Piecewise Linear Value Function Approximation for Factored MDPs

Pascal Poupart and Craig Boutilier
Dept. of Computer Science
University of Toronto
Toronto, ON, M5S 3H5
ppoupart,cebly@cs.toronto.edu

Relu Patrascu and Dale Schuurmans
Department of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
rpatrasc,dale@cs.uwaterloo.ca

Abstract

A number of proposals have been put forth in recent years for the solution of Markov decision processes (MDPs) whose state (and sometimes action) spaces are *factored*. One recent class of methods involves linear value function approximation, where the optimal value function is assumed to be a linear combination of some set of basis functions, with the aim of finding suitable weights. While sophisticated techniques have been developed for finding the best approximation within this constrained space, few methods have been proposed for choosing a suitable basis set, or modifying it if solution quality is found wanting. We propose a general framework, and specific proposals, that address both of these questions. In particular, we examine *weakly coupled MDPs* where a number of subtasks can be viewed independently modulo resource constraints. We then describe methods for constructing a piecewise linear combination of the subtask value functions, using greedy decision tree techniques. We argue that this architecture is suitable for many types of MDPs whose combinatorics are determined largely by the existence of multiple conflicting objectives.

1 Introduction

Markov decision processes (MDPs) form the foundations of most recent work in decision-theoretic planning and reinforcement learning. Classical solution techniques for MDPs, however, generally rely on explicit state and action space enumeration, and thus suffer from the "curse of dimensionality." Specifically, since realistic domains are often *factored*—that is, the state space consists of assignments of values to a set of variables—they have states spaces that grow exponentially with the number of relevant variables.

Fortunately, the factored nature of an MDP often admits compact representation [7; 3]. For example, dynamic Bayes nets (DBNs) can be used to represent the dynamics of the MDP, taking advantage of the fact that actions tend to have independent effects on state variables, and that these effects depend only on the status of a small set of other variables [6; 3]. Additive reward functions can also be used to great effect [2; 12]. Methods exist for exploiting these forms of structure when solving an MDP, obviating the need for state space enumeration, and producing compact representations of value functions (VFs) and policies. These include exact and approximate methods for piecewise constant representations [7; 3; 10; 5] and feature-based approaches [1; 18].

Among feature-based models, *linear approximations* have proven popular. In linear approximations, a small set of basis functions (over state space) is assumed, and the VF is taken to be a linear combination of these functions. Recently, several clever proposals have shown how to find the best linear approximation, given a fixed basis set, in a way that exploits the factored nature of an MDP [8; 16; 9]. These models use basis functions over a small set of variables and DBN action representations to ensure computation is effective. These approaches have the potential to scale well for certain classes of problems.

The main drawback of linear models is the need for a good basis set. While these approaches may scale, the quality of the approximation depends critically on the underlying basis. If no decent approximate VF lies in the subspace spanned by the basis, it is impossible to obtain good solutions using such techniques. Unfortunately, in the recent work on linear approximations for factored MDPs, no proposals exist for either: (a) the choice of a good basis; or (b) the modification of an existing basis to improve decision quality. Studies to date have used simple characteristic functions over (very small) subsets of state variables.

We address both of these problems in this paper. We first describe one technique for the generation of a suitable basis set, based on the notion of *subtask value functions*: these arise naturally in *weakly coupled MDPs (WCMDPs)* [12], a general class of large, factored MDPs. A WCMDP is one in which a process can be decomposed into a number of subprocesses corresponding to distinct objectives, with each of these subprocesses coupled in a weak sense. The weakly coupled nature of an MDP can be discovered through analysis of its DBN representation. Our first technique for basis function generation exploits weak coupling and can be thought of as relying on domain-specific properties. We then describe a general framework for the incremental construction of a suitable basis for linear approximation of a factored MDP. This approach relies on no special domain properties, and can be instantiated in a number of concrete ways [14]. We focus in this paper on a particular instantiation of our framework that allows for the construction of a *piecewise linear (PWL)* combination of basis functions. We argue that this model is especially suited to the solution of WCMDPs, a fact supported by our empirical results.

We begin in Section 2 with a brief overview of factored and weakly coupled MDPs and existing methods for linear approximation for factored MDPs. In Section 3, we describe our general framework for incremental basis function construction, and discuss a decision-tree approach for the construction of PWL combinations of basis functions in Section 4. We offer some preliminary experimental results in Section 5 and conclude in Section 6.

2 Linear Approximations of MDPs

We begin with an overview of MDPs, a discussion of factored and weakly-coupled MDPs, and recent techniques for linear function approximation.

2.1 Markov Decision Processes

We assume a fully-observable MDP with finite sets of states $\mathcal{S}$ and actions $\mathcal{A}$, transition function $\Pr(s, a, t)$, reward function $R(s, a)$, and a discounted infinite-horizon optimality criterion with discount factor β. $\Pr(s, a, t)$ denotes the probability with which the system transitions to state t when action a is taken at state s, while $R(s, a)$ denotes the immediate utility of taking action a at state s. A *stationary policy* $\pi : \mathcal{S} \to \mathcal{A}$ determines a particular course of action. The *value* of a policy π at state s, $V^{\pi}(s)$, is the expected sum of future discounted rewards over an infinite horizon:

$$E_{\pi}[\sum_{t=0}^{\infty} \gamma^t R^t | S^0 = s].$$

The function V^{π} can be computed as the solution to the following linear system:

$$V^{\pi}(s) = R(s, \pi(s)) + \beta \sum_{t \in \mathcal{S}} Pr(s, \pi(s), t) \cdot V^{\pi}(t) \qquad (1)$$

The operator on the r.h.s. of Eq. 1 is referred to as the *backup operator for policy* π, denoted B^{π}; V^{π} is thus a fixed point of B^{π}. We denote by B^a the backup operator for the policy that applies action a at each state.

