representation.

Factored POMDPs [Boutilier and Poole, 1996] are one approach to representing large structured POMDPs compactly. In this framework, a state is implicitly described by an assignment to some set of *state variables*. A *dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989] can then allow a compact representation of the transition model, by exploiting the fact that the transition of a variable often depends only on a small number of other variables. Furthermore, the momentary rewards can often also be decomposed as a sum of rewards related to individual variables or small clusters of variables. Finally, the observations can be decomposed into observation variables, each one giving evidence about a small subset of the variables.

Even when a large POMDP can be represented compactly using a factored model, finding an optimal policy is still intractable: Exact POMDP solutions are *EXP-hard* [Littman, 1996] and in many cases *undecidable* [Madani *et al.*, 1999]. Algorithms require the manipulation of a piecewise linear value function, where each piece has a representation which is linear in the number of states, thus, exponential in the number of state variables. One approach is to approximate the solution using an approximate value function with a compact representation. In this paper, we represent each piece of the value function as a linear combination of basis functions, where each basis function has a restricted *domain*, i.e., depends on only a small number of the state variables. This allows us to address the cost of representing the value function. Furthermore, we present an algorithm that exploits the structure in the factored POMDP and in the value function to perform value and policy iteration efficiently with this compact representation.

2 Partially observable Markov decision processes

In this section, we briefly present the traditional approach for solving POMDPs; more details can be found in [Littman, 1996]. A *Partially Observable Markov Decision Process (POMDP)* is defined as a tuple $(S, \mathcal{O}, A, R, \mathcal{P})$ where: S is a finite set of $|S|$ states; $\mathcal{O}$ is a finite set of $|\mathcal{O}|$ possible observations; A is a set of $|A|$ actions; R is a *reward function* $R : S \times A \mapsto \mathbb{R}$, such that $R(s, a)$ represents the reward obtained by the agent in state s after taking action a; and $\mathcal{P}$ is a set of *Markovian transition models*, one for each action a, such that $P_a(s' \mid s)$ represents the probability of going from state s to state s' with action a and a corresponding set of *observation models* $P_a(o \mid s')$ which gives the probability of making observation o after taking action a and transitioning to state s'. We will be assuming that the POMDP has an infinite horizon and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$.

Although the agent cannot directly observe the state of the system, it is possible to maintain a probability distribution over the states. We denote the *belief state* b as an $|S| \times 1$

vector, where $b(s)$ is the probability that the system is at state s. Once the agent takes action a and makes observation o, it is possible to update this belief state by a simple application of Bayes' rule:

$$b_o^a(s') = \frac{\sum_{s \in S} P(s', o \mid s, a) b(s)}{P(o \mid b, a)};$$

where, $P(o \mid b, a) = \sum_{s, s' \in S} P(s', o \mid s, a) b(s)$.

2.1 Value Iteration and Incremental Pruning

The belief state summarizes all the information present in the previous observations, i.e., it is a sufficient statistic. Thus, it is possible to recast the POMDP as a fully observable continuous MDP, where the state has $|S|$ dimensions and represents all possible belief states.

This continuous-state MDP can be solved by a *Value Iteration* algorithm, which relies on successive applications of the dynamic programming (DP) update rule:

$$V^{(t+1)}(b) = \max_a \sum_{s \in S} b(s) R(s, a) + \gamma \sum_{o \in \mathcal{O}} P_a(o \mid b) V^{(t)}(b_o^a). \tag{1}$$

Smallwood and Sondik [1973] proved that the optimal value function with horizon t is piecewise linear and convex. More precisely, it can be represented as the maximum of several linear functions, each function corresponding to the value of some particular t-step policy. Thus, the value function can be represented by a finite set of $|S|$-dimensional vectors of real numbers: $\{\alpha_1, ..., \alpha_k\}$, such that the value of a belief state b is given by:

$$V(b) = \max_{i=1 \ldots k} \langle b \bullet \alpha_i \rangle, \tag{2}$$

where $\langle \cdot \bullet \cdot \rangle$ is the standard dot product: $\langle b \bullet \alpha \rangle = \sum_{s \in S} b(s) \cdot \alpha(s)$. The DP step preserves the piecewise linearity and convexity of the value function: Given some set of vectors that represent the t-step value function, we can generate a new set of vectors that represent the $t+1$-step value function.

As we discussed, the t-step value function is the maximum of a set of linear functions, one for each t-step policy. The number of such policies is enormous: We can view a policy as a branching tree, with a branch for each possible observation at step t, and a branch for each possible action that the agent might take in response to this action. Thus, the total number of possible strategies is $(|A|^{|\mathcal{O}|})^t$. Each of these induces a vector (or linear function) in the t-step value function.

Fortunately, many of these vectors are redundant, because the strategies they represent are suboptimal. In other words, we might have a α_j, such that there is no belief state b, for which this vector is larger than the others in the set:

$$\forall\, b\, \exists\, \alpha_i, i \neq j, \langle b \bullet \alpha_i \rangle \geq \langle b \bullet \alpha_j \rangle.$$

Such vectors, called *dominated* vectors, do not affect the value function, and can be pruned from the set of vectors representing the value function without affecting it. The *Incremental Pruning Algorithm* of Cassandra *et al.* [Cassandra *et al.*, 1997] is based on the key insight that the pruning operation can be performed incrementally, alleviating the need to generate this large set of vectors in many cases and reducing the size of the linear program that are generated. Incremental Pruning and its extensions have been shown empirically to be faster than alternative algorithms for value iteration in POMDPs.

We now provide a formal description of the incremental pruning algorithm, which we extend to the case of factored POMDPs in Section 3. The key step in the value iteration algorithm is the DP step, where we generate the $t+1$-step value function from the t-step value function. We perform this step by back-projecting the set of vectors in the step-t value function to obtain the set of vectors in the step-$t+1$ value function. To present this operation, it is convenient to divide the DP backup in Eq. (1) into three steps:

$$\begin{aligned} V^{(t+1)}(b) &= \max_a V_a^{(t+1)}(b); \\ V_a^{(t+1)}(b) &= \sum_{o \in \mathcal{O}} V_{a,o}^{(t+1)}(b); \\ V_{a,o}^{(t+1)}(b) &= \frac{\sum_{s \in S} b(s) R(s, a)}{|\mathcal{O}|} + \gamma P_a(o \mid b) V^{(t)}(b_o^a). \end{aligned}$$

Each of these value functions is piecewise linear and convex. The value function can therefore be represented by a unique minimal set of vectors. We will denote these sets: $\Upsilon^{(t+1)}$, $\Upsilon_a^{(t+1)}$ and $\Upsilon_{a,o}^{(t+1)}$.

