A Causal Approach to Hierarchical Decomposition of Factored MDPs

Anders Jonsson AJONSSON@CS.UMASS.EDU
Andrew Barto BARTO@CS.UMASS.EDU
Autonomous Learning Lab, Dept. of Computer Science, Univ. of Massachusetts, Amherst MA 01003, USA

Abstract

We present Variable Influence Structure Analysis, an algorithm that dynamically performs hierarchical decomposition of factored Markov decision processes. Our algorithm determines causal relationships between state variables and introduces temporally-extended actions that cause the values of state variables to change. Each temporally-extended action corresponds to a subtask that is significantly easier to solve than the overall task. Results from experiments show great promise in scaling to larger tasks.

1. Introduction

Learning and planning in tasks modeled as Markov decision processes, or MDPs, becomes increasingly difficult as the size of the state set grows. Many existing techniques do not scale well to larger tasks since the complexity increases exponentially with the number of dimensions describing a task (the "curse of dimensionality"). One way to alleviate the curse of dimensionality is to decompose a task into smaller pieces, solve each piece individually, and combine the pieces into an overall solution. We present Variable Influence Structure Analysis, or VISA, an algorithm that dynamically performs hierarchical decomposition of factored MDPs, i.e., MDPs described by several state variables.

VISA decomposes factored MDPs by introducing temporally-extended actions, which are actions that enable learning and planning on multiple levels of temporal abstraction (Dietterich, 2000; Parr & Russell, 1998; Sutton, Precup & Singh, 1999). Benefits of using temporally-extended actions include more efficient exploration and reuse of knowledge in subsequent tasks. Each temporally-extended action corresponds to a stand-alone task that can be solved independently, and there exists a principled theory of how to combine temporally-extended actions into a global solution.

The theory of temporally-extended actions does not specify how to select the stand-alone tasks. Several researchers have developed algorithms that identify temporally-extended actions from experience. One approach is to identify useful subgoals and introduce temporally-extended actions that accomplish the subgoals (Digney, 1996; McGovern & Barto, 1998; Menache, Mannor & Shimkin, 2002; Şimşek & Barto, 2004). Another approach is to perform learning in several tasks and identify temporally-extended actions that are useful across tasks (Pickett & Barto, 2002; Thrun & Schwartz, 1995). Mannor et al. (2004) recently proposed a clustering method that divides the state space into regions and introduces temporally-extended actions for moving between regions. Our approach most closely resembles that of Hengst (2002), who orders state variables according to their frequency of change and introduces one level of temporally-extended actions for each state variable.

The VISA algorithm uses a compact model of factored MDPs introduced by Boutilier, Dearden & Goldszmidt (1995). When an action is executed, the resulting value of a state variable usually depends only on a subset of the state variables. The model takes advantage of this structure by introducing dynamic Bayes networks, or DBNs (Dean & Kanazawa, 1989), approximating the transition probabilities and expected reward associated with actions. Because the DBN model does not exhaustively enumerate all states, it alleviates the curse of dimensionality. Several researchers have developed efficient algorithms for solving factored MDPs when the DBN model is given (Boutilier et al., 1995; Feng, Hansen & Zilberstein, 2003; Guestrin, Koller & Parr, 2001; Kearns & Koller, 1999).

The DBN model expresses a notion of causality be-

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

tween state variables conditional on the actions. Assume that there is a state variable $\mathbf{S}_L$ representing my location and a state variable $\mathbf{S}_M$ representing whether there is music playing. If my location is next to the stereo and I press the power button, it will cause music to play. If I am not next to the stereo, making a motion to press the button will fail to play music. There is a causal relationship between $\mathbf{S}_L$ and $\mathbf{S}_M$ conditional on the action of pressing the power button. Now assume that the state of music being played has no impact on my location. Then the causal relationship between $\mathbf{S}_L$ and $\mathbf{S}_M$ is one-way. With respect to $\mathbf{S}_L$, it is possible to ignore $\mathbf{S}_M$ without changing the dynamics of the task. There is an opportunity to decompose the task into subtasks that include $\mathbf{S}_L$ but exclude $\mathbf{S}_M$.

The type of one-way causal relationships discussed above is likely to be present in a number of realistic tasks. For example, in robot navigation tasks in which the robot has to perform additional tasks, success of the additional tasks usually depends on location, but location does not depend on the additional variables. Our algorithm exploits the opportunity to decompose tasks that exhibit one-way causal relationships.