Our aim is to find a policy π^* that maximizes value at each state. The *optimal VF*, denoted V^*, is unique and is the fixed point of the following *Bellman backup operator* [11]:

$$V^*(s) = \max_{a \in \mathcal{A}} R(s, a) + \beta \sum_{t \in \mathcal{S}} Pr(s, a, t) \cdot V^*(t) \qquad (2)$$

A number of algorithms exist to construct the optimal VF, including dynamic programming algorithms such as value and policy iteration. We focus here on a simple linear program (LP), whose solution is V^*:

$$\textit{Min: } \sum_s V(s) \quad \textit{Subj. to: } V(s) \geq (B^a V)(s), \forall a, s \qquad (3)$$

Here each $V(s)$ is a variable, and the value $(B^a V)(s)$ is a linear function of these variables, as seen in Eq. 1. For many classes of MDPs, exact solution using LP methods is not as effective as using dynamic programming algorithms [15]. The value of the LP formulation, however, becomes apparent when we consider linear approximation [16; 9].

2.2 Factored and Weakly Coupled MDPs

One weakness of the classical MDP formulation is its reliance on explicit transition and reward functions. When the state space of the MDP is *factored*—i.e., when states correspond to the instantiation of state variables—an MDP can often be specified more compactly by exploiting regularities in the reward function and the dynamics [3]. We assume a set of (for simplicity, boolean) state variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Each state is thus a vector $\mathbf{x}$ assigning a value to each variable.

Reward often depends only on the status of a few state variables, or additively on "local" reward functions. We assume

$$R(\mathbf{x}, a) = \sum_{j=1}^{m} R_j(\mathbf{x}_j^r, a)$$

where each R_j is a function that depends on a small subset $\mathbf{X}_j^r \subset \mathbf{X}$, and $\mathbf{x}_j^r$ denotes the restriction of $\mathbf{x}$ to the variables in $\mathbf{X}_j^r$. Similarly, dynamics can often be specified compactly. We assume the effect of each action a can be decomposed into independent effects on each variable X_i, and that its effect on X_i depends on a small subset $\mathbf{X}_i^a \subset \mathbf{X}$ of variables. A local function $\Pr(X_i | a, \mathbf{X}_i^a)$ denotes the distribution over X^i given any assignment to $\mathbf{X}_i^a$. We then have

$$\Pr(\mathbf{x}, a, \mathbf{x}') = \prod_i \Pr(x'_i | a, \mathbf{x}_i^a).$$

We refer to the local function $\Pr(X_i | a, \mathbf{X}_i^a)$ as the conditional probability table or *table* for X_i under action a. This forms the basis of DBN action representations.

This representation allows MDPs to be encoded concisely, requiring space linear in the number of variables if each table R_j or $\Pr(X_i)$ refers to a bounded number of variables. The size of the representation can be reduced even further by using specialized representations for these tables, such as decision trees [3] or ADDs [10]. Furthermore, several techniques can take advantage of this structure to avoid state space enumeration when solving the MDP. If a candidate VF depends on only a few variables, the fact that each variable depends on only a small number of parents ensures that applying a Bellman backup results in a new VF that depends only on a few variables [3].

Finally, this type of representation allows us to identify *weakly coupled MDPs (WCMDPs)*. A WCMDP is one in which the reward function is decomposable as above, and the set of variables *relevant* to each R_j is small. The variables relevant to each R_j are determined as follows [2]: the variables $\mathbf{X}_j^r$ are relevant to R_j; and if X_i is relevant to R_j, then so are the variables $\mathbf{X}_i^a$ for all a.[1] WCMDPs arise in many guises, but most often when the combinatorics of a given problem are largely due to the existence of many competing subobjectives [2; 12]. When determining the variables relevant to one objective, other objective variables do not play a role; thus, the objectives are coupled only through the existence of a common core of relevant variables. Problems that exhibit such structure include resource allocation problems, and scheduling of tasks in multi-user domains. We elaborate on WCMDPs in Section 3.1.

[1] Note the recursive nature of this definition.

2.3 Linear Approximations

A common way to approximate VFs is with linear approximators [18; 8; 16]. Given a small set of *basis functions* $\mathcal{F} = \{f_1, \cdots, f_m\}$ over the state space, a *linear value function* V is defined as $V(s) = \sum_i w_i f_i(s)$, or $V = \mathbf{F}\mathbf{w}$, for some set of coefficients (or *weights*) $\mathbf{w} = \langle w_1, \ldots, w_m \rangle$. Here $\mathbf{F}$ denotes a matrix whose columns are the functions f_i. Unless $\mathbf{F}$ spans a subspace that includes V^*, any linear VF will be, at best, an approximation of V^*. The aim is then to find the best linear approximation of the true VF, using a suitable error metric.

An important challenge, the construction of good linear approximators for factored MDPs, has recently been tackled in [8; 16], resulting in techniques that can find approximately optimal linear approximators in a way that exploits the structure of the MDP without enumerating state space. We assume that each basis function f_j is compact, referring only to a small set of variables $\mathbf{X}_j^f$. Linear value and policy iteration are described in [8], while a factored LP solution technique is presented in [16; 9]. We discuss the method proposed in [16].