Now, let $\Upsilon^{(t)}$ represent the set of vectors in the step-t value function. First, we define the backprojection $\Gamma_{a,o}(\alpha)$ of a vector α. Note that this vector is normally constructed one element at a time by iterating over s:

$$\Gamma_{a,o}(\alpha)(s) = \frac{R(s, a)}{|\mathcal{O}|} + \gamma \sum_{s' \in S} P_a(o \mid s') P(s' \mid s) \alpha(s'). \tag{3}$$

We can now generate a new set of vectors for $V_{a,o}^{(t+1)}$:

$$\widetilde{\Upsilon}_{a,o}^{(t+1)} = \left\{ \Gamma_{a,o}(\alpha_k) \ :\ \alpha_k \in \Upsilon^{(t)} \right\}.$$

To generate the vectors in $V_a^{(t+1)}$, we will need another definition. Let the *cross sum*, $\oplus$, between two sets of vectors, Υ and Υ', be defined as:

$$\Upsilon \oplus \Upsilon' = \{\alpha + \alpha' | \alpha \in \Upsilon, \alpha' \in \Upsilon'\}.$$

A new set of vectors for $V_a^{(t+1)}$ can be generated by:

$$\widetilde{\Upsilon}_a^{(t+1)} = \bigoplus_{o \in \mathcal{O}} \widetilde{\Upsilon}_{a,o}^{(t+1)}.$$

Finally, we can generate the vectors for $V^{(t+1)}$:

$$\widetilde{\Upsilon}^{(t+1)} = \bigcup_{a \in A} \widetilde{\Upsilon}_a^{(t+1)}.$$

The actual value function is the maximum of these vectors.

As we discussed, the resulting set of vectors often contains redundancies due to dominated vectors. We can represent the value function much more compactly by *pruning* the dominated vectors, leaving only the ones that participate in defining the value function. We define this operation as:

$$\Upsilon^{(t+1)} = prune\,(\widetilde{\Upsilon}^{(t+1)}).$$

```
PRUNE(Υ̃)
    Υ = {};
    PRUNE ALL POINTWISE DOMINATED VECTORS
      FROM Υ̃;
    REPEAT
        FOR SOME VECTOR α ∈ Υ̃ SOLVE:
          b = DOMINATESLP(α, Υ);
        IF α IS DOMINATED
            REMOVE α FROM Υ̃;
        ELSE
            FIND α' = BEST(b, Υ̃);
            Υ = Υ ∪ {α'} ;
    UNTIL Υ̃ IS EMPTY;
    RETURN Υ.
```

Figure 1: Algorithm for performing the pruning operation.

```
POINTWISEDOMINATES(α, Υ)
    FOR EACH α' ∈ Υ
        IF ∀s  α'(s) ≥ α(s) RETURN true;
    RETURN false.
```

Figure 2: Algorithm for checking for pointwise domination.

As shown by Cassandra *et al.*, we can perform this operation incrementally as well, due to the identity:

$$\begin{aligned}\Upsilon_a^{(t+1)} &= prune\,(\widetilde{\Upsilon}_{a,o_1}^{(t+1)} \oplus \widetilde{\Upsilon}_{a,o_2}^{(t+1)} \oplus \ldots \oplus \widetilde{\Upsilon}_{a,o_{|\mathcal{O}|}}^{(t+1)});\\ &= prune\,(prune\,(\widetilde{\Upsilon}_{a,o_1}^{(t+1)} \oplus \widetilde{\Upsilon}_{a,o_2}^{(t+1)}) \ldots \oplus \widetilde{\Upsilon}_{a,o_{|\mathcal{O}|}}^{(t+1)}).\end{aligned}$$

The algorithm to perform the *prune* operation, due to White and Lark [White, 1991], is summarized in Figure 1. There are two ways it can prune vectors from the set. In the simplest case, there might be a pair of vectors where one is larger than the other for all states. The smaller one can be pruned, using the *pointwiseDominates* function in Figure 2. The other case occurs when a vector is not dominated by a single other vector, but it is dominated by a set of vectors, which we call *set dominance*. In this case, one can write a linear program to test for this general type of dominance, as shown in Figure 4. This linear program seeks to find the belief state b such that the difference between the value the vector α gives to that belief state ($\langle b \bullet \alpha \rangle$) and the value given by the set ($\max_{\alpha_k \in \Upsilon} \langle b \bullet \alpha_k \rangle$) is maximal. If this distance d is non-positive, the vector α is dominated by the set and it can be discarded. Otherwise, we must find the best vector in $\widetilde{\Upsilon}$ at the belief state b, as performed by the *best* function in Figure 3, and add it to the minimal set. The *best* function uses a lexicographic "less than" operator $<_{lex}$ to break ties [Littman, 1996].

Once we have computed the value function, we can derive the optimal policy, which is implicitly represented in the value function:

$$\pi(b) = \arg\max_a \sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, a) V(b_o^a).$$

2.2 Policy Iteration

An alternative approach to value iteration is to search in the space of policies. In fully observable MDPs, policy iteration

```
BEST(b, Υ)
    max = −∞;
    FOR EACH α ∈ Υ
        IF {⟨b • α⟩ > max} OR
          {(⟨b • α⟩ = max) AND (α_max <_lex α)}

            α_max = α;
            MAX = ⟨b • α⟩;
    RETURN α_max.
```

Figure 3: Algorithm for finding the vector that gives the maximal value to belief state b.

```
DOMINATESLP(α, Υ)
    SOLVE LINEAR PROGRAM:
        VARIABLES: b_1, ..., b_s, d;
        MAXIMIZE: d;
        SUBJECT TO:
            ⟨b • (α − α_k)⟩ ≥ d   ∀α_k ∈ Υ ,
            Σ_i b_i = 1 , b_i ≥ 0,   ∀i.

    IF d ≤ 0
        RETURN dominated;
    ELSE
        RETURN b.
```

Figure 4: Linear program for checking if the set of vectors Υ dominates a vector α.

is very successful, and often converges much faster than value iteration. Sondik [1978] suggested the use of policy iteration for POMDPs, proposing that the policy be represented as a finite state machine. Hansen [1998] proposed a more practical and implementable version of policy iteration for POMDPs, and proved that it converges to the optimal policy. He also showed, empirically, that, as in MDPs, policy iteration converges faster tha value iteration. In this section, we review Hansen's policy iteration algorithm.

