Approximating Value Trees in Structured Dynamic Programming

Craig Boutilier and Richard Dearden
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, CANADA
cebly@cs.ubc.ca, dearden@cs.ubc.ca

Abstract

We propose and examine a method of approximate dynamic programming for Markov decision processes based on structured problem representations. We assume an MDP is represented using a dynamic Bayesian network, and construct value functions using decision trees as our function representation. The size of the representation is kept within acceptable limits by pruning these value trees so that leaves represent *possible ranges of values*, thus approximating the value functions produced during optimization. We propose a method for detecting convergence, prove errors bounds on the resulting approximately optimal value functions and policies, and describe some preliminary experimental results.

1 Introduction

Markov decision processes (MDPs) have come to play an increasingly important role in AI research, forming the basic model for much recent research in decision-theoretic planning (DTP) and reinforcement learning (RL). The aim in both DTP and RL is to discover a *policy* for the behavior of an agent in a (generally) stochastic environment. The resulting policy should offer good or optimal long-term performance in the sense of maximizing expected accumulation of reward. The key distinction between DTP and RL is that the former assumes an *immediate reward function* and *action model* representing the system dynamics are known, whereas the latter takes both of these to be unknown quantities that must be learned (possibly implicitly).

With a known action model and rewards, optimization methods based on dynamic programming can be used to produce an optimal policy [1, 13, 20]. But a serious problem for dynamic programming is the curse of dimensionality: the time (and space) required grows polynomially with the size of the state space, which itself grows exponentially with the number domain features. This problem is exacerbated in RL because of the sampling requirements for each state.

One way of addressing this problem in the case of both known and unknown models is through the use of *aggregation* methods (or generalization), in which a number of states are grouped because they have similar or identical values and/or action choice. These aggregates are treated as a single state in dynamic programming algorithms for the solution of MDPs or the related methods used in RL [22, 2, 16, 4, 5, 11, 12, 9, 17]. Such aggregations can be based on a number of different problem features, such as similarity of states according to some domain metric; but most methods generally assume that the states so grouped have the same value. In addition, such schemes can be exact or approximate, adaptive or fixed, and uniform or nonuniform, and can be generated using *a priori* problem characteristics or learned generalizations.

In this paper, we consider the problem of constructing an approximately optimal policy when the action-model and reward function are known.[1] In addition, we assume that the action model is specified using a compact and natural specification language, namely *dynamic Bayesian networks* [18, 10]. In previous work, we described a method for optimal policy construction that exploited the problem structure laid bare by the Bayes net representation [5]. Our algorithm built aggregations in a nonuniform and adaptive way, representing value functions (and policies) using decision trees, and performed structured dynamic programming using this representation.

Unfortunately, with many problems, even structured representations may not help greatly with optimal policy construction, for the optimal value function may take on a large number of distinct values, precluding compact rep-

[1] The close relationship between RL methods such as *Q-learning* [26] and the solution of MDPs with known models suggests that our ideas should be applicable to the unknown-model setting (see Section 6).

resentation. However, often the distinctions made are of minor importance—if states with roughly the same value can be grouped, good (though possibly suboptimal) policies should result. The approximation schemes we present in this paper consider *pruning* the tree representation of value functions at intermediate stages of policy construction. This method thus exploits prior problem structure in a way that leads to very informed approximation.

In Section 2, we describe MDPs and their structured representation using dynamic Bayes nets, followed in Section 3 by a brief description of the SPI algorithm of [5] that performs optimization using a decision tree representation of value functions. We then focus on issues arising due to approximation of these value trees. We first describe, in Section 4, an algorithm for pruning (and ordering) a single value tree, using methods adapted from those in the literature on classification by decision trees [3, 25]. In Section 5, we describe a structured version of value iteration that approximates the n-step optimal value functions it produces using the pruning method. These approximate value trees are labeled with value *ranges* that are guaranteed to contain the true values of the states to which they refer. This allows local error bounds to be maintained during computation with minimal effort. These will typically be much tighter than possible global bounds. Moreover, while approximation of value functions can sometimes lead to arbitrarily bad results [8], maintaining accurate value ranges allows us to circumvent convergence problems. We show convergence, describe error bounds, and report on some preliminary experimental results. We conclude with a discussion of the applicability of these ideas to reinforcement learning.

2 MDPs and Structured Representations

We assume that the system to be controlled can be described as a fully-observable, discrete state *Markov decision process* [1, 13, 19], with a finite set of system states S. The controlling agent has available a finite set of actions A which cause stochastic state transitions: we write $Pr(s, a, t)$ to denote the probability action a causes a transition to state t when executed in state s. A real-valued reward function R reflects the objectives of the agent, with $R(s)$ denoting the (immediate) utility of being in state s.[2] A (stationary) *policy* $\pi : S \rightarrow A$ denotes a particular course of action to be adopted by an agent, with $\pi(s)$ being the action to be executed whenever the agent finds itself in state s. We assume an infinite horizon (i.e., the agent will act indefinitely) and that the agent accumulates the rewards associated with the states it enters.