The LP formulation of a factored MDP above can be encoded compactly when an MDP is factored. First, notice that the objective function Eq. 3 can be encoded compactly:

$$\sum_{\mathbf{x}} V(\mathbf{x}) = \sum_{\mathbf{x}} \sum_{j} w_j f_j(\mathbf{x}) = \sum_{j} w_j y_j \qquad (4)$$

where $y_j = 2^{n-|\mathbf{x}_j^f|} \sum_{\mathbf{x}_j^f} f_j(\mathbf{x}_j^f)$. Intuitively, each y_j is the sum of the values assigned by function f_j, multiplied by the number of states at which they apply, and can be precomputed. Observe that the variables are the weights $\mathbf{w}$, which determine the values $V(\mathbf{x})$. Second, the set of constraints in Eq. 3 can be encoded compactly by observing that this set is equivalent to[2]

$$\max_{\mathbf{x}} V(\mathbf{x}) - (B^a V)(\mathbf{x}) \geq 0, \forall a \qquad (5)$$

Since V is compactly representable as the sum of compact functions, $(B^a V)$ is similarly representable. Specifically, the construction of $(B^a f_j)$ for basis function f_j can exploit the fact that it refers only to a small subset of variables; the *regression* of f_j through a produces a function that includes only those variables $\mathbf{X}_i^a$ for each $X_i \in \mathbf{X}_j^f$, and variables in $\mathbf{X}_k^r$ [3]. The maximization over $\mathbf{x}$ is nonlinear, but can be encoded using the clever trick of [8]. For a fixed set of weights, a *cost network* can be solved using variable elimination to determine this max without state space enumeration. While this technique scales exponentially with the maximum number of variables in any function (i.e, the functions f_j, $(B^a f_j)$, or intermediate factors constructed during variable elimination), this "local exponential" blow up can often be avoided if more sophisticated representations like ADDs are used [10].

An approach that offers even greater computational savings is the incremental constraint generation technique proposed in [16]. The LP above can be rewritten as minimizing Eq. 4, s.t.

$$\sum_j w_j C_j(\mathbf{x}, a) \geq R(\mathbf{x}, a), \forall \mathbf{x}, a \qquad (6)$$

where $C_j(\cdot, a)$ is a function that refers only to variables $\mathbf{X}_j^f$ and $\mathbf{X}_i^a$ for each $X_i \in \mathbf{X}_j^f$. More precisely, we have

$$C_j(\mathbf{x}, a) = f_j(\mathbf{x}_j^f) - \beta \sum_{\hat{\mathbf{x}}_j^f} \Pr(\hat{\mathbf{x}}_j^f | \mathbf{x}_{j,a}^f, a) f_j(\hat{\mathbf{x}}_j^f)$$

where $\mathbf{x}_{j,a}^f$ refers to the set instantiation of variables $\mathbf{X}_i^a$ for each $X_i \in \mathbf{X}_j^f$. This LP is solved without constraints, then using the cost network technique to compute $\min_{\mathbf{x}} \min_a \sum_j w_j C_j(\mathbf{x}, a)$, the state-action pair that maximally violates the constraints in Eq. 3 is determined. This constraint is added to the LP, which is then resolved.

In matrix form, we can rewrite this LP as

$$\min_{\mathbf{w}} \mathbf{y}^\top \mathbf{w} \textit{ subject to } \mathbf{C}\mathbf{w} \geq \mathbf{r} \qquad (7)$$

where $\mathbf{C}$ is a matrix whose m columns correspond to the functions $C_j(\mathbf{x}, a)$. Thus $\mathbf{C}$ has $|\mathbf{X}||\mathcal{A}|$ rows. The advantage of constraint generation is that the rows of $\mathbf{C}$ are added incrementally, and the LPs being solved are dramatically smaller than those described above: the number of constraints ultimately added is $O(m)$ (i.e., the number of basis functions), considerably smaller than the number of constraints required by the LP generated by the cost network. Once all constraints are generated, the LP constraints are $\mathbf{C}^*\mathbf{w} \geq \mathbf{r}^*$, where $\mathbf{C}^*$ and $\mathbf{r}^*$ are restricted to the $O(m)$ active constraints.

We observe that this LP attempts to minimize L_1-error, not Bellman or L_∞-error, as is usual when solving MDPs. Furthermore, this LP model imposes a one-sided constraint on L_1-error, so it cannot strictly be viewed as minimizing L_1-error. L_∞-error can be tackled directly using algorithms like policy and value iteration [8], but at higher computational cost. The difficulties associated with minimizing different error metrics in the LP context are discussed in [14].

3 Basis Function Selection

While linear approximations scale well, determining *a priori* the solution quality one can obtain using a given basis set is difficult. Ideally, V^* would be an element of the subspace spanned by $\mathcal{F}$, in which case an exact solution could be found. If this is not the case, the quality of the best approximation could be gauged by considering the projection of V^* on this subspace. However, since we do not have access to V^*, choosing a suitable basis set is problematic. Indeed, no serious proposals for this problem exist in the recent literature on factored linear approximations. Since solution quality depends critically on the choice of basis, we must consider methods that allow selection of a good initial basis set, or intelligent revision of a basis if solution quality is unacceptable. We consider both of these problems.

3.1 Subtask Value Functions

In a variety of MDPs, the combinatorial explosion in state space (and often action space) size is caused by the presence

[2] When approximation is used, this LP can be viewed as approximately minimizing L_1-error.