Hansen's algorithm represents policies as finite state machines. The algorithm iterates through policies, starting with some initial finite state machine $\Delta^{(0)}$, the iteration is composed of two steps:

1. **Value determination:** compute the value of acting according to $\Delta^{(t)}$;
2. **Policy improvement:** use the computed value function to update the finite state machine to $\Delta^{(t)}$.

The finite state machine Δ is a tuple $(\mathcal{M}, \mathcal{O}, \pi, \tau)$. For a given machine state $m \in \mathcal{M}$ and observation $o \in \mathcal{O}$, is represented by:

- $\pi(m)$ — the action associate with the machine state m;
- $\tau(m, o)$ — next machine state after observing o at m.

The first key step in Hansen's algorithm is the value determination step. Here, for each finite state machine $\Delta^{(t)}$, we must compute the value of acting according to the policy it represents. Note that, once we are at a particular machine state, the policy is fully determined. Hence, the value function associated with a given finite state machine and a given starting machine state m is a linear value function. We can view the finite state machine in its entirety as representing a choice of policies, where our only choice is the machine state in which we begin. The optimal value function associated with this machine is a maximum of a set of vectors, one for each machine state m.

Based on this insight, we can perform value determination for the machine state by solving a set of linear equations whose unknowns are the coefficients of the linear functions associated with the different machine states:

$$\begin{aligned} \alpha_m(s) \quad = \quad & R(s, \pi(m)) + \\ & \gamma \sum_{s' \in S, o \in \mathcal{O}} P(o \mid s', \pi(m)) \\ & \quad P(s' \mid s, \pi(m)) \alpha_{\tau(m,o)}(s'). \end{aligned} \tag{4}$$

This system contains an $|S|$-dimensional vector α_m for each machine state m; its i^{th} component is the expected discounted value of starting the finite state controller in machine state m when in environment state i. Thus, this linear system contains $|\mathcal{M}| \cdot |S|$ equations and unknowns — one for each of the $|S|$ coefficients of the vectors in each of the $|\mathcal{M}|$ machine states. This linear system can be solved exactly for small problems.

The policy improvement step is, at a high level, similar to the analogous step for MDPs. We construct a policy that is greedy relative to our current value function, and then use that to compute a new value function. For POMDPs, the process is as follows. For each observation o, we select the action that gives the highest payoff, assuming that the value function represents the long-term payoff at the next step. This operation is executed by performing a DP step, giving us a value function with one step of lookahead. This value function is represented as a set of vectors. We then construct a policy that is optimal relative to that one step lookahead, by updating our finite state machine.

More formally, we first take the vectors associated with the current finite state machine $\Delta^{(t)}$ to define the set $\Upsilon^{(t)}$ and perform a DP step, as described above, to obtain the minimal set of vectors $\Upsilon^{(t+1)}$. This set of vectors forms the basis for the definition of the new finite state machine. Note that $\widetilde{\Upsilon}^{(t+1)}$ is the union of $\widetilde{\Upsilon}_a^{(t+1)}$; hence, each vector α in $\widetilde{\Upsilon}^{(t+1)}$ (and hence in $\Upsilon^{(t+1)}$) is associated with some particular action $a[\alpha]$. Furthermore, $\widetilde{\Upsilon}_a^{(t+1)}$ is defined as the cross-sum of sets of vectors, one for each observation. Hence, α is associated with a set of vectors, one for each observation o. Each of these constituent vectors is derived as the backprojection of the linear value function associated with some machine state in $\Delta^{(t)}$; we use $m_o[\alpha]$ to denote this particular machine state. Intuitively, α represents the value function that would be derived from the following policy: first take action $a[\alpha]$, and, upon seeing observation o, go to the machine state $m_o[\alpha]$ in $\Delta^{(t)}$, and behave according to $\Delta^{(t)}$ from then on.

We now define $\Delta^{(t+1)}$ by taking $\Delta^{(t)}$ and updating it using the vectors in $\Upsilon^{(t+1)}$. For each vector $\alpha \in \Upsilon^{(t+1)}$, we perform the following update.

1. If $a[\alpha]$ and the successor links $m_o[\alpha]$ are the same as some state existing machine state in $\Delta^{(t)}$, then simply ignore α.
2. Else, add a machine state m to $\Delta^{(t+1)}$ with $\pi(m) = a[\alpha]$, and $\tau(m, o) = m_o[\alpha]$ for all o. If α dominates a value function vector associated with some existing machine state m' in $\Delta^{(t)}$, then eliminate m', and make all transitions that point to m' point to m instead.

Finally, we prune from $\Delta^{(t+1)}$ any machine state that does not have a corresponding vector in $\Upsilon^{(t+1)}$, as long as it is not reachable from a machine state that has a corresponding vector in $\Upsilon^{(t+1)}$.

3 Factored POMDPs and Linear Value Functions

The exact algorithms presented in the previous sections can find optimal policies for small problems, but have been restricted to problems with tens of states, due to their computational complexity. To attempt to solve more complex problems, Boutilier and Poole [1996] proposed a framework of factored POMDPs that can represent large problems compactly. Furthermore, they propose an algorithm for exploiting this representation to improve the efficiency of exact computation of the value function. Subsequently, Hansen and Feng [2000] extended this algorithm and presented an implementation that can solve larger problems. Both of these approaches make the assumption that the vectors composing the value function can be represented with a tree structure that assigns the same value to many individual components of the vector. If the vectors composing the value function have a structure amenable to this representation, these approaches can give an exponential reduction in the amount of space needed to represent each vector.

However, the vectors composing the value function are not always amenable to a tree-structured representation. As discussed in [Koller and Parr, 1999], the exact value functions even for simple factored systems can grow exponentially in size. Many researchers have proposed the use of a linear approximation, where an approximate value function is represented as a linear combination of basis function. This approach was first proposed for a variety of unfactored MDPs [Tsitsiklis and Van Roy, 1996] and applied to factored MDPs in [Koller and Parr, 2000; Guestrin *et al.*, 2001]. They show that even a small set of basis functions can provide a high-quality approximation to a high-dimensional value function. In this paper, we apply this idea to POMDPs, by using the same approximation for the individual value-function vectors that comprise the POMDP value function. In this section, we show how the value and policy iteration algorithms for factored POMDPs can exploit this compact representation for efficient computation.