In order to compare policies, we adopt *expected total discounted reward* as our optimality criterion; future rewards are discounted by rate $0 \leq \beta < 1$. The value of a policy π can be shown to satisfy [13]:

$$V_\pi(s) = R(s) + \beta \sum_{t \in S} Pr(s, \pi(s), t) \cdot V_\pi(t)$$

The value of π at any initial state s can be computed by solving this system of linear equations. A policy π is *optimal* if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and policies π'. The *optimal value function* V^* is the same as the value function for any optimal policy. Techniques for constructing optimal policies and value functions for discounted problems have been well-studied; we discuss these in the next section.

One difficulty with the general presentation of MDPs given above is its failure to exploit natural problem structure. Most systems are characterized by a set of random variables or propositions that describe relevant features, and actions and rewards are specified in terms of these features [15, 4, 24]. In addition, since the state space grows exponentially with the number of features, explicit specification and computation over the state space can be problematic.

We assume that a set of atomic propositions $\mathbf{P}$ describes our system, inducing a state space of size $2^{|\mathbf{P}|}$, and use two-stage temporal or *dynamic Bayesian networks* to describe our actions [10, 5]. For each action, we have a Bayes net with one set of nodes representing the system state prior to the action (one node for each variable), another set representing the world after the action has been performed, and directed arcs representing causal influences between the these sets. Each post-action node has an associated *conditional probability table* (CPT) quantifying the influence of the action on the corresponding variable, given the value of its influences (see [5, 6] for a more detailed discussion of this representation).[3] Figure 1(a) illustrates this representation for a single action.[4]

The lack of an arc from a pre-action variable X to a post-action variable Y in the network for action a reflects the independence of a's effect on Y from the prior value of X. We capture additional independence by assuming structured CPTs. In particular, we use a *decision tree* to represent the function that maps combinations of parent variable val-

[2] More general formulations of reward (e.g., adding action costs) offer no special complications.

[3] To simplify the presentation, we consider only binary variables and assume that no arcs are directed between post-action nodes; but these assumptions can easily be relaxed.

[4] This is a toy domain in which a robot is supposed to get coffee from a coffee shop across the street, can get wet if it is raining unless it has an umbrella, and is rewarded if it brings coffee when the user requests it, and penalized (to a lesser extent) if it gets wet [5, 7]. This network describes the action of fetching coffee.

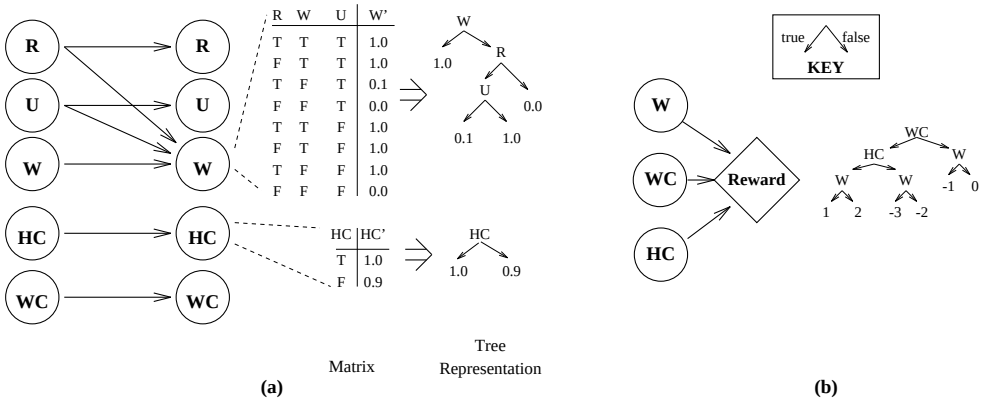

Figure 1: (a) Action Network with Tree-structured CPTs; and (b) Reward Tree

ues to (conditional) probabilities. For instance, the trees in Figure 1(a) show that U influences the probability of W becoming true (as a consequence of the action), but only if R is true and W is false (left arrows are assumed to be labeled "true" and right arrows "false"). Thus, additional regularities in transition probabilities are used to provide a more compact representation than the usual (locally exponential) CPTs (the matrices). This can be exploited computationally, as we describe below. A similar representation can be used to represent the reward function R, as shown in Figure 1(b). We call this the (immediate) reward tree, *Tree*(R).