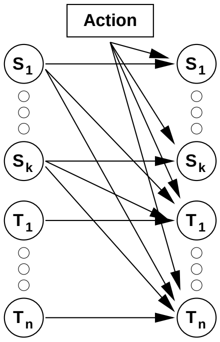

Figure 1: DBN for a generic resource allocation problem.

of multiple, conflicting objectives. For instance, in a manufacturing setting we might need to allocate resources (e.g., machines) to different orders placed by clients. If the process plan for a specific order is more or less fixed, then the problem is one of resource allocation. In an office environment, a robot might be charged with performing tasks of differing priorities for many users.

In problems like these, the underlying MDP is often *weakly coupled*: given a choice of action (e.g., an assignment of resources to each order) each subtask (e.g., order) has a certain small set of state variables that are relevant to determining how best to achieve it, and this subset has little or no overlap with that of other objectives. Thus, each subtask can be viewed as an independent MDP, defined over a much smaller set of variables, that can be meaningfully solved. The subtask MDPs are weakly coupled because their state and action spaces (e.g., feasible resource assignments) are linked: performing a specific action in one subtask MDP influences which actions can be concurrently executed in another (e.g., because it consumes resources).

To illustrate, consider a resource allocation problem with n potential tasks, $T_1, \ldots, T_n$, each of which may be active or inactive, and can change status stochastically (e.g., this might reflect the placement or retraction of orders). We have k resources, each of which can be applied at any point in time to the achievement of any active task. The *status* of resource j, denoted by variable S_j determines how effective that resource is in the completion of its assigned task. The status of a resource evolves stochastically, depending on its use at each time step (e.g., consider machines requiring maintenance or workers needing breaks). Multiple resources can be applied to a task, thus the size of the action space is $O(k^{n+1})$. A DBN illustrating the dependencies for such a problem is illustrated in Figure 1. Finally, we assume that a reward r_i is associated with the successful completion of an active task T_i.

This MDP can be decomposed readily into distinct subtask MDPs for each T_i. Since variables T_j ($j \neq i$) have no influence on T_i or the reward associated with T_i, the subtask MDP for T_i has as its only variables $S_1, \ldots, S_k$ and T_i. For small numbers of resources, this subtask MDP can be solved optimally. Of course, the optimal solutions for the different subtask MDPs may not be compatible. The policies for different subtasks are coupled by the resources—in particular, by constraints on the feasible actions one can apply to jointly to each task. Notice that the action spaces are also considerably reduced in the subtask MDPs.

WCMDPs have been examined recently and several techniques proposed to take advantage of their structure [2; 12; 17]. Given a factored MDP with an additive reward function reflecting subtask structure, constructing a (factored) subtask MDP for each objective is straightforward (see the discussion of relevant variables in Section 2.2) [2]. In the example above, backchaining through the DBN allows us to construct the subtask MDPs for each task, starting only with the variables T_i (which are the only "reward variables").

If a subtask MDP is of manageable size, it can be solved to produce the optimal *subtask value function*, defined on the set of variables relevant to that MDP. All the techniques described in [2; 12; 17] use subtask VFs to great effect to approximate the solution of the full WCMDP. For instance, using heuristic techniques to piece together a global policy using subtask VFs, problems involving several thousand boolean variables (and similarly sized action spaces) can be solved [12].

Subtask VFs are ideal candidates for a basis set. If, for example, we have k subtasks of widely differing priorities (or having different deadlines) the optimal policy might have the form: complete the highest priority subtask (using all resources); then complete the next subtask; and so on. In this case, the optimal VF is: $V(\mathbf{x}) = V^1(\mathbf{x}^1) + \beta^{t_1} V^2(\mathbf{x}^2) + \beta^{t_1+t_2} V^3(\mathbf{x}^3) + \ldots$, where V^i is the VF for subtask i, defined over variables $\mathbf{X}^i$, and t_i is the expected time to completion of task i under the optimal policy. Thus a linear combination of subtask VFs may provide a good approximation.

Unfortunately, a linear combination of subtask VFs may not always be suitable. For instance, if subtasks become active stochastically, the allocation of resources will often depend on the status of each task. One should then focus on a high priority task i (and get value V^i) only if that task is active and suitable resources are available; otherwise one might focus on a lower priority task. Thus the optimal VF might best be approximated by a *piecewise linear* combination of subtask VFs, where different linear approximators are "used" in different regions of state space. For example, a VF might take the form: *If* $\mathbf{c}$, $V(\mathbf{x}) = V^1(\mathbf{x}^1) + \beta^{t_1} V^2(\mathbf{x}^2)$*; if* $\overline{\mathbf{c}}$, $V(\mathbf{x}) = V^3(\mathbf{x}^3) + \beta^{t_3} V^4(\mathbf{x}^4)$. Here tasks 1 and 2 should be tackled when condition $\mathbf{c}$ holds (say these two high priority tasks are active), and tasks 3 and 4 handled otherwise. We elaborate on such PWL approximators in Section 4.

3.2 Basis Function Addition

The use of subtask VFs requires that the underlying MDP exhibit a certain structure. As such, it can be viewed as a domain dependent method for boosting the performance of linear approximators. If domain dependent structure, or other heuristic information, is unavailable, domain independent methods are needed to construct a suitable basis set. For this reason, a more general framework is needed for constructing and revising basis sets. We present such a framework now. This approach is described in much more detail in [14]; but we overview the approach here, since it is relevant to our development of piecewise linear approximators in Section 4.