3.1 Representation of factored POMDPs

In a factored POMDP, 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 $\mathrm{Dom}(X_i)$. A state $\mathbf{x}$ defines a value $x_i \in \mathrm{Dom}(X_i)$ for each variable X_i. As in the general POMDP framework, each action specifies a transition model and an observation model. In the case of factored POMDPs, both of these are represented as a *dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989].

Let X_i denote the variable X_i at the current time and X'_i the variable at the next step. The *transition graph* associated with an action a is a two-layer directed acyclic graph G_a whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$. We denote the parents of X'_i in the graph by $\mathit{Parents}_a(X'_i)$. For simplicity of exposition, we assume that $\mathit{Parents}_a(X'_i) \subseteq \mathbf{X}$; i.e., all arcs in the DBN are between variables in consecutive time slices. (This assumption can be relaxed, but our algorithm becomes somewhat more complex.) Each node X'_i is associated with a *conditional probability distribution (CPD)* $P_a(X'_i \mid \mathit{Parents}_a(X'_i))$. The transition probability $P_a(\mathbf{x}' \mid \mathbf{x})$ is then defined to be $\prod_i P_a(x'_i \mid \mathbf{u}_i)$, where $\mathbf{u}_i$ is the value in $\mathbf{x}$ of the variables in $\mathit{Parents}_a(X'_i)$. The transition dynamics of a POMDP are defined via a separate DBN model $\langle G_a, P_a \rangle$ for each action a. We can now represent the conditional probability distributions associated with the action a by $P_a(X'_i \mid \mathit{Parents}_a(X'_i))$.

Next, we must represent our observation space. Here, our observations are described by a set of observation variables: $\mathbf{O} = \{O_1, \ldots, O_n\}$. We associate a set of observation variables $\mathbf{O}_a$ with each action a, i.e., the set of observable variables can be different for different actions. For simplicity of exposition, we make two assumptions. First, we assume that $\mathit{Parents}_\tau(O_i) \subseteq \mathbf{X}'$; i.e., the observations depend on the state reached after an action is taken. Second, we assume that the observation variables are all leaves in G_a. Therefore, $P_a(\mathbf{o}_a \mid \mathbf{x}')$ can be represented by $P_a(\mathbf{o}_a \mid \mathbf{u}'_a)$, where $\mathbf{u}'_a$ is the value in $\mathbf{x}'$ of the variables in $\mathit{Parents}_a(\mathbf{O}_a)$. As we will see, we will eventually need to assume that the set of parents $\mathit{Parents}_a(\mathbf{O}_a)$ is not too large. In other words, each action focuses the attention of the agent to a certain part of the system. For example, a factory maintenance agent fixing a particular machine can observe only the state of the machine he is fixing, or perhaps a few neighboring machines as well. This assumption is reasonable in many settings. Note, however, that we do not need to make another common assumption [Koller and Parr, 2000], that each action can directly influence only a small subset of the variables in the system. Thus, our factory agent can take a single action that turns off all of the machines in the factory.

Finally, we need to provide a compact representation of the reward function. 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.

Definition 3.1 *A function f is* restricted *to a domain* $\mathbf{C} \subseteq \mathbf{X}$ *if* $f : \mathrm{Dom}(\mathbf{C}) \mapsto \mathbb{R}$. *If f is restricted to* $\mathbf{Y}$ *and* $\mathbf{Y} \subset \mathbf{Z}$*, we will use $f(\mathbf{z})$ as shorthand for $f(\mathbf{y})$ where $\mathbf{y}$ is the part of the instantiation $\mathbf{z}$ that corresponds to variables in $\mathbf{Y}$.*

Let $R^a_1, \ldots, R^a_r$ be a set of functions, where each R^a_i is restricted to variable cluster $\mathbf{W}_i \subset \{X_1, \ldots, X_n\}$. The reward function for state $\mathbf{x}$ is defined to be $\sum_{i=1}^r R^a_i(\mathbf{x}) \in \mathbb{R}$.

3.2 Factored linear value functions

As we discussed, we can think of each vector in the representation of the value function as a value function in itself. For example, in policy iteration, the vector $\alpha_m(s)$ associated with machine state m represents the expected discounted reward obtained by being at state s and following the policy of the finite state machine starting at m.

Our algorithm will compute approximations of the piecewise linear value function by maintaining approximate representations for each vector. In fully observable MDPs, a very popular choice to approximate a value functions in fully observable MDPs uses *linear regression*. Here, we define our space of allowable value functions $\mathcal{V} \in \mathcal{H} \subseteq \mathbb{R}^{|S|}$ via a set of *basis functions* $H = \{h_1, \ldots, h_k\}$. A *linear value function* over H is a function $\mathcal{V}$ that can be written as $\mathcal{V}(s) = \sum_{j=1}^k w_j h_j(s)$ for some coefficients $\mathbf{w} = (w_1, \ldots, w_k)$. We define $\mathcal{H}$ to be the linear subspace of $\mathbb{R}^{|S|}$ spanned by the basis functions H. It is useful to define an $|S| \times k$ matrix A whose columns are the k basis functions, viewed as vectors. Our approximate value function is then represented by $A\mathbf{w}$.

The idea of using linear value functions for dynamic programming was proposed, initially, by Bellman et al. [1963] and has been further explored recently [Tsitsiklis and Van Roy, 1996; Koller and Parr, 1999; 2000; Guestrin *et al.*, 2001]. The basic idea is as follows: in the solution algorithms, whether value iteration or policy iteration, we use only value functions within $\mathcal{H}$. Whenever the algorithm takes a step that results in a value function $\mathcal{V}$ that is outside this space, we *project* the result back into the space by finding the value function within the space which is close to $\mathcal{V}$.

In the case of factored MDPs, it was argued that many problems can be well-approximated using a linear combination of functions each of which refers only to a small number of variables. More precisely, a value function is said to be a *factored (linear) value function* if it is a linear value function over the basis $h_1, \ldots, h_k$, where each h_i is restricted to some subset of variables $\mathbf{C}_i$. In our factory example, we might have a basis function for the (binary) variable representing the state of each machine in the factory; the basis function will have the value 1 if the machine is operational, and 0 otherwise. We might also have basis functions for pairs of machines that are directly correlated, in that the output of one is the input to the other.