3 Structured Policy Construction

A very simple algorithm for optimal policy construction is *value iteration* [1]. We produce a sequence of n*-step optimal value functions* V^n by setting $V^0 = R$, and defining

$$V^{i+1}(s) = \max_{a \in \mathcal{A}} \{ R(s) + \beta \sum_{t \in S} \mathit{Pr}(s, a, t) \cdot V^i(t) \} \quad (1)$$

The sequence of functions V^i converges linearly to V^* in the limit. Each iteration is known as a *Bellman backup*. After some finite number n of iterations, the choice of maximizing action for each s forms an optimal policy π and V^n approximates its value. In particular, one simple stopping criterion requires termination when

$$\|V^{i+1} - V^i\| \leq \frac{\varepsilon(1-\beta)}{2\beta} \quad (2)$$

(where $\|X\| = \max\{|x| : x \in X\}$ denotes the supremum norm). This ensures the resulting value function V^{i+1} is within $\frac{\varepsilon}{2}$ of the optimal function V^* at any state, and that the induced policy is ε-optimal (i.e., its value is within ε of V^*) [19].

In an effort to mitigate the curse of dimensionality, researchers have sought to use aggregation or generalization to group states. One possible approach uses action models to form regions in the state space that have identical value and performs dynamic programming steps in this way [5, 12]. We briefly describe a structured version of value iteration (SVI) based on this intuition: at each stage, V^i will be represented as a decision tree.[5]

In value iteration, we need to produce the sequence of value functions $V^0, V^1, \cdots$ using Bellman backups, and we'd like to do so using a compact function representation such as decision trees. Clearly, *Tree*(R) provides a structured representation of V^0. In addition, given any such structured value tree V, we can use the Bayes net action description to produce a Q*-tree* Q_a for any action a. This tree describes the value of performing action a assuming terminal value is given by V (i.e., a Q-function [26]). Roughly, each branch of V determines a region of the state space with a unique value. The Bayes net for a allows us to easily determine *the conditions that influence the probability of reaching any such region* when a is performed: we simply read from the network the variables that influence the variables in V. Intuitively, we perform a stochastic generalization of goal regression [12]. Rather than provide details, we illustrate the intuitions with a simple example (see [5] for details).

Consider the initial value tree V in Figure 2 (again, left arrows denote true, right arrows false) and suppose action a (Figure 1(a)) is to be performed. We can determine the *future value* of a as follows: to determine whether we end up in the left or right subtree of V, we must know the probability of a making *WC* true. The conditions (prior to the ac-

[5] This algorithm is a minor variant of the SPI algorithm [5], which is based on *modified policy iteration* [20]. The basic operations are the same, though intermediate policies are not produced in SVI.

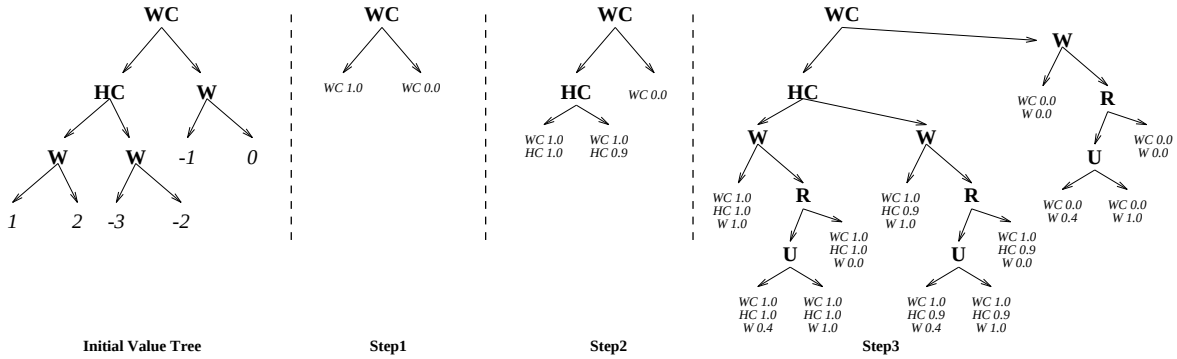

Figure 2: Generating a Q-tree

tion) that influence *WC* becoming true (after the action) can be read from the network for a, giving rise to the first partial tree (Step 1). Note that the probability of *WC* becoming true (i.e., the actual effect of the action on *WC*) labels each leaf in this partial tree. We emphasize that the *WC* occurrences labeling interior nodes of the tree refer to the pre-action state, while leaves refer to the probability of *WC* in the post-action state.

