Correlated Action Effects in Decision Theoretic Regression

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

Abstract

Much recent research in decision theoretic planning has adopted Markov decision processes as the model of choice, and has attempted to make their solution more tractable by exploiting problem structure. *Structured policy construction* algorithms achieve this by a decision theoretic analogue of *goal regression*, using action descriptions based on Bayesian networks with tree-structured conditional probability tables. At present, these algorithms are unable to deal with actions with correlated effects. We describe a new decision theoretic regression operator that corrects this weakness.

Introduction

Recent research in the use of Markov Decision Problems (MDPs) for decision-theoretic planning (DTP) has focussed on solution methods that avoid explicit enumeration of the state space while constructing (approximately) optimal policies. Such techniques include the use of *reachability analysis* to eliminate unreachable states (Dean *et al.* 1993; Barto, Bradtke, & Singh 1995), and *state aggregation*, whereby various states are grouped together and treated as a single state. Recently, methods for automatic aggregation have been developed in which certain problem features are ignored, making certain states indistinguishable (Chapman & Kaelbling 1991; Boutilier & Dearden 1994; Dearden & Boutilier 1997; Boutilier, Dearden, & Goldszmidt 1995; Tsitsiklis & Roy 1996).

In some of these aggregation techniques, the use of standard AI representations such as STRIPS or Bayesian networks to represent actions in an MDP can be exploited to help construct the aggregations. In particular, they can be used to help identify which variables are *relevant* to the determination of value or to the choice of action. This connection has lead to the insight that the basic operations in computing optimal policies for MDPs can be viewed as a generalization of *goal regression* (Boutilier, Dearden, & Goldszmidt 1995). A *Bellman backup* (Bellman 1957) for a specific action a is a regression step where, instead of determining the conditions under which one specific goal proposition will be achieved when a is executed, we determine the conditions under which a will lead to a number of different "goal regions" (each having different value) such that the probability of reaching each of these regions is fixed by the conditions so determined. Any set of conditions so determined for action a is such that the states having those conditions all accord the same expected value to the performance of a. The net result of this *decision theoretic regression operator* is a partitioning of state space into regions that assign different expected value to a.

A decision theoretic regression operator of this form is developed in (Boutilier, Dearden, & Goldszmidt 1995). The value functions being regressed are represented using decision trees, and the actions that are regressed through are represented using Bayes nets with tree-structured conditional probability tables. As shown there (see also (Boutilier & Dearden 1996)), classic algorithms for solving MDPs, such as value iteration or modified policy iteration, can be expressed purely in terms of decision theoretic regression, together with some tree manipulation. Unfortunately, the particular algorithm presented there assumes that actions effects are uncorrelated, imposing a restriction on the types of Bayes nets that can be used to represent actions.The aim of this paper is to correct this deficiency by describing a decision theoretic regression algorithm that handles such correlations in the effects of actions. Although this paper does not offer much in the way of a conceptual advance in the understanding of the decision theoretic regression, and builds directly on the observations in (Boutilier, Dearden, & Goldszmidt 1995; Boutilier & Dearden 1996), the modifications of these approaches to handle correlations are substantial enough, both in technical detail and in spirit, to warrant special comment.

MDPs and Their Representation

We assume that the system to be controlled can be described as a fully-observable, discrete state *Markov decision process* (Bellman 1957; Howard 1960; Puterman 1994), 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 that 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,