As shown for the fully observable MDP case in [Koller and Parr, 2000; Guestrin *et al.*, 2001], factored value functions provide the key to doing efficient computations over the exponential-sized state sets that we have in factored MDPs. The key insight is that restricted domain functions (including our basis functions) allow certain basic operations to be implemented very efficiently.

In the context of POMDPs, each vector α will be represented as a linear combination of basis function h_i:

$$\alpha(\mathbf{X}) = \sum_{i=1}^{K} w_{\alpha,i}\, h_i(\mathbf{C}_i).$$

This representation can be exploited for computational benefits. For example, we can compute the value of a belief state according to a vector α compactly by:

$$\begin{aligned}\langle b \bullet \alpha \rangle &= \sum_{\mathbf{x} \in \mathbf{X}} \sum_{i=1}^{K} w_{\alpha,i} h_i(\mathbf{x}) \cdot b(\mathbf{x}); \\ &= \sum_{i=1}^{K} w_{\alpha,i} \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i)\, b_i(\mathbf{c}_i); \qquad (5)\end{aligned}$$

where $b_i(\mathbf{c}_i) = \sum_{\mathbf{x}\in\mathbf{X}_{\mathbf{c}_i}} b(\mathbf{x})$, using $\mathbf{X}_{\mathbf{c}_i}$ to refer to settings to all variables in $\mathbf{X}$ that are consistent with $\mathbf{c}_i$. These $b_i(\mathbf{c}_i)$ represent the marginal of the belief state over the variables in $\mathbf{C}_i$. Therefore, we can compute the value of a belief state exactly by only summing over $b_i(\mathbf{c}_i)$ and not the full, exponentially large, belief state $b(\mathbf{x})$.

Factored linear value functions also admit an efficient implementation of two other important operations. The first is to find the maximum of a factored linear function F over the exponentially large state space S. More precisely, assume that we have a function $F = \sum_{j=1}^{m} f_j(\mathbf{Z}_j)$, which is a linear combination of m functions f_j, each one with domain restricted to $\mathbf{Z}_j$. Our goal is to find $\max_{\mathbf{x}} F(\mathbf{x})$, i.e., to find the state $\mathbf{x}$ over which F is maximized. As observed by Koller and Parr [2000], we can maximize such a function F using *non-serial dynamic programming* [Bertele and Brioschi, 1972] or *cost networks* [Dechter, 1999]. See [Guestrin *et al.*, 2001] for a description of the algorithm.

The second key computational step is a projection of a vector into the linear subspace induced by a set of basis functions. The form of the projection depends on our choice of norm. More formally:

Definition 3.2 *A* projection operator Π *is a mapping* $\Pi : \mathbb{R}^{|S|} \rightarrow \mathcal{H}$. Π *is said to be a* projection w.r.t. a norm $\|\cdot\|$ *if:* $\Pi\mathcal{V} = A\mathbf{w}^*$ *such that* $\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|A\mathbf{w} - \mathcal{V}\|$. ∎

Several norms have been previously used, weighted-$\mathcal{L}_2$ in [Koller and Parr, 1999], $\mathcal{L}_2$ in [Koller and Parr, 2000] and max-norm ($\mathcal{L}_\infty$) in [Guestrin *et al.*, 2001]. For our purposes, either $\mathcal{L}_2$ or $\mathcal{L}_\infty$ would be possible. We present the rest of the paper using the $\mathcal{L}_\infty$ projection, which has better theoretical motivation and good experimental performance [Guestrin *et al.*, 2001].

The *maxnorm projection* is also known as the task of finding the Chebyshev solution to an overdetermined linear system of equations [Cheney, 1982]. The problem is defined as finding $\mathbf{w}^*$ such that:

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|C\mathbf{w} - \mathbf{b}\|_\infty . \tag{6}$$

We denote this projection operation by Π_∞.

Our focus is on cases where $C\mathbf{w}$ has the form $\sum_j w_j f'_j(\mathbf{Z}_j)$, for $\mathbf{Z}_j$ a subset of $\mathbf{X}$, and similarly, $\mathbf{b}$ is a factored linear function. In other words, we want to approximate a factored function $\mathbf{b}$ as a linear combination of particular basis functions, each with a small domain. As discussed by Guestrin *et al.* [2001], the solution of Eq. (6) can, in general, be performed using a linear program over the state space. More importantly, they show that the linear program can be reformulated to use an alternative set of variables, based on the factored representation of the functions. Hence, the max-norm projection can be performed effectively, without having to resort to an explicit enumeration of the entire exponentially-sized state space. See [Guestrin *et al.*, 2001] for the detailed algorithm.

4 Efficient algorithms for factored POMDPs

In this section, we will show how the basic operations in the Incremental Pruning algorithm and policy iteration for POMDPs can be executed efficiently for factored POMDPs using a factored linear approximate value function. The main operations we must deal with are: the DP step, in particular, computing backprojections and testing for dominance; and value determination. The first of these two steps is necessary for both algorithms, whereas the latter is used only in the policy iteration algorithm. The remaining operations in both algorithms can easily be implemented in a way that does not grow with the size of the state space. Therefore, if we find efficient algorithms for these three main operations, we have an efficient implementation for both Incremental Pruning and policy iteration for factored POMDPs.

4.1 Factored DP step

A key step in both algorithms is the DP step, which takes a t-step value function and generates the associated $t+1$-step value function. In both cases, the basic operation is the backprojection of a vector α (Eq. (3)).

4.1.1 Factored backprojection

As observed by Koller and Parr [1999] in the context of MDPs, the backprojection of a value function whose domain is restricted to some set $\mathbf{Y}'$ is a function whose domain is restricted to the *backprojection* of $\mathbf{Y}'$ — the parents of $\mathbf{Y}$ in the transition model. More formally (with some abuse of notation), we define the *backprojection of* $\mathbf{Y}$ *through* a as the set of parents of $\mathbf{Y}'$ in the transition graph G_a as $\Gamma_a(\mathbf{Y}') = \cup_{Y'\in\mathbf{Y}'} \mathit{Parents}_a(\mathbf{Y}')$. We can now show that:

$$\Gamma_{a,\mathbf{o}}(\alpha)(\mathbf{x}) = \frac{\sum_{i=1}^{r} R_i^a(\mathbf{x})}{|\mathbf{O}_a|} + \gamma \sum_{i=1}^{K} w_{\alpha,i} \sum_{\mathbf{u}'_a \in \mathit{Parents}_a(\mathbf{O}_a)} P_a(\mathbf{o}_a \mid \mathbf{u}'_a) P_a(\mathbf{u}'_a \mid \Gamma_a(\mathbf{u}'_a)) \sum_{\mathbf{c}'_i \in \mathbf{C}'_i} P_a(\mathbf{c}'_i \mid \Gamma_a(\mathbf{C}'_i), \mathbf{u}'_a) h_i(\mathbf{c}'_i),$$