We perform a similar "explanation" of *HC* (Step 2). Note that we only care about its value when *WC* is possibly true; thus the influences for *HC* are only added to the left branch of the first partial tree. Finally, the (more interesting) conditions under which W becomes true or false are also read from the network and added to the partial tree (Step 3). At this point, the leaves of this partial tree are labeled with probabilities that determine the precise probability of ending up in any of the regions determined by the initial value tree. By computing this expected future value (together with discounting and adding the immediate reward), we determine the tree Q_a shown in Figure 3(a).

Given a set of Q-trees (one for each action) produced using a value function V^i, we can construct a tree representing V^{i+1} by simply "merging" these trees; that is, we create a minimal *subsuming tree* (one that makes all distinctions common to the set) and choose the maximum Q-value for each new region. This corresponds to performing a Bellman backup according to Equation (1). Once a good value function V^n is obtained, the set of Q-trees with respect to V^n can be used to produce a structured policy, by merging and labeling with maximizing actions.

4 Approximate Value Trees

The most important feature of SVI is that it produces a sequence of value trees that accurately represent the optimal n-step value functions, and produces the smallest trees possible based solely on structure (modulo variable ordering). Unfortunately, it may be *inherently* difficult to construct an optimal value function and policy for certain problems because they fail to exhibit enough structure to admit compact representation.[6] Certainly, there is a very clear tendency for the sequence of value trees produced in SVI to make progressively more fine-grained distinctions, some of which may have a marginal effect on value.

We now consider strategies that remove distinctions (nodes) in the tree that induce *small* differences in value. The resulting *pruned* value tree will no longer reflect regions that have identical value, but regions of similar value. Our basic policy construction scheme will be an approximate version of SVI (called ASVI). In broad outline, we will construct a sequence of *approximate ranged value trees* by: a) pruning a value tree so that it makes fewer distinctions and approximates its true value; b) generating a new value tree by structured Bellman backup based on the approximate value tree. Essentially, we will perform region-based dynamic programming, but coalesce regions that make distinctions of marginal utility. We describe the ASVI algorithm in the next section. We first describe ranged value trees and strategies for pruning a single tree.

Suppose we are given a value tree such as that in Figure 3(a), but are unhappy with its size. A simple way to reduce the size of the tree is to replace a (nontrivial) subtree with a single leaf, for example, as shown in Figure 3(b). Since we no longer distinguishing (e.g.) WR-states from $W\overline{R}$-states, the resulting tree can only approximately represent the true value function. One obvious choice of value assignment for the larger region (new leaf) is the midpoint of the values being replaced—this minimizes the maximum error in the approximate value tree. We could instead label

[6] For instance, one can easily construct examples of small (polynomial) Bayes net descriptions of MDPs that have value functions with many (exponential) distinct values [5].

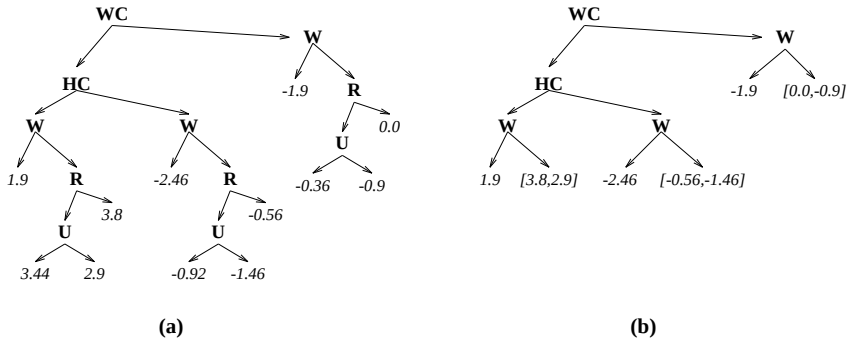

Figure 3: (a) Value Tree and (b) Pruned Range Tree

the new region with a *range* encompassing all replaced values (as shown in the figure). Ranges play a valuable role in ASVI, so we assume that all approximate value functions are represented by *ranged value trees* (r-trees): each leaf is labeled with a range $[u, l]$ representing the maximum (upper) and minimum (lower) values associated with states in the corresponding region. Point-valued regions (hence, exact value functions) are represented by setting $u = l$.

For any ranged value function V, we take the *upper value function* $V^{\uparrow}$ to be the value function induced by considering the upper entries of V. The *lower value function* $V^{\downarrow}$ and the *midpoint function* $V^{\leftrightarrow}$ are defined in the obvious way. In choosing a particular value for a region given V (e.g., in action selection), one can obviously recover the midpoint from the range and use the function $V^{\leftrightarrow}$ as needed. For any state s and ranged function V, we define *span*(s) to be $u - l$, where u and l are the upper and lower values for the region containing s. The span of V is the maximum of all such spans. The maximum error in the induced value function $V^{\leftrightarrow}$, assuming that the ranges in V contain the true values of all states, is *span*$(V)/2$.