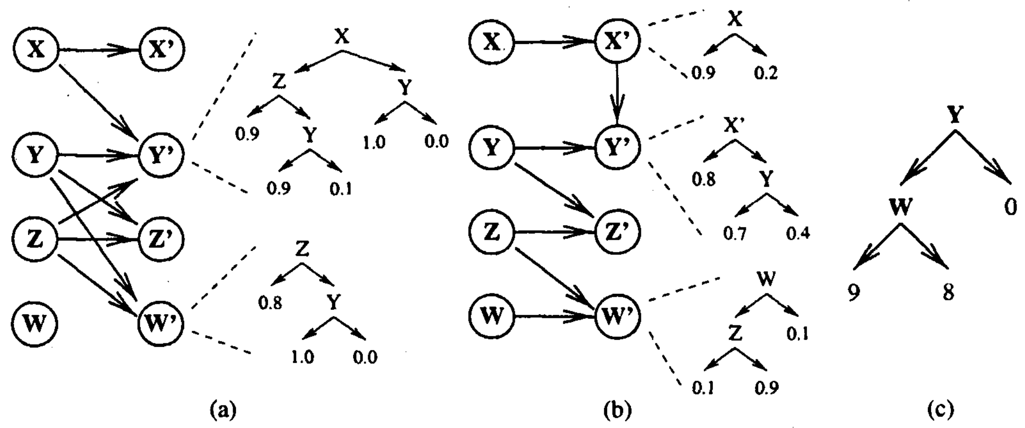

Figure 1: (a) Action network (no correlations); (b) Action network (correlations); and (c) Reward Tree. Left arrows in trees are assumed to be labeled "true", and right arrows "false".

with $R(s)$ denoting the (immediate) utility of being in state s.[1] 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 (Howard 1960):

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

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.

A number of techniques for constructing optimal policies exist. In *value iteration* (Bellman 1957) we produce a sequence of *n-step optimal value functions* V^n by setting $V^0 = R$, and defining

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

The sequence of functions V^i converges linearly to V^* in the limit. Each iteration is known as a *Bellman backup*. By performing Bellman backups with a fixed policy π, we can compute the value of π. We define the *Q-function* (Watkins & Dayan 1992), which maps state-action pairs into values, as follows:

$$Q(s, a) = \{R(s) + \beta \sum_{t \in S} Pr(s, a, t) \cdot V(t)\} \quad (2)$$

This denotes the value of performing action a at state s and subsequently executing a policy of value V. We use Q_a to denote the Q-function for a particular action a (i.e., $Q_a(s) = Q(s, a)$). Value iteration and successive approximation can be implemented by repeated construction of Q-functions (using the current value function), and the appropriate selection of Q-values (either by maximization at a particular state, or by using the policy to dictate the correct action and Q-value to apply to a state).

Action and Reward Representation

One of the key problems facing researchers regarding the use of MDPs for DTP is the "curse of dimensionality:" the number of states grows exponentially with the number of problem variables, as does the size of the representation of the MDP and the computational requirements of solution techniques. Fortunately, several good representations for MDPs, suitable for DTP, have been proposed. These include stochastic STRIPS operators (Kushmerick, Hanks, & Weld 1994; Boutilier & Dearden 1994) and dynamic Bayes nets (Dean & Kanazawa 1989; Boutilier, Dearden, & Goldszmidt 1995). We will use the latter.

We assume that a set of variables **V** describes our system. To represent each action we use a *dynamic Bayes net* (DBN) 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 these nodes. We write X' to denote that variable X after the occurrence of the action and X to denote X before the action. 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 (Boutilier, Dearden, & Goldszmidt 1995; Boutilier & Goldszmidt 1996) for a more detailed

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

discussion of this representation).[2] Figures 1(a) and (b) illustrate this representation for two different actions. We use $\Pi(X')$ to denote the parents of node X' in a network and *val*(X) to denote the values variables X (or X') can take.

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; that is, we exploit *context-specific independence* (Boutilier *et al.* 1996). In particular, we use a *decision tree* to represent the function that maps combinations of parent variable values to (conditional) probabilities. For instance, the trees in Figure 1(a) show that Z influences the probability of Y becoming true (as a consequence of the action), but only if X is true. We refer to the tree-structured CPT for node X' in the network for action a as *Tree*(X', a). We make special note of the existence of the arc between X' and Y' in Figure 1(b). This indicates that the effects of action a on X and Y are *correlated.*

A decision tree *Tree*(R) is also used to represent the reward function R, as shown in Figure 1(c). Similar trees are also used for value and Q-functions.