where the settings to $\Gamma_a(\mathbf{u}'_a)$ and $\Gamma_a(\mathbf{C}'_i)$ on the right hand side of the conditioning bar in P represent the assignment to these variables specified in $\mathbf{x}$. Note that the conditioning on $\mathbf{u}'_a$ in the term $P(\mathbf{c}'_i \mid \Gamma_a(\mathbf{C}'_i), \mathbf{u}'_a)$ is necessary when $\mathbf{C}'_i \cap \mathit{Parents}_a(\mathbf{O}_a) \neq \emptyset$, to guarantee that the settings of $\mathbf{c}'_i$ used in the summation are consistent with the value of $\mathbf{u}'_a$.

Therefore, the vector resulting from $\Gamma_{a,\mathbf{o}}(\alpha)$ is composed of the sum of restricted domain functions, each one having domain restricted to the backprojection of the basis functions union with the backprojection of the observation variables: $\Gamma_a(\mathbf{C}'_i) \cup \Gamma_a(\mathit{Parents}_a(\mathbf{O}_a))$. If the transition model is sparse, so that variables have a small number of parents, and our basis functions and observation sets are not too large, these component value functions can be compactly represented and manipulated.

4.1.2 Projection of vectors

Note that the function generated by the backprojection of vectors, $\Gamma_{a,o}(\alpha)$, need not be in the space spanned by the basis functions. Therefore, we must project it back into that space. Thus, we are interested in finding the set of weights $\mathbf{w}^*$ such that $\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|A\mathbf{w} - \Gamma_{a,o}(\alpha)\|_\infty$. Note that, as we showed in Section 4.1.1, $\Gamma_{a,o}(\alpha)$ is a linear combination of restricted domain functions. As we discussed in Section 3.2, this optimization can be performed efficiently in closed form by solving a linear program.

Therefore, we will generate the vectors for $V_{a,o}^{(t+1)}$ by first applying $\Gamma_{a,o}(\alpha)$ and then applying Π_∞ to make sure they are in the space spanned by the basis functions. We can now define a non-minimal set of vectors for our approximation of $V_{a,o}^{(t+1)}$:

$$\widetilde{\Upsilon}_{a,o}^{(t+1)} = \left\{ \Pi_\infty[\Gamma_{a,o}(\alpha_k)] \ : \ \alpha_k \in \Upsilon^{(t)} \right\}.$$

4.2 Testing for dominance

The next key operation in incremental pruning is to eliminate dominated vectors from a set of vectors $\widetilde{\Upsilon}$ to obtain a minimal set Υ. As discussed in Section 2.1, there are two types of dominance. The simplest, *pointwise dominance*, occurs when there is a pair of vectors, where one is smaller than the other in every state. The second, more general, *set dominance* occurs when a vector is not dominated by a single other vector, but by a set of vectors. In this section, we show how to test for these two types of dominance in factored POMDPs, without explicitly enumerating the exponentially large state space.

4.2.1 Pointwise dominance

In the pointwise dominance testing, for some pair of vectors α and α', the algorithm checks if $\alpha(s) \leq \alpha'(s)$, for each state of the POMDP. In the explicit case, illustrated in Figure 2, one must perform this test for every state. However, in factored POMDPs, this procedure would have a computational cost which is exponential in the number of state variables, making it intractable.

Fortunately, if both vectors are represented as a linear combination of basis functions, then we can reformulate this question as a test for whether

$$\max_{\mathbf{x}} \left[\sum_{i=1}^{K} (w_{\alpha,i} - w_{\alpha',i}) \ h_i(\mathbf{c}_i) \right] \leq 0,$$

where $\mathbf{c}_i$ is the value of the variables in $\mathbf{C}_i$ in the assignment $\mathbf{x}$. This formulation is equivalent to the pointwise dominance question: the maximum will be negative if and only if $\alpha(\mathbf{x}) \leq \alpha'(\mathbf{x})$ for all states. It is easy to verify that we can test this condition efficiently for factored linear value functions using the efficient algorithm of maximization over state space, discussed in Section 3.2. More precisely, we can apply that algorithm with $f_j(\mathbf{C}_j) = (w_{\alpha,j} - w_{\alpha',j}) \ h_j(\mathbf{C}_j)$.

4.2.2 Set dominance

The second type of dominance necessary for pruning dominated vectors is *set dominance*. Here, we are interested in testing if a vector α is dominated by a set of vectors Υ. As described in Section 2.1, this test can be performed, in the explicit state space case, by solving a linear program, shown in Figure 4 . This linear program seeks to find the belief state b such that the difference between the value the vector α gives to that belief state ($\langle b \bullet \alpha \rangle$) and the value given by the set ($\max_{\alpha_k \in \Upsilon} \langle b \bullet \alpha_k \rangle$) is maximal. If this distance d is non-positive, the vector α is dominated by the set and it can be discarded.

The problem with this explicit formulation of the linear program is that it contains variables $b_1, \ldots, b_{|S|}$ that represent the belief for every state s. However, the number of states is exponential in the number of state variables. Thus, this linear program requires exponentially many variables.

However, as we show in this section, our factored representation for vectors allows us to generate a compact linear program to test for dominance. First, note that these belief variables b_i are only necessary to represent the constraint $\langle b \bullet (\alpha - \alpha') \rangle \geq d$. In Section 3.2, we showed that we do not need to have an explicit representation of the belief state to compute such dot products. We only need the marginals over the variables in the domains of the basis functions to compute the dot product exactly. This is shown in Eq. (5), which we repeat here:

$$\langle b \bullet \alpha \rangle = \sum_{i=1}^{K} w_{\alpha,i} \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i) \ b_i(\mathbf{c}_i);$$

where the domain of each basis function h_i is restricted to a subset of the variables $\mathbf{C}_i$. This simplification hints that our linear program does not need a variable for the belief of every state s, but, more concisely, it needs only to maintain a factored representation of the belief state.