When pruning an r-tree, we may either want the most accurate tree of a fixed (maximum) size, or the smallest tree of a fixed (minimum) accuracy. This problem, of course, is strongly related to work on pruning decision trees in classification. Given a fixed decision tree (assuming training has been completed), Bohanec and Bratko [3] present an algorithm for producing the sequence of pruned trees of decreasing size such that each tree in the sequence is the most accurate among all trees of that size. We can apply similar ideas in our setting, where the aim is to produce ranged value functions with the smallest span. The algorithm is shown in Figure 4. Several points are worth noting. We assume that the initial tree is ranged, and produce the new r-trees with (possibly) larger ranges by collapsing subtrees. The sequence of trees is implicit in the variable SEQ-LABEL: subtrees are replaced in the order described by SEQ-LABEL. If variables are boolean, the sequence of produced trees is dense (there is a tree for each size less than the initial size of the tree). Finally, in practice, the algorithm is not run to completion. Instead, we terminate when either: a) the r-tree at some point has a range larger than some maximum specified range δ, in which case the *previous* pruned tree is the desired tree; or b) the r-tree has been reduced to some maximum allowable size. Which of these choices is used will be application dependent.

The amount of pruning that one can do by removing subtrees, within acceptable tolerances—indeed the size of the tree before pruning—may be strongly influenced by the node ordering used in the value tree. Again, this issue arises in research on classification [21, 25]. Finding the smallest decision tree representing a given function is NP-hard [14], but there are feasible heuristics one can use in our setting to reorder the tree to make it smaller and/or more amenable to pruning. Among these, one appears rather promising and is strongly related to the information gain heuristic [21].[7]

5 Policy Construction with Approximate Value Functions

5.1 The ASVI Algorithm and its Properties

Armed with a method for pruning an r-tree, we now examine how this can be applied to a policy construction technique like SVI. Our basic strategy can be described as follows. We use the reward *Tree*(R) as the ranged function $\overline{V}^0$ (the ranges initially will be point values). Given any

[7] Roughly, we take the existing tree and categorize each variable according to the size of the ranges induced when it is either true or false—this can be done in one sweep through the tree. The variable with the smallest ranges is installed at the root of the tree (e.g., see [25]). This is repeated with the variables remaining in the new subtrees. We defer details to the full paper. (Thanks to Will Evans for his help with these ideas.)

Input: ranged value tree T
Output: Labels SEQ-LABEL indicating order in which to replace subtrees rooted at labeled node

1. Let $SEQ = 1$
2. Let F be the set of penultimate nodes in T (non-leaf nodes *all* of whose children are leaves)
3. For each $n \in F$, set R-label$(n) = [u_n, l_n]$ where $u_n = \max\{u : [u, l] \text{ labels a child of } n\}$, and $l_n = \min\{l : [u, l] \text{ labels a child of } n\}$, and
4. While $F \neq \emptyset$
   (a) Let $n = \arg\min\{u_n - l_n : n \in F\}$
   (b) Set $SEQ\text{-}LABEL(n) = SEQ$; $SEQ = SEQ + 1$
   (c) Set $F = F - \{n\}$
   (d) If $m =$ *Parent*(n) exists and *Children*$(m) \cap F = \emptyset$ then add m to F and set R-label$(m) = [u_m, l_m]$ where $u_m = \max\{u_c : [u_c, l_c] \text{ is R-label of a child } c\}$ $l_m = \min\{l_c : [u_c, l_c] \text{ is R-label of a child } c\}$

Figure 4: Algorithm for Optimal Sequence of Pruned Ranged Value Trees

1. Set $\overline{V}^0 = R$; set $i = 0$
2. Prune $\overline{V}^0$, under some pruning criterion, to produce $\widetilde{V}^0$
3. Repeat until stopping criterion holds (w.r.t. $\widetilde{V}^i$ and $\widetilde{V}^{i+1}$)
   (a) Construct ranged Q-trees $\overline{Q}_a^{i+1}$ for each action a using $\widetilde{V}^i$ as the terminal value function
   (b) Merge the trees $\overline{Q}_a^{i+1}$ to produce ranged value function $\overline{V}^{i+1}$
   (c) Prune $\overline{V}^{i+1}$, under some pruning criterion, to produce $\widetilde{V}^{i+1}$; Increment i

Figure 5: General Structure of ASVI

ranged function $\overline{V}^i$, we create a pruned tree $\widetilde{V}^i$ by pruning $\overline{V}^i$ within some specified tolerance (or size) to get a more compact, but approximate, representation of $\overline{V}^i$. The pruned tree $\widetilde{V}^i$ is then used as the basis for Bellman backups to produce a new r-tree $\overline{V}^{i+1}$. This new r-tree is constructed in a manner very similar to that used in ordinary SVI, the key difference lying in the use of the ranges in $\widetilde{V}^i$ instead of point values. We note that $\overline{V}^{i+1}$ is itself an approximation of the true $i+1$-step value function V^{i+1}, since it was produced using an approximation of V^i. However, V^{i+1} will be further approximated by pruning $\overline{V}^{i+1}$ to produce $\widetilde{V}^{i+1}$.