Regression with Uncorrelated Effects

In (Boutilier, Dearden, & Goldszmidt 1995; Boutilier & Dearden 1996) structured versions of modified policy iteration and value iteration are developed in which value functions and policies are represented using decision trees, and the DBN representation of the MDP is exploited to build these compact policies.[3] The key to these algorithms is a *decision theoretic regression operator* used to construct the Q-function for an action a given a specific value function. If the value function is tree-structured, this algorithm produces a *Q-tree*, a tree-structured representation of the Q-function that obviates the need to compute Q-values on a state-by-state basis.

Let a be the action described in Figure 1(a), and let the tree in Figure 1(c) correspond to some value function V (call it *Tree*(V)). To produce the Q-function Q_a based on V according to Equation 2, we need to determine the probabilities with which different states s make the conditions dictated by the branches of *Tree*(V) true.[4] It should be clear, since a's effects on the variables in *Tree*(V) exhibit certain regularities (as dictated by its network), that Q_a should also exhibit certain regularities. These are discovered in the following algorithm for constructing a Q-tree represent-

[2]To simplify the presentation, our examples use binary variables.

[3]See (Dietterich & Flann 1995) for a similar, though less general, method in the context of reinforcement learning (determinism and specific goal regions are assumed).

[4]We ignore the fact that states with different reward have different Q-values; these differences can be added easily once the future reward component of Equation 2 has been spelled out.

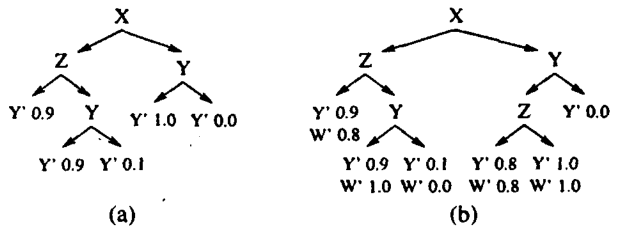

Figure 2: Decision theoretic regression: no correlations

ing (the future value component of) Q_a given *Tree*(V) and a network for a.

```
1. Generate an ordering $O_V$ of variables in Tree$(V)$.
2. Set Tree$(Q_a) = \emptyset$
3. For each variable $X$ in Tree$(V)$ (using ordering $O_V$):
   (a) Find $C$, the set of contexts (partial branches) in
       Tree$(V)$ that lead to an occurrence of $X$.
   (b) At any leaf of Tree$(Q_a)$ such that $\Pr(c) > 0$
       for some $c \in C$: replace the leaf with a copy of
       Tree$(X, a)$ at that leaf (retain $\Pr(X)$ at each leaf of
       Tree$(X, a)$; remove any redundant nodes from this
       copy; for each $Y$ ordered before $X$ such that $\Pr(Y)$
       labeled this leaf of Tree$(Q_a)$, copy $\Pr(Y)$ to each
       leaf of Tree$(X, a)$ just added.
4. At each leaf of Tree$(Q_a)$, replace probabilities labeling
   leaf with $\sum_{c \in C} \Pr(c)V(c)$, using these probabilities to
   determine $\Pr(c)$ for any context (branch) of Tree$(V)$.
```