Thus, we might consider reformulating our linear program for set domination as follows:

$$\begin{array}{ll} \text{Variables:} & d, \{b_{\mathbf{c}_i}\}; \\ \text{Maximize:} & d \\ \text{Subject to:} & \sum_{i=1}^{K} (w_{\alpha,i} - w_{\alpha',i}) \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i) b_{\mathbf{c}_i} \geq d, \\ & \hfill \forall \alpha' \in \Upsilon; \\ & \{b_{\mathbf{c}_i}\} \text{ represents a legal belief state.} \end{array} \tag{7}$$

Unfortunately, this straightforward formulation is not adequate, because it is not easy to ensure that the variables $b_{\mathbf{c}_i}$ are the marginals associated with a single coherent probability distribution b. Assume, for example, that our state space is defined via the variables A, B, C, D, and that we have four clusters: $\mathbf{C}_1 = \{A, B\}$, $\mathbf{C}_2 = \{B, C\}$, $\mathbf{C}_3 = \{C, D\}$, and $\mathbf{C}_4 = \{D, A\}$. We are given four distributions over these four clusters, and we would like to guarantee that they are all derived from a single joint distribution over A, B, C, D. It is easy to check that the marginals are locally consistent; for example, we can easily construct linear equations that represent the constraint that $b_1(b) = b_2(b)$ for all b. However, local consistency does not, in general, imply global consistency, and it is easy to construct examples where each of the marginals is locally consistent but there is no single joint distribution that is consistent with all of them.

We can address this problem using the notion of *decomposable models* [Lauritzen and Spiegelhalter, 1988]. In these models, local consistency does imply global consistency. First, we construct a graph in which the nodes are the variables in our distribution and we have an edge between two nodes if the variables appear in a cluster together. The graph for the clusters above is shown in Figure 5, without the dashed edge. We can now triangulate the graph, i.e., add edges so that all loops of length greater than three have at least one edge that "cuts across" the loop. For example, we might add

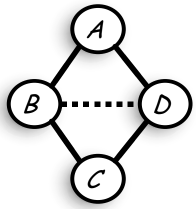

Figure 5: Nondecomposable model for probability distribution.

the dashed edge between B and D. We can now construct a set of *cliques*, which are maximal fully-connected subgraphs in this graph. In our example, we have two cliques, which are A, B, D and B, D, C.

We can now consider marginal distributions over the cliques, i.e., $\{b_{a,b,d}\}$ and $\{b_{b,d,c}\}$. It is straightforward to verify that, if these two sets of numbers are distributions, and if they agree on the marginals over the shared variables B and D, then they are consistent with some joint distribution over A, B, C, D. More generally, each one of our original clusters $\mathbf{C}_i$ will be a subset of some clique in this graph. We denote the variables in this clique by $\overline{\mathbf{C}}_i$, and use $\overline{\mathbf{c}}_i$ to denote an assignment to those variables. We now define a *clique tree*, whose nodes are the cliques in the graph, and whose edges are selected to satisfy the *running intersection property*: given two cliques in the clique tree $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{C}}_j$, if we have that variable $Y \in \overline{\mathbf{C}}_i \cap \overline{\mathbf{C}}_j$, then Y must be in every clique that is on the path between them in the tree. Let the separators $\mathbf{S}_{i,j} \in \mathcal{S}$ be the intersection between two clusters that are directly connected in the clique tree: $\mathbf{S}_{i,j} = \overline{\mathbf{C}}_i \cap \overline{\mathbf{C}}_j$, if there is an edge between $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{C}}_j$ in the clique tree. An assignment to the separator variables is denoted by $\mathbf{s}_{i,j}$. We use $\mathrm{Dom}(\overline{\mathbf{C}}_i \mid \mathbf{s}_{i,j})$ to represent the assignments of values to $\overline{\mathbf{C}}_i$ that are consistent with the assignment $\mathbf{S}_{i,j} = \mathbf{s}_{i,j}$. We can now test whether a set of marginal distributions $\{b_{\overline{\mathbf{c}}_i}\}$ represents a coherent probability distribution by testing whether

$$\sum_{\overline{\mathbf{c}}_i \in \mathrm{Dom}(\overline{\mathbf{C}}_i \mid \mathbf{s}_{i,j})} b_{\overline{\mathbf{c}}_i} = \sum_{\overline{\mathbf{c}}_j \in \mathrm{Dom}(\overline{\mathbf{C}}_j \mid \mathbf{s}_{i,j})} b_{\overline{\mathbf{c}}_j} \ .$$

Using this construction, we can now define a factored linear program that precisely solves the set domination problem. We simply use the variables $b_{\overline{\mathbf{c}}_i}$ rather than $b_{\mathbf{c}_i}$ in our LP, as shown in Figure 6. Note that the inequality constraint

$$\sum_{i=1}^{K} (w_{\alpha,i} - w_{\alpha',i}) \sum_{\overline{\mathbf{c}}_i \in \overline{\mathbf{C}}_i} h_i(\overline{\mathbf{c}}_i)\, b_{\overline{\mathbf{c}}_i} \geq d$$

is the factored representation for $\langle b \bullet (\alpha - \alpha') \rangle \geq d$, where the belief state b is represented compactly by $\{b_{\overline{\mathbf{c}}_i}\}$. Thus, we can check for dominance for any belief state b in closed form by considering only belief states $\{b_{\overline{\mathbf{c}}_i}\}$, yielding an exponential saving in the size of the linear program.

The techniques provided so far allow the DP step to exploit the structure of a factored POMDP in order to speed-up the computation. They allow us to implement an approximate version of value iteration for factored POMDPs.

FACTOREDDOMINATESLP(α, Υ)
SOLVE LINEAR PROGRAM:
VARIABLES: d, AND
$b_{\overline{\mathbf{c}}_i}$ $\forall\, \overline{\mathbf{c}}_i \in \overline{\mathbf{C}}_i$, $\forall \overline{\mathbf{C}}_i \in \overline{\mathcal{C}}$;
MAXIMIZE: d;
SUBJECT TO:
$\sum_{i=1}^{K} (w_{\alpha,i} - w_{\alpha',i}) \sum_{\overline{\mathbf{c}}_i \in \overline{\mathbf{C}}_i} h_i(\overline{\mathbf{c}}_i)\, b_{\overline{\mathbf{c}}_i} \geq d$
$\forall \alpha' \in \Upsilon$;
$\sum_{\overline{\mathbf{c}}_i \in \overline{\mathbf{C}}_i} b_{\overline{\mathbf{c}}_i} = 1$, $b_{\overline{\mathbf{c}}_i} \geq 0$, $\forall i$;
$\sum_{\overline{\mathbf{c}}_i \in \overline{\mathbf{C}}_i^{(\mathbf{s}_{i,j})}} b_{\overline{\mathbf{c}}_i} = \sum_{\overline{\mathbf{c}}_j \in \overline{\mathbf{C}}_j^{(\mathbf{s}_{i,j})}} b_{\overline{\mathbf{c}}_j}$, $\forall \mathbf{s}_{i,j} \in \mathbf{S}_{i,j}$, $\forall \mathbf{S}_{i,j} \in \mathcal{S}$.