The approximate SVI algorithm (ASVI) is described in general terms in Figure 5. The general structure shows the production of a sequence of (ranged) approximations $\overline{V}^n$, $\widetilde{V}^n$ of the optimal n-step value functions V^n. The function $\overline{V}^n$ is produced by structured Bellman backup and $\widetilde{V}^n$ by additional pruning. We elaborate on the crucial steps in the algorithm below.

The steps involved in producing the value function $\overline{V}^{i+1}$ (i.e., Steps 3(a) and 3(b)) are reasonably straightforward, but deserve some elaboration. The production of the r-tree $\overline{Q}_a^{i+1}$ proceeds exactly as it does in SVI with one minor exception. Since the target value tree $\widetilde{V}^i$ is labeled with ranges, it is a simple matter to produce ranges for the leaves of $\overline{Q}_a^{i+1}$: we simply take the expected future value using the upper values for the target regions to produce the upper value for a leaf in the new tree and lower values similarly. Conceptually, we can treat the new Q-tree as having ranges produced using the trees $\widetilde{V}^{i\uparrow}$ and $\widetilde{V}^{i\downarrow}$.

Slightly more subtle is the merging of Q-trees in Step 3(b). Merging requires that for each state we determine which action choice maximizes future expected value. In SVI this is reasonably straightforward: we find a partition (tree) that subsumes each Q-tree and label the leaves of this larger tree with the maximum value from the corresponding partitions in the set of Q-trees. In ASVI, these partitions are labeled with ranges that can't necessarily be compared with a max operator. Instead we label the leaves of $\overline{V}^{i+1}$ with the *maximum* of all upper labels of the corresponding partitions in the Q-trees, and the *maximum* of all lower labels of the corresponding partitions. Clearly, choosing the maximum of the upper labels is correct and bounds the true value of a state s. In the case of the lower labels, there exists an action that guarantees state s has the maximum of the expected values among the lower labels, namely the action used to derive the maximizing Q-tree. This is therefore a tight lower bound on the true value of state s.[8]

The termination of ASVI raises some interesting issues. Exact value iteration is guaranteed to converge because the transformation operation (the Bellman backup) on value functions is a contraction operator with respect to the supremum norm (see Equation (2)). The same does not apply when the intermediate value functions are approximated. Indeed, without a well-thought out stopping criterion, we can construct (quite straightforward and natural) examples in which the pruning of value trees causes ASVI to cycle through a sequence of identical value functions without termination.[9] To deal with this situation, we adopt a fairly conservative approach: we stop whenever the ranges of two consecutive value functions indicate that the criterion given

[8] Note that this argument relies crucially on the fact that we need not *pick an action* at this point; there will generally be no single action that one can assign to each state in the region to ensure this maximum lower bound is achieved for all states. But this is irrelevant to the construction of the value function.

[9] For further discussion of convergence problems that arise due to approximation, see [8].

Table 1: Results in the 400 state domain for both fixed and sliding tolerance pruning.

Fixed Tolerance				Sliding Tolerance			
Pruning	Iterations	Time (s)	Max. Error	Pruning	Iterations	Time (s)	Max. Error
0	20	761	0				
1	13	156	0.105	5%	11	100	0.708
2	12	129	0.478	10%	11	68	2.190
3	11	88	0.707	15%	11	59	4.596
5	11	70	1.037	20%	10	43	6.012
7	8	46	2.189	30%	8	28	18.295
9	2	1	42.47	40%	6	14	21.436
10	2	1	74.135	50%	5	8	33.019

by Equation (2) *might* be satisfied. Specifically, the use of encompassing ranges allows us to test this condition in a way that is impossible with simple point valued approximation. For any two ranged value functions V, W, we define

$$(V \overset{\circ}{-} W)(s) = \min\{|r - r'| : \; V^{\downarrow}(s) \leq r \leq V^{\uparrow}(s), \; W^{\downarrow}(s) \leq r \leq W^{\uparrow}(s)\}$$

We terminate ASVI when the following condition holds:

$$\|V^{i+1} \overset{\circ}{-} V^{i}\| \leq \varepsilon \tag{3}$$

In other words, when the ranges for every state in successive value approximations either overlap or lie within ε of one another, we terminate. We note that testing this condition with two r-trees is quite simple, involving only the construction of a (minimal) subsuming tree.