We illustrate the algorithm on the example above. We will *regress* the variables of *Tree*(V) through action a in the order Y, W (generally, we want to respect the ordering within the tree as much as possible). We first regress Y through a, producing the tree shown in Figure 2(a). Notice that this tree accurately reflects $\Pr(Y')$ when a is executed given that the previous state satisfies the conditions labeling the branches. We then regress W through a and add the results to any branch of the tree so far where $\Pr(Y) > 0$ (see Figure 2(b)). Thus, *Tree*(W, a) is not added to the rightmost branch of the tree in Figure 2(a) since if Y is known to be false, W has no impact on reward, as dictated by *Tree*(V). Notice also that because of certain redundancies in the tests (internal nodes) of *Tree*(Y, a) and *Tree*(W, a), certain portions of *Tree*(W, a) can be deleted. Figure 2(b) now accurately describes the probabilities of both Y and W given that a is executed under the listed conditions, and thus dictates the probability of making any branch of *Tree*(V) true. The (future component of the) expected value of performing a can be computed at each leaf of this tree using $\sum\{\Pr(c)V(c) : c \in \textit{branches}(\textit{Tree}(V))\}$.

It is important to note that the justification for this very simple algorithm lies in the fact that, in the network for a, Y' and W' are independent given any context k labeling a branch of *Tree*(Q_a). This ensures that the term $\Pr(Y'|k)\Pr(W'|k)$ corresponds to $\Pr(Y', W'|k)$. Since no action effects are correlated,

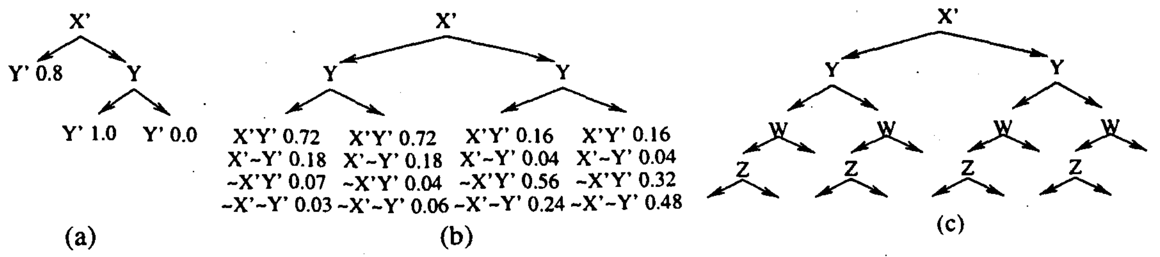

Figure 3: Decision theoretic regression with correlations

the effect of a on any variable is independent given knowledge of the previous state (i.e., the post-action variables are independent given the pre-action variables).

Regression with Correlated Effects

As noted above, the fact that action effects are uncorrelated means that knowledge of the previous state renders all post-action variables independent. This is not the case when effects are correlated as in Figure 1(b), leading several difficulties for decision theoretic regression. The first is that although we want to compute expected value of a given only the state s of pre-action variables, the probability of post-action variables that can influence value (e.g., Y') is not specified solely in terms of the pre-action state, but also involves post-action variables (e.g., X'). This difficulty is relatively straightforward to deal with, requiring that we sum out the influence of post-action variables on other post-action variables.

The second problem requires more sophistication. Because action effects are correlated, the probability of the variables in *Tree*(V) may also be correlated, so determining the probability of attaining a certain branch of *Tree*(V) by considering the "independent" probabilities of attaining the variables on the branch is doomed to failure. For instance, if both X and Y lie on a single branch of *Tree*(V), we cannot compute $\Pr(X'|s)$ and $\Pr(Y'|s)$ independently to determine the probability $\Pr(X', Y'|s)$ of attaining that branch. To deal with this, we construct Q-trees where the *joint distribution* over certain variables is computed.

Consider action a in Figure 1(b) and *Tree*(V) in Figure 1(c). Using the algorithm from the previous section to produce *Tree*(Q_a), we would first regress Y' through a to obtain the tree shown in Figure 3(a). Continuation of the algorithm will not lead to a legitimate Q-tree, since it involves a post-action variable X', so we must replace the occurrence(s) of X' with *Tree*(X', a). Since X' and Y' are correlated, we need a separate probability for each combination of their values at each leaf. To compute these probabilities we merge the probabilities from *Tree*(X', a) and Figure 3(a), resulting in Figure 3(b). For example, in the leftmost branch, $\Pr(X')$ is 0.9 from *Tree*(X', a), and $\Pr(Y'|X')$ is 0.8, so $\Pr(X', Y')$ is 0.72.[5]

As with the previous example, we must now regress W through a and add the result to any branches on which $\Pr(Y') \geq 0$ (in this case, all the branches). The final tree (minus the labels on its leaves) appears in Figure 3(c).