We assume some set of candidate basis functions $\mathcal{B}$ and an initial basis set $\mathcal{F}_0$. At each iteration k, we compute the best linear approximation w.r.t. $\mathcal{F}_k$, and estimate its error. If the error is unacceptable, and sufficient computation time is available, we then use some *scoring metric* to estimate the improvement offered by each element of $\mathcal{B}$ w.r.t. $\mathcal{F}_k$, and add the best $f \in \mathcal{B}$ to obtain $\mathcal{F}_{k+1}$.

This generic framework can be instantiated in many ways. First, we must define the set $\mathcal{B}$ suitably. We might assume a fixed *dictionary* of candidate basis functions, and score each explicitly. We will adopt this approach below. However, one might also define $\mathcal{B}$ implicitly, and use methods that *construct* a suitable candidate [14].[3]

We also require a scoring metric. An obvious, and computationally demanding, approach would involve adding each candidate function f to $\mathcal{F}_k$ and resolving, in turn, each resulting LP. This gives an exact measure of the value of adding f. Other less demanding approaches are possible. One we consider here is the *dual constraint violation* heuristic.

When we solve the LP Eq. 7, we obtain the corresponding values of the *dual variables* λ_j, one per contraint: because we use constraint generation, all constraints generally will be active, and all $\lambda_j > 0$. If we add f to our current basis set (with corresponding column $\mathbf{c}$ in the LP, and sum of values y), this imposes a new constraint in the dual LP: $\boldsymbol{\lambda}^\top \mathbf{c} \leq y$. If this constraint is satisfied given the current value of $\boldsymbol{\lambda}$, we will make no progress (since the current solution remains optimal). The degree to which this dual constraint is *violated*—i.e., the magnitude of $\boldsymbol{\lambda}^\top \mathbf{c} - y$, provided it is greater than 0—is a good heuristic measure of the value of adding f. Note again that the set of dual variables is $O(m)$ due to incremental constraint generation. The dual constraint violation heuristic scores each basis function in the dictionary using this measure and adds that function to the basis with maximal score.

This framework is inherently greedy: it considers the *immediate* impact of adding a candidate f to the current basis.

4 Piecewise Linear Value Functions

As suggested above, subtask VFs can often best approximate the optimal VF when combined in a piecewise linear fashion. We now describe an algorithm for constructing PWL approximations using subtask VFs as the underlying basis set. Our model uses greedy decision tree construction to determine appropriate regions of state space in which to use different combinations of basis functions. This framework can be seen as a way of incorpating both a domain dependent technique for basis function selection, and a domain independent technique for basis function addition. Indeed, nothing in this approach requires that the underlying basis set comprise the subtask VFs; but we expect WCMDPs to benefit greatly from this model.

The use of decision trees in value function approximation, both in solving MDPs and in reinforcement learning, is rather common. Examples include generalization techniques in reinforcement learning [4], dynamic discretization of continuous state spaces [13], and their use in constructing piecewise constant value function representation for MDPs [3].

4.1 Evaluating Local Splits

We assume a small set of m basis functions $\mathcal{F}$ has been provided *a priori*, with each f_j defined over a small subset $\mathbf{X}_j^f \subset \mathbf{X}$ of our state variables. These might be, say, the optimal subtask VFs for a WCMDP, or a basis constructed using some domain-independent method. The model we adopt is one in which the linear approximation can vary in different parts of state space. These regions are determined by building a decision tree that splits on the variables $\mathbf{X}$.

Before providing details, we illustrate the intuitions by considering a single split of the VF on a fixed variable. Rather than determining the best linear approximator, suppose we allow the weight vector to take on different values, $\mathbf{w}^x$ and $\mathbf{w}^{\overline{x}}$, when variable X is true and false, respectively. So we have:

$$V(\mathbf{x}) = \sum_i w_i^x f_i(\mathbf{x}) \quad \text{for any} \quad \mathbf{x} \in [x]$$

$$V(\mathbf{x}) = \sum_i w_i^{\overline{x}} f_i(\mathbf{x}) \quad \text{for any} \quad \mathbf{x} \in [\overline{x}].$$

Letting $\mathbf{M}^x$ be a “mask” matrix that selects those states where X is true—i.e., a diagonal matrix with 1 at each x-state and 0 at each $\overline{x}$-state—and defining $\mathbf{M}^{\overline{x}}$ similarly, our approximation is

$$V = \mathbf{M}^x \mathbf{F} \mathbf{w}^x + \mathbf{M}^{\overline{x}} \mathbf{F} \mathbf{w}^{\overline{x}} \tag{8}$$

Our goal is to find the optimal *pair* of weight vectors:

$$\textit{Min:} \ \min_{\mathbf{w}^x, \mathbf{w}^{\overline{x}}} \sum_{\mathbf{x} \in [x]} \sum_j f_j(\mathbf{x}) w_i^x + \sum_{\mathbf{x} \in [\overline{x}]} \sum_j f_j(\mathbf{x}) w_j^{\overline{x}}$$

$$\textit{s.t.:} \ B^a(\mathbf{M}^x \mathbf{F} \mathbf{w}^x + \mathbf{M}^{\overline{x}} \mathbf{F} \mathbf{w}^{\overline{x}}) - (\mathbf{M}^x \mathbf{F} \mathbf{w}^x + \mathbf{M}^{\overline{x}} \mathbf{F} \mathbf{w}^{\overline{x}}) \leq 0, \forall a$$

Note that unless the MDP completely decouples along variable X, we must optimize the weights $\mathbf{w}^x$, $\mathbf{w}^{\overline{x}}$ jointly.