Regardless of the pruning criterion, as long as it produces sound r-trees, we can show the following results.

Prop. 1 *Let $\widetilde{V}^i$ be the ith value tree produced by ASVI. Then $\widetilde{V}^{i\downarrow} \leq V^i \leq \widetilde{V}^{i\uparrow}$.*

Thus the i-step ranged value functions "contain" the optimal i-step value functions for the MDP. For finite-horizon problems, this is an important characteristic. It allows one to specify different error bounds for different regions of the state space—if the value function is accurate in certain regions of the state space, this knowledge is not "washed out" in global error by larger error in other regions. With respect to infinite horizon policies, Proposition 1 guarantees termination:

Prop. 2 *If the stopping criterion specified by Equation (3) is used, ASVI is guaranteed to terminate. Its rate of convergence is linear (at least that of value iteration).*

Finally, it is easy to verify the following error bounds.

Prop. 3 *Let ASVI terminate (according to Equation (3)) with ranged value function $V = \widetilde{V}^i$; and let $\delta = span(V)$. Then*

$$\|V^* - V\| \leq \frac{\beta(2\delta + \varepsilon)}{1 - \beta}$$

The induced policy π is such that

$$\|V^* - V_\pi\| \leq \frac{2\beta(2\delta + \varepsilon)}{1 - \beta}$$

We note that the methods of SPI, as described in Section 3, can be applied to produce a tree-structured policy using the midpoint value function $\widetilde{V}^{n\leftrightarrow}$.[10]

It is also worth noting that the argument for convergence of ASVI cannot be applied to ASPI (approximate structured modified policy iteration). ASPI requires that intermediate policies be produced (and partially evaluated); but because ranges are used one cannot generally guarantee that the sequence of policies is *improving*. One action may be better in one part of a region but worse in another. While ASPI works well on many examples, it can rather easily fall into cyclic behavior. Thus, value iteration seems the ideal candidate for approximation using ranges. However, we are currently investigating more refined applications of these ideas to policy iteration-based algorithms.

5.2 Practical Considerations and Results

In this section, we consider some of the more pragmatic issues associated with putting ASVI into practice. We first consider pruning strategies. Suppose we have a desired "percentage" tolerance t for error (e.g., all approximate values should lie within $t = 0.1$, or 10%, of true value). There are two ways to implement such a tolerance: a) a *fixed tolerance* set at

$$t\frac{\beta}{1 - \beta}|R^{\max} - R^{\min}|$$

[10] See [23] for discussion of policy error given an approximate value function.

or b) a *sliding tolerance*, where the tree for the n-stage to go function V^n is pruned using a tolerance of

$$t \sum_{i=0}^{n} \beta^i |R^{\max} - R^{\min}|$$

(Here $R^{\max}$ and $R^{\min}$ are the maximum and minimum immediate rewards; hence these terms reflect the largest range in values obtainable over a finite or infinite horizon.) A sliding tolerance is sometimes useful since the magnitude of value functions V^n tends to grow with n—at early stages a fixed tolerance may prune too aggressively, especially if we are interested in producing reasonably accurate value functions at all stages. However, both approaches give good results, for the fixed tolerance scheme will stop pruning smaller distinctions eventually. Experiments suggest that fixed tolerance runs faster, produces less complicated trees at convergence, and is only slightly less accurate that sliding tolerance.

Related to this is the fact that aggressive pruning often removes small distinctions present in *Tree*(R), which are reintroduced in the next stage, which are pruned again, etc. We are currently exploring methods that prune *Tree*(R) initially, and only introduce those distinctions after a sufficient number of iterations. Finally, the error bounds for SVI are extremely conservative and are due primarily to the need to detect convergence. Most of our experiments do not come close to achieving such poor results. However, we are exploring methods for running several more iterations of "fine-tuning" after convergence, focusing on specific parts of the state space, and methods to detect whether additional progress is being made.[11]

Tables 1 and 2 display the results of pruning in two different domains. In both cases, *Iterations* is the number of iterations of SVI required for convergence, and *Max. Error* is the maximum (over all states) of the difference between the optimal value of a state and its actual value under the (approximately optimal) policy returned by ASVI. Table 2 also contains the average error over all states, and the number of leaves in the ranged value tree produced. Note that there is no pruning of *Tree*(R) or fine-tuning of the reward tree in either domain, but we are currently exploring such techniques.

The first domain (Table 1) is a more complex version of the coffee-robot domain described earlier. It contains eight actions and six variables (two of which are five-valued and the rest boolean) giving 400 states. Values of states in this domain range from -90 to zero. Even relatively small amounts of pruning, resulting in errors of less than two percent, result in an extremely large reduction in computation time.[12] As expected, the actual errors are much smaller than the theoretical maximum except when pruning ranges become very large and result in trivial policies.