Each leaf in Figure 3(c) records the tables of probabilities for X' and Y', as well as a probability for W'. If there are a large number of correlated variables, these tables of probabilities can grow very large. Fortunately, it is often possible to reduce the size of the tables by summing out variables. In the Figure, we note that although X' appears in the table, it does not appear in *Tree*(V), and therefore is not required to calculate the new value tree. Therefore we can sum X' out of the tables, reducing each table to a single probability.

While we can sum out variables after constructing the new tree, we can also sum them out while tree-construction is proceeding. For example, as we are building the tree in Figure 3(b), we can observe that the probability of X' is unnecessary, and sum out its influence on Y' immediately as follows:

$$\begin{aligned}\Pr(Y'|s) &= \textstyle\sum_{x' \in val(X')} \Pr(Y'|x', s) \cdot \Pr(x'|X) \\ &= \textstyle\sum_{x' \in val(X')} \Pr(Y'|x', Y) \cdot \Pr(x'|X)\end{aligned}$$

How can we decide when to sum out the influence of a variable, and when to retain the (local) joint representation? Intuitively, we want to retain the joint distribution for a variable if we are going to "need" it again in the future. Any variable that appears in *Tree*(V) (or *Tree*(R)) will obviously be needed, but as Figure 4 shows, not all other variables can be summed out.

In Figure 4(a), assuming that *Tree*(V) is as before, when we regress Y' through a, we introduce a tree in which both X' and Z' appear. If we then substitute *Tree*(X') for X', we might be tempted to sum out X' as in Figure 3. In this example however, we "need" X' later since Y' is not independent of X' given Z' and

[5]This local joint distribution need not be computed or represented explicitly. Any *factored* representation, e.g., storing directly $\Pr(Y')$ and $\Pr(W'|Y')$, can be used. In fact, when a number of variables are correlated, we generally expect this to be the approach of choice.

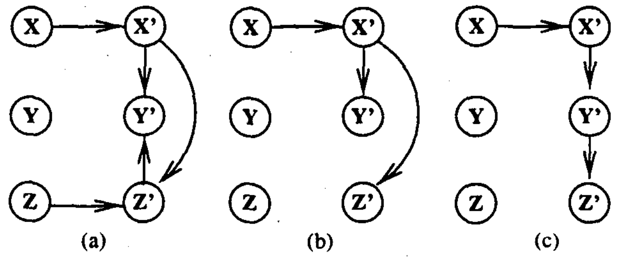

Figure 4: When to sum out variables, and when to keep their joint distribution.

$\Pi(X')$. When we later substitute *Tree*(Z') for Z', we will need the joint distribution of X' and Y' in order to compute the correct joint distribution of all three. Once all the substitutions are complete, we can then safely sum out X' since it is not required to compute the actual value of each leaf in the new tree.

Figure 4(b) illustrates a similar issue. When we regress Y' through a here, we obtain a tree containing node X', which subsequently gets replaced by *Tree*(X', a). The term $\Pr(Y')$ should be computed explicitly by summing the terms $\Pr(Y'|x') \cdot \Pr(x'|X)$ over values x'. However, looking at the *Tree*(V), we see that Z' will be regressed wherever $\Pr(Y') > 0$, and that Z' also depends on X'. This means that (ignoring any context specific independence) Z' and Y' are correlated given the previous state s. This dependence is mediated by X', so we will need to explicitly use the joint probability $\Pr(Y', X')$ to determine the joint probability $\Pr(Y', Z')$. In such a case, we say that X' is *needed* and we do not sum out its influence on Y'. In an example like this, however, once we have determined $\Pr(Y', X', Z')$ we can decide to sum out X' if it won't be needed further.