This optimization can be performed in exactly the same manner as described in Section 2.3. We observe that for each function f_j, the “masked” version of this depends on the same variables as originally, with the possible addition of X. Furthermore, the dependence on X is trivial: in the positive case, the function takes the constant value 0 if X is false, and takes the value indicated by the original if X is true. An ADD representation of the masked function thus has only one more node than the original (i.e., it does *not* double the size of the function representation). Since these functions are themselves “small,” the same cost network and constraint generation methods can be applied directly.

The approximation above is a *piecewise linear function* over the original basis set, but can also be viewed as *linear* approximator over a *new* basis set. We have replaced the original basis set with the masked copies: the new basis set is

$$\{\mathbf{M}^x f : f \in \mathcal{F}\} \cup \{\mathbf{M}^{\overline{x}} f : f \in \mathcal{F}\}$$

4.2 Decision Tree Construction

The intuitions above suggest an obvious greedy technique for constructing a PWL approximator. We build a decision tree,

[3] We explore a variety of such domain independent basis function construction techniques, and scoring metrics, in [14].

where each interior node splits the state space on some variable X, and each leaf is labeled with a suitable weight vector denoting the linear approximation to be used in that part of the state space.[4] The algorithm is initialized by computing the optimal linear weight vector. The initial tree consists of a single leaf (the root). At each iteration, we extend the current tree as follows: (a) we evaluate the improvement offered by splitting each leaf using each variable, using some scoring metric; (b) the best split is applied, and the optimal PWL VF (or some approximation) for the new tree is computed. The algorithm terminates when no split offers decent improvement, or the tree reaches some size limit.

A key component of the algorithm is the choice of scoring metric. We consider three metrics in this paper:

Full LP: The full LP (FLP) metric evaluates a split of the decision tree by computing the optimal PLW approximator for the extended tree. For a tree with t leaves, evaluating a split requires solving an LP involving $m(t+1)$ weight variables: we have $t-1$ weight vectors for the unsplit leaves, and two new weight vectors for the split leaves.

Fixed Weight LP: The fixed weight LP (FWLP) metric evaluates a split of the decision tree by computing the optimal weight vector for the two new regions created, but holds the weights for each other region fixed (to their values in the preceding solution). Evaluating a split thus requires solving an LP involving $2m$ variables.

Max Dual Constraint Violation: This metric uses the LP solution for the current tree to evaluate the degree of dual constraint violation associated with the new basis functions. A split on X at the end of a branch labeled $\mathbf{y}$ is equivalent to adding the basis functions $\mathbf{M}^{x\mathbf{y}} f_j$ (for each $f_j \in \mathcal{F}$) to the current basis. Each of these new functions is scored using the dual constraint violation heuristic, and the maximum of these scores (over each j) is taken as the score of the split.[5]

These evaluation techniques are listed from most to least expensive. The full LP method finds the myopically optimal split. It requires solving an LP (using the usual cost network method for constraint generation) for each candidate split. These LPs are larger than those for the linear approximator: since we have a larger weight set, we generally need to add more constraints, each requiring a cost network evaluation. The fixed weight LP method is similar, but since we hold all nonsplit weights fixed, there are fewer variables, fewer constraints, and fewer cost networks evaluated (at most twice the number as with the original linear method). The fixed weight technique does not necessarily find the optimal split: since values in other parts of state space are fixed, they are uninfluenced by the change in value at the split states. We can view this as analogous to asynchronous (block) dynamic programming [1]. Once a split is chosen, we can then reoptimize all weights; or if we believe the MDP is strongly decoupled, we might use the weights computed during evaluation to label the split leaves, but not reassess other weights.

The dual constraint violation method is by far the cheapest. Each candidate split can be evaluated using just a handful of inner product computations. No optimization is required.

Finally, with each of these scoring metrics, one heuristically chooses a split by not re-evaluating the scores of previously unsplit nodes. That is, when the leaves of a tree have been scored at one iteration, they are not rescored at a subsequent tree unless they are split. This method is heuristic since the score of a split at a leaf is not local: it depends on the current basis set (viewing the union of basis functions at each leaf as the basis). However, the true score of a leaf can only go down when other leaves are split; its contribution to an extended basis set can be no greater than its contribution to a smaller set. Thus this *fixed score* method always associates with each leaf an upper bound on the true score.

There is a "hidden" cost associated with decision tree construction, since the masked basis functions $\mathbf{M}^{\mathbf{y}} f_j$ at leaf $\mathbf{y}$ refer to all variables along that branch. As the trees get deeper, table-based representations of the functions become much larger. However, as noted above, the ADD representation of these functions (nor their regressions $B^a \mathbf{M}^{\mathbf{y}} f_j$) needn't grow exponentially with the number of variables (i.e., the depth of the tree). Furthermore, the anticipated expense of cost network evaluation can be computed and combined with the scoring metric when considering a split, in an effort to induce a preference for shallower trees.

5 Empirical Results

We describe in this section some very preliminary empirical results. We demonstrate the decision quality of the tree growing technique as a function of the number of splits, using the three scoring metrics described above. We compare this to the optimal linear approximator obtained using subtask value functions, and to that obtained using bases comprising only indicator functions over one or two variables (the only method used in the literature). Naturally, since the best linear approximators are special cases of PWL approximators, decision quality can only improve as we split. What we aim to demonstrate is that quality improves significantly, and that this technique offers a *useful* way to improve a linear approximation. We use the value of the LP objective as a surrogate for quality of the resulting policy in most cases, but report on Bellman error in one example for illustration.