Table 2: Results in an exponential domain for fixed maximum error pruning.

Pruning	Iter	Time(s)	MaxErr	AvgErr	Tree size
0	44	3153	0	0	1024
0.25	42	63	1.38	0.047	88
0.5	33	31	2.12	0.058	80
1.0	14	6	1.91	0.068	50
1.5	3	0.3	2.13	0.077	10
2.0	1	0.1	8.10	0.567	1

Table 1 also compares the effects of fixed and sliding tolerance pruning. Fixed tolerance pruning is somewhat faster than sliding tolerance when compared on runs with similar error; this is due to the much smaller trees produced early on in ASVI. These small initial trees greatly reduce the variety of possible value functions that can be produced, so that, for example, any pruning range between six and eight will produce the same value tree for this domain. In contrast, sliding tolerance pruning results in much more gradual changes in value as the tolerance t increases, making it easier to select good pruning tolerances.

The second domain (Table 2) is one in which every state has a unique value, leading to worst-case behavior for structured methods such as SVI. While the problem has a compact input description, SVI must (ultimately) perform backups for each state, with the additional overhead of tree construction. The problem consists of ten boolean variable and ten actions, with 1024 states having values from 0 to 10. This domain demonstrates the value of pruning in ASVI in preventing the exponential blowup which leads to the very poor performance of SVI. It also shows that even when the same number of iterations are necessary for convergence, the reduced tree sizes lead to large performance improvements. This domain is an example of one where variable ordering is critical. Had we chosen a poor ordering, it is likely that very little pruning would have been possible, resulting in little or no savings.[13]

[11] Other measures such as the span seminorm look promising in this respect.

[12] The time without pruning was produced using SVI and will therefore appear quite slow. Better exact algorithms such as SPI (i.e., structured modified policy iteration) can also be used. The pruning times still compare very favorably with the 434s for finding the optimal value using SPI.

[13] We have not yet experimented with variable reordering; ASVI uses orderings implicit in the problem representation, which tend to be natural and compact. However, examples like this point to the need for further exploration in reordering before pruning.

6 Concluding Remarks

There are a number of directions that remain to be explored, including additional experiments with the strategies suggested in the previous section. We are also interested in approximation methods based on algorithms other than value iteration. Finally, in the full paper we describe the application of our pruning method to RL. Roughly, following Dietterich and Flann [12], we can use action descriptions to perform (a stochastic, non-goal-based) generalization of goal-regression along explored trajectories. Pruning produces generalizations of the state space that correspond to regions with approximately the same (currently estimated) value.

Acknowledgements: Thanks to Will Evans, Simon Kasif and Ron Kohavi for their discussions and pointers to relevant literature. This research was supported by NSERC Grant OGP0121843 and a UBC Graduate Fellowship.

References

[1] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[2] D. P. Bertsekas and D. A. Castanon. Adaptive aggregation for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34:589–598, 1989.

[3] Marko Bohanic and Ivan Bratko. Trading accuracy for simplicty in decision trees. *Machine Learning*, 15:223–250, 1994.

[4] Craig Boutilier and Richard Dearden. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1016–1022, Seattle, 1994.

[5] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, Montreal, 1995.

[6] Craig Boutilier and Moisés Goldszmidt. The frame problem and bayesian network action representations. In *Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence*, Toronto, 1996. to appear.

[7] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996. to appear.

[8] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*. MIT Press, Cambridge, 1995.

[9] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 726–731, Sydney, 1991.

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

[11] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1121–1127, Montreal, 1995.

[12] Thomas G. Dietterich and Nicholas S. Flann. Explanation-based learning and reinforcement learning: A unified approach. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 176–184, Lake Tahoe, 1995.

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

[14] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5:15–17, 1976.

[15] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1073–1078, Seattle, 1994.

[16] Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Proceedings of the Eighth International Conference on Machine Learning*, pages 333–337, Evanston, IL, 1991.

[17] Andrew W. Moore and Christopher G. Atkeson. The partigame algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*, 1995. To appear.

[18] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, 1988.

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

[20] Martin L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127–1137, 1978.

[21] J. Ross Quinlan. *C45: Programs for Machince Learning*. Morgan Kaufmann, San Mateo, 1993.

[22] Paul L. Schweitzer, Martin L. Puterman, and Kyle W. Kindle. Iterative aggregation-disaggregation procedures for discounted semi-Markov reward processes. *Operations Research*, 33:589–605, 1985.

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

[24] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):365–379, 1990.

[25] Paul E. Utgoff. Decision tree induction based on efficient tree restructuring. Technical Report 95–18, University of Massachusetts, March 1995.

[26] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.