Finally, suppose that Z' depends indirectly on X', but that this dependence is mediated by Y', as in Figure 4(c). In this case, we can sum out X' and claim that X' is *not needed*: X' can only influence Z' through its effect on Y'. This effect is adequately summarized by $\Pr(Y'|X)$; and the terms $\Pr(Y', X'|X)$ are not needed to compute $\Pr(Y', Z'|X)$ since Z' and X' are independent given Y'. We provide a formal definition of *need* in Section .

Figure 4(a) illustrates another important issue. Suppose we proceed as described above, first regressing Y' through a, then substituting *Tree*(X') for X', and then *Tree*(Z') for Z'. At this point we have reintroduced X' into the tree, since it is a parent of Z', and must now eliminate it again. To prevent this occurring, we require that when a variable Y' is regressed through a, if any two of its post-action parents lie on the same branch of *Tree*(Y'), these nodes in *Tree*(Y') must be replaced by their trees in an order that respects the dependence among post-action variables in a's network. More precisely, let a *post-action ordering* O_P for action a be any ordering of variables such that, if X' is a parent of Z', then Z' occurs *before* X' in this ordering (so the ordering goes against the direction of the within-slice arcs). Post-action variables in *Tree*(Y'), or any tree obtained by recursive replacement of post-action variables, must be replaced according to some post-action ordering O_P.

Decision Theoretic Regression Algorithm

To formalize the algorithm, we assume that an action a in network form has been provided with tree-structured CPTs (that is, *Tree*(X', a) for each post-action variable X'), as well as a value tree *Tree*(V). We let O_V be an ordering of the variables within *Tree*(V), and O_P some post-action ordering for a. The following algorithm constructs a Q-tree for Q_a with respect to *Tree*(V).

1. Set *Tree*$(Q_a) = \emptyset$
2. For each variable X in *Tree*(V) (using O_V):
   (a) Find C, the set of contexts (partial branches) in *Tree*(V) that lead to an occurrence of X.
   (b) At any leaf l of *Tree*(Q_a) such that $\Pr(c) > 0$ for some $c \in C$, add *simplify*$($*Tree*$(X', a), l, k)$ to l, where k is the context in *Tree*(Q_a) leading to l (we treat l as its label).
3. At each leaf of *Tree*(Q_a), replace the probability terms (of which some may be joint probabilities) labeling the leaf with $\sum_{c \in C} \Pr(c)V(c)$, using these probabilities to determine $\Pr(c)$ for any context (branch) of *Tree*(V).

The intuitions from our earlier examples are part of the algorithm that produces *simplify*$($*Tree*$(X', a), l, k)$. Recall that l is a leaf of the current (partial) *Tree*(Q_a) and is labeled with (possibly joint) probabilities of some subset of the variables in *Tree*(V). Context k is the set of conditions under which those probabilities are valid; note that k can only consist of pre-action variables. Simplification involves the repeated replacement of the post-action variables in *Tree*(V) and the recording of joint distributions if required. It proceeds as follows:

1. *Reduce* *Tree*(X', a) for context k by deleting redundant nodes.
2. For any variables Y' in *Tree*(X', a) whose probability is part of the label for l, *replace* Y' in *Tree*(X', a), respecting the ordering O_P in replacement. That is, for each occurrence of Y' in *Tree*(X', a): (a) merge the subtrees under Y' corresponding to values y of Y' that have positive probability, deleting Y'; (b) compute $\Pr(X'|Y', m) \cdot \Pr(Y')$ for each leaf in the merged subtree (let this leaf correspond to context $m = k \wedge k'$, where k' is the branch through *Tree*(X', a)); (c) if Y' has been regressed at l or is *needed* in context m, label this leaf of the merged tree with the joint distribution over X', Y'; otherwise, sum out the influence of Y'.
3. For any remaining variables Y' in *Tree*(X', a), *replace* Y' in *Tree*(X', a), respecting the ordering O_P in replacement. To do this: (a) replace each occurrence of Y' with *Tree*(Y', a) (and reduce by context $n = k \wedge k'$, where k' is the branch through *Tree*(X', a) leading to Y'); (b) to each leaf l' of the *Tree*(Y', a) just added, merge the subtrees under Y' corresponding to values y of Y' that have positive probability at l'; (c) proceed as in Step 2.

4. Repeat Step 3 until all new post-action variables introduced at each iteration of Step 3 have been removed. For any variable removed from the tree, we construct a joint distribution with X' if it is *needed*, or sum over its value if it is not.

These steps embody the intuitions described earlier. We note that when we refer to $\Pr(Y')$ as it exists in the tree, it may be that $\Pr(Y')$ does not label the leaf explicitly but jointly with one or more other variables. In such a case, when we say that $\Pr(X', Y')$ should be computed, or Y' should be summed out, we intend that X' will become part of the explicit joint involving other variables. Any variables that are part of such a cluster are correlated with Y' and hence with X'. Variables can be summed out once they are no longer needed.

The last requirement is a formal definition of the concept of *need*—as described above, this determines when to retain a joint representation for a post-action variable that is being removed from *Tree*(Q_a). Let l be the label of the leaf where X' is being regressed, k be the context leading to that leaf, Y' be the ancestor of X' being replaced, and k' the context labeling the branch through (partially replaced) *Tree*(X', a) where the decision to compute $\Pr(X')$ or $\Pr(X', Y')$ is being made. We say that Y' is *needed* if:

1. there is a branch b of *Tree*(V) on which Y' lies, such that b has positive probability given l; or
2. there is a branch b on which Z lies, such that b has positive probability given l; $\Pr(Z')$ is not recorded in l; and there is a path from Y' to Z' in a's network that is *not* blocked by $\{X', k, k'\}$.

Concluding Remarks

We have presented an algorithm for the construction of Q-trees using a Bayes net representation of an action, with tree-structured CPTs, and a tree-structured value function. Unlike earlier approaches to this problem, this algorithm works with arbitrary Bayes net action descriptions, and is not hindered by the presence of "intra-slice" arcs. Forcing someone to specify actions without correlations is often unnatural, and the translation into a network with no intra-slice arcs (e.g., by clustering variables) may cause a blowup in the network size and a failure to exploit many independencies in decision theoretic regression.

We note that this algorithm will behave exactly as the algorithms discussed in (Boutilier, Dearden, & Goldszmidt 1995; Boutilier & Dearden 1996) if there are no correlations. While we expect MDPs often to contain actions that exhibit correlations, it seems likely that many of these correlations will be localized.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1–2):81–138.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton: Princeton University Press.

Boutilier, C., and Dearden, R. 1994. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1016–1022.

Boutilier, C., and Dearden, R. 1996. Approximating value trees in structured dynamic programming. In *Proceedings of the Thirteenth International Conference on Machine Learning*, 54–62.

Boutilier, C., and Goldszmidt, M. 1996. The frame problem and Bayesian network action representations. In *Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence*, 69–83.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, 115–123.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1104–1111.

Chapman, D., and Kaelbling, L. P. 1991. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 726–731.

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

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 574–579.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision theoretic planning. *Artificial Intelligence* 89:219–283.

Dietterich, T. G., and Flann, N. S. 1995. Explanation-based learning and reinforcement learning: A unified approach. In *Proceedings of the Twelfth International Conference on Machine Learning*, 176–184.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Cambridge: MIT Press.

Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1073–1078.

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

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

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8:279–292.