We consider a generic weakly coupled resource allocation problem of the type described in Section 3.1, with n periodic tasks and k indistinguishable resources. When j of the k resources are applied to an active task T_i, there is probability $1-(q_i)^j$ of successfully *completing* that task (q_i is the probability that one unit of resource would *fail* to complete the task, a standard noisy-or model). A completed task becomes inactive. An inactive task i becomes active with probability p_i^{occ} and an active task becomes inactive (if not completed) with probability p_i^{lv}. A reward r_i is obtained if task i is completed. A resource j can be *usable* or *depleted*, indicated by status variable S_j. If usable resource j is applied to a task, it depletes with probability p_j^d, and at each stage a depleted re-

[4] We proceed as if all variables are binary. Binary (aggregate) and multiway splits of multivalued variables are straightforward.

[5] Other ways (e.g., conic combinations) can be used to combine the scores of these basis functions. We note that we only have to consider the scores of one masked set (e.g., X true), since $\mathbf{M}^{x\mathbf{y}} f_j, \mathbf{M}^{\mathbf{y}} f_j$ jointly span $\mathbf{M}^{\overline{x}\mathbf{y}} f_j$.

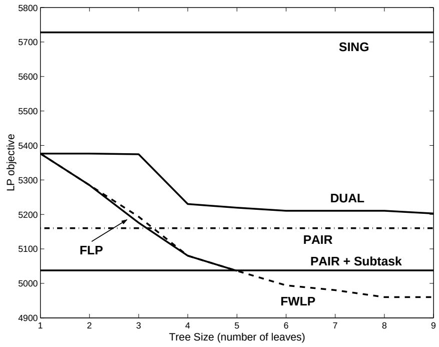

Figure 2: Resource allocation task with no dominant tasks.

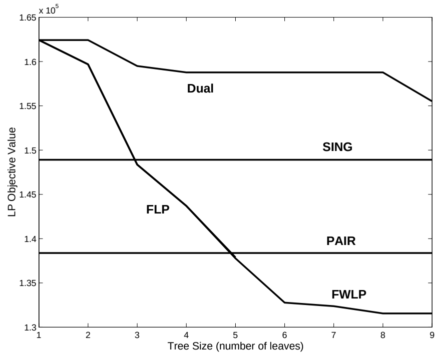

Figure 3: Resource allocation task with two dominant tasks.

source has probability p_j^r of becoming usable again. assigned resources. Since this problem is weakly coupled, we use the subtask value functions for each T_i as an initial basis set.

To illustrate the benefits of PWL approximators, we first consider two small versions of this problem, with $n = 4$ tasks, and $k = 2$ resources. In the first, all tasks have roughly the same level of priority (i.e., similar rewards and probabilities). Figure 2 illustrates the value of the LP objective (which roughly minimizes L_1 error) as a function of the number of regions (i.e. number of decision tree leaves) used in the PWL approximator constructed using each of the scoring metrics described above. As we see, in all cases, decision quality improves with additional splits, which is hardly surprising. We also see that the more expensive scoring metrics are producing much better splits. FLP, since it produces optimal myopic splits, clearly dominates the other methods. FWLP, while much cheaper computationally, also finds improving splits identical to FLP except in one instance. The dual metric, unfortunately, does not fare as well. Note that each curve starts at the same spot: the value of the best linear approximator over the subtask VFs. For comparison, we include the objective value obtained by the best linear approximator over indicator functions on all single variables (SING) and all pairs of variables (PAIR). Note that after very few splits, the PWL approximators provide better VFs than these linear functions.[6]

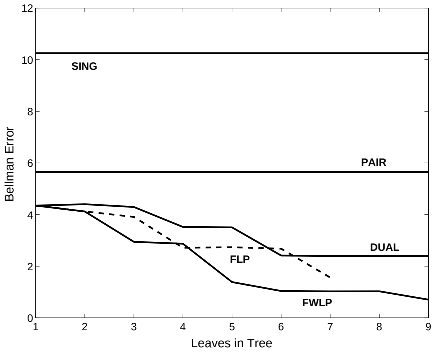

Figure 4: Bellman error for 2 resources, 5 tasks.

We also include the linear approximator over the basis with PAIR indicators and the n subtasks VFs. Adding the subtasks VFs induces substantial improvement over PAIR, indicating their suitability as basis functions. Note that subtask VFs alone do not do as well as pairs, simply because the size of the pairs basis set is substantially larger and spans a larger subspace.

We show the same results in Figure 3 for a variant of the problem in which two of the four tasks have much higher priority than the others. In this case, the values of the low priority tasks have little influence on the optimal value function, since resources are often held in reserve in case a high priority task should pop up. Again we see that the same relative order emerge among the PWL approximators, and that decision quality is better than that of the linear approximators.

We also note that the PWL model can be used to produce piecewise constant VFs using a single constant basis function. In general, if the VF of an MDP has a small decision tree representation, this method will find it quickly.

We also consider some slightly larger problems. Figure 4 shows similar results for a 2-resource, 5-task problem; but Bellman error is plotted rather than LP-objective value. Notice that in this example, subtask VFs provide a better basis than either SING or PAIR even before splitting. Computation times for each iteration of the decision tree algorithm vary with the scoring metric. Averaged over the first 6 splits (7 leaves), we have (in CPU seconds) the following times: FLP – 691s; FWLP – 388s; Dual – 935s.[7] We note that the dual times are based on an unoptimized implementation and cannot be meaningfully compared to the others (but we include it for completeness).