IF LP IS INFEASIBLE OR $d \leq 0$
RETURN *dominated*;
ELSE
RETURN $\{b_{\overline{\mathbf{c}}_i}\}$;

Figure 6: Factored Linear program for checking if the set of vectors Υ dominates a vector α.

4.3 Factored value determination

To provide a factored algorithm for policy iteration, we need to provide an efficient algorithm for one additional task: the task of *value determination* — approximating the value of a policy represented as a finite state machine. As in the explicit case, each machine state m is associated with a vector α_m, where $\alpha_m(\mathbf{x})$ represents the value of being at state $\mathbf{x}$ and following the policy described by the finite state machine starting from machine state m. In the explicit case, we can compute these values exactly by solving the set of linear equations shown in Eq. (4). Again, the number of states is exponential in the number of state variables, making the exact computation of these values intractable. Thus, we will resort to the same approximation scheme, using factored linear value functions.

In this approximation framework, each vector is represented by weights $\mathbf{w} = (w_1, \ldots, w_k)$. There are $|\mathcal{M}|$ such vectors, one for every machine state. We will use $w_{m,i}$ to denoted the weight the vector associated with machine state m gives to basis function h_i.

We can now formalize this approximation problem: we want to find the weights for all vectors simultaneously, such that the value determination equations, Eq. (4), are satisfied as well as possible in terms of max-norm error. In other words, we are trying to find an approximate set of value functions $A\mathbf{w}_m$, one for each machine state m, that minimize the max-norm difference between the approximate value functions and their backprojection. Thus, we want $A\mathbf{w}_m$ to be close to

$$\sum_{\mathbf{o} \in \mathbf{O}_{\pi(m)}} \Gamma_{\pi(m),\mathbf{o}}(A\mathbf{w}_{\tau(m,\mathbf{o})}).$$

Based on Eq. (7), this problem can be written for factored

POMDPs, as:

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_{|\mathcal{M}|}} \max_{m} \left\| \sum_{i=1}^{K} w_{m,i}\, h_i(\mathbf{c}_i) - R(\mathbf{x},\pi(m)) - \gamma \sum_{\mathbf{x}'\in\mathbf{X}'} P(\mathbf{x}' \mid \mathbf{x},\pi(m)) \sum_{\mathbf{o}\in\mathcal{O}} P(\mathbf{o} \mid \mathbf{x}',\pi(m)) \sum_{i=1}^{K} w_{\tau(m,\mathbf{o}),i}\, h_i(\mathbf{c}'_i) \right\|_{\infty} . \quad (8)$$

Note that the w's appear both in the value function (the left-hand term) and in its backprojection (the right-hand term). However, it is easy to manipulate the expression so that it has the form of Eq. (6), as in the value determination algorithm for fully observable MDPs of [Guestrin *et al.*, 2001]. As we discussed in Section 3.2, this optimization can be performed efficiently by solving a factored linear program.

Thus, we can exploit the structure in factored POMDPs to find efficiently approximations to the value of a policy represented as a finite state machine. The policy improvement step can also be implemented efficiently unchanged, using the techniques described above for testing dominance. Hence, we can perform approximate policy iteration efficiently in factored POMDPs.

5 Discussion and future work

In this paper, we presented new algorithms for performing approximate value and policy iteration in POMDPs. These algorithms approximate each vector that composes the piecewise linear convex value functions by a linear combination of basis functions, thus dealing with problem of exponentially large representations of vectors. Furthermore, this representation allows for the operations in approximate value and policy iteration to be performed efficiently for factored POMDPs. We show how factored structure can be exploited for an approximate version of the Incremental Pruning algorithm of [Cassandra *et al.*, 1997] and of the policy iteration for POMDPs algorithm of [Hansen, 1998].

An interesting next step would be to deal directly with simultaneous factored observations, where many observation variables are observed at every time step.

References

[Bellman *et al.*, 1963] R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation – a new computational technique in dynamic programming. *Math. Comp.*, 17(8):155–161, 1963.

[Bertele and Brioschi, 1972] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, New York, 1972.

[Boutilier and Poole, 1996] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, August 1996. AAAI Press.

[Cassandra *et al.*, 1997] A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental prunning: A simple, fast, exact method for partially observable markov decision processes. In *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference*, pages 54–61, Providence, Rhode Island, August 1997. Morgan Kaufmann.

[Cheney, 1982] E. W. Cheney. *Approximation Theory*. Chelsea Publishing Co., New York, NY, 2nd edition, 1982.

[Dean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[Guestrin *et al.*, 2001] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, Washington, August 2001. Morgan Kaufmann.

[Hansen and Feng, 2000] Eric Hansen and Zhengzhu Feng. Dynamic programming for pomdps using a factored state representation. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, Colorado, April 2000.

[Hansen, 1998] Eric Hansen. *Finite-Memory Control of Partially Observable Systems*. PhD thesis, University of Massachusetts Amherst, Amherst, Massachusetts, 1998.

[Koller and Parr, 1999] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*. Morgan Kaufmann, 1999.

[Koller and Parr, 2000] D. Koller and R. Parr. Policy iteration for factored mdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, Stanford, California, June 2000. Morgan Kaufmann.

[Lauritzen and Spiegelhalter, 1988] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50(2):157–224, 1988.

[Littman, 1996] Michael Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Department Computer Science, Brown University, Providence, Rhode Island, 1996.

[Madani *et al.*, 1999] O. Madani, A. Condon, and S. Hanks. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision process problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Florida, July 1999. AAAI Press.

[Smallwood and Sondik, 1973] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Sondik, 1978] E. J. Sondik. The optimal control of partially observable Markov decision processes over the infinite horizon: Discounted costs. *Operations Research*, 26:282–304, 1978.

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

[White, 1991] C. C. White. A survey of solution techniques for partially observed Markov decision processes. *Annals of Operations Research*, 32:215–230, 1991.