Figure 5 shows LP-objective value for a 1-resource, 20-task problem for both DUAL and PAIR. The error for SING is not plotted as it is about 5 times as high as for PAIR. Finally, a similar plot is shown in Figure 6 for a 2-resource, 10-task problem (again SING is not shown). In the former, the dual metric offers an improved solution after only two splits, while in the latter, the subtask VFs themselves provide

[6] The results for FLP are shown only up to five leaves in this and the subsequent graph.

[7] The implementation is in Matlab; calls to optimized C++ routines are used for FLP and FWLP, but not for dual. We project the same optimization applied to dual would yield 10-fold speed up. Experiments were run on a 700MHz PCs running Linux.

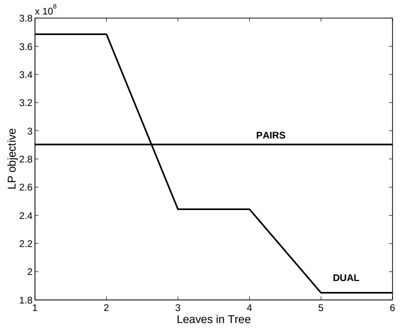

Figure 5: LP-objective value for 1 resource, 20 tasks.

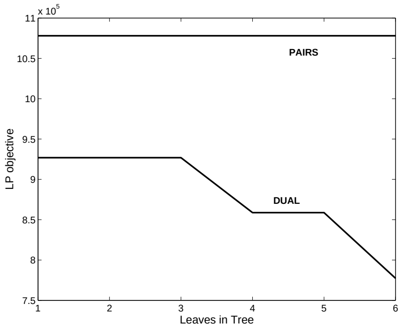

Figure 6: LP-objective value for 2 resources, 10 tasks.

a better solution than the pairs. In the latter case, an improved solution is found after 3 splits. Computation times for the 20-task splits average 1784s, whereas the 10-task splits average 1801s. Again we emphasize that the dual running times are based on an unoptimized implementation.

While these results should be viewed as preliminary, they are encouraging. The evidence suggests that subtask VFs provide very good basis sets for factored linear approximators. Furthermore, we see that with very few splits, the PWL approach can offer further improvement over a purely linear approximator.

6 Concluding Remarks

While linear approximators have proven to be valuable tool in the solution of factored MDPs, to date, no concrete proposals have been put forth for basis function selection and revision in the factored setting. We a described a concrete means for basis function selection using subtask VFs, and suggested a family of techniques for basis function revision. We investigated in some depth the use of decision tree techniques to produce PWL approximators. Our empirical results show that even with few splits, decision quality can be greatly improved relative to standard linear approximators.

Future research directions include the development of *hybrid* basis revision techniques, where new functions can be added directly to the basis, along with splitting of state space. Further experimentation is also needed to determine the range of problems on which this approach works well. Finally, we plan to investigate splitting criteria that tradeoff computational cost for projected improvement in decision quality.

Acknowledgements

Thanks to the referees for their comments. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

References

[1] D. P. Bertsekas and J.. N. Tsitsiklis. *Neuro-dynamic Programming*. Athena, Belmont, MA, 1996.

[2] C. Boutilier, R. I. Brafman, and C. Geib. Prioritized goal decomposition of Markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *Proc. Fifteenth International Joint Conf. on AI*, pp.1156–1162, Nagoya, 1997.

[3] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. Fourteenth International Joint Conf. on AI*, pp.1104–1111, Montreal, 1995.

[4] D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proc. Twelfth International Joint Conf. on AI*, pp.726–731, Sydney, 1991.

[5] T. Dean, R. Givan, and S. Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proc. Thirteenth Conf. on Uncertainty in AI*, pp.124–131, Providence, RI, 1997.

[6] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comput. Intel.*, 5(3):142–150, 1989.

[7] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artif. Intel.*, 89:219–283, 1997.

[8] C. Guestrin, D. Koller, and R. Parr. Max-norm projections for factored MDPs. In *Proc. Seventeenth International Joint Conf. on AI*, pp.673–680, Seattle, 2001.

[9] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Info. Processing Sys. 14 (NIPS-2001)*, Vancouver, 2001.

[10] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proc. Fifteenth Conf. on Uncertainty in AI*, pp.279–288, Stockholm, 1999.

[11] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[12] N. Meuleau, M. Hauskrecht, K. Kim, L. Peshkin, L. P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled Markov decision processes. In *Proc. Fifteenth National Conf. on AI*, pp.165–172, Madison, WI, 1998.

[13] A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Mach. Learn.*, 21:199–234, 1995.

[14] R. Patrascu, P. Poupart, D. Schuurmans, C. Boutilier and C. Guestrin. Greedy linear value-approximation for factored Markov decision processes. In *Proc. Eighteenth National Conf. on AI*, Edmonton, 2002. to appear.

[15] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[16] D. Schuurmans and R. Patrascu. Direct value approximation for factored MDPs. In *Advances in Neural Info. Processing Sys. 14 (NIPS-2001)*, Vancouver, 2001.

[17] S. P. Singh and D. Cohn. How to dynamically merge Markov decision processes. In *Advances in Neural Info. Processing Sys. 10*, pp.1057–1063. MIT Press, Cambridge, 1998.

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