2. Markov decision processes

A finite Markov decision process, or MDP, is a tuple $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$, where S is a finite set of states, A is a finite set of actions, $\Psi \subseteq S \times A$ is a set of admissible state-action pairs, P is a transition probability function, and R is an expected reward function. As a result of executing an action $a \in A_s \equiv \{a' \in A \mid (s, a') \in \Psi\}$ in state $s \in S$, the process transitions to a state $s' \in S$ with probability $P(s' \mid s, a)$ and receives an expected reward $R(s, a)$. The objective of an MDP is to find a stochastic policy π that maximizes the expected discounted return $R_t = E\{\sum_{k=t}^{\infty} \gamma^{k-t} R(s_k, a_k)\}$, where $\gamma \in (0, 1]$ is a discount factor, by selecting action $a \in A$ with probability $\pi(s, a)$ in each state $s \in S$.

A factored MDP is described by a set of state variables $\{\mathbf{S}_i\}_{i\in D}$, where D is a set of indices. The set of states $S = \times_{i\in D} Val(\mathbf{S}_i)$ is the cross-product of the value sets $Val(\mathbf{S}_i)$ of each state variable $\mathbf{S}_i$. A state $s \in S$ assigns a value $s_i \in Val(\mathbf{S}_i)$ to each state variable $\mathbf{S}_i$. We use the coffee task (Boutilier et al., 1995), in which a robot has to deliver coffee to its user, to illustrate factored MDPs. The coffee task is described by six binary variables: $\mathbf{S}_{\mathtt{L}}$, the robot's location (office or coffee shop); $\mathbf{S}_{\mathtt{U}}$, whether the robot has an umbrella; $\mathbf{S}_{\mathtt{R}}$, whether it is raining; $\mathbf{S}_{\mathtt{W}}$, whether the robot is wet; $\mathbf{S}_{\mathtt{C}}$, whether the robot has coffee; and $\mathbf{S}_{\mathtt{H}}$, whether the user has coffee. To distinguish between variable values we use the notation $Val(\mathbf{S}_i) = \{i, \bar{i}\}$,

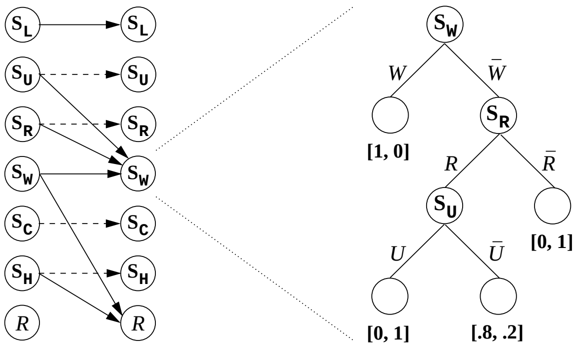

Figure 1. The DBN for action GO in the coffee task

where $\overline{L}$ = office and L = coffee shop. An example state is $s = (\overline{L}, U, R, \overline{W}, C, \overline{H})$. The robot has four actions: GO, causing its location to change and the robot to get wet if it is raining and it does not have an umbrella; BC (buy coffee) causing it to hold coffee if it is in the coffee shop; GU (get umbrella) causing it to hold an umbrella if it is in the office; and DC (deliver coffee) causing the user to hold coffee if the robot has coffee and is in the office. All actions have a chance of failing. The robot gets a reward of 0.9 when the user has coffee (H) plus a reward of 0.1 when it is dry ($\overline{W}$).

For any $D' \subseteq D$, let $S_{D'} = \times_{i\in D'} Val(\mathbf{S}_i)$ be the joint value set of the subset of state variables $\{\mathbf{S}_i\}_{i\in D'}$, and let $f_{D'} : S \to S_{D'}$ be the projection from S onto $S_{D'}$. We define a context $c_{D'} \in S_{D'}$, $D' \subseteq D$, to be a partial assignment of values to the state variables.

2.1. DBN model

The DBN model (Boutilier et al., 1995) of a factored MDP contains one DBN per action. Figure 1 shows the DBN for action GO in the coffee task. Nodes on the left represent state variables at the current time step, and nodes on the right represent state variables at the next time step. There are also nodes corresponding to expected reward. The value of a state variable $\mathbf{S}_i$ as a result of executing GO depends on the values of state variables that have edges to $\mathbf{S}_i$ in the DBN. A dashed line indicates that a state variable is unaffected by action GO. Figure 1 also illustrates the conditional probability tree, or CPT, associated with state variable $\mathbf{S}_{\mathtt{W}}$. We assume that there are no edges between state variables at a same time step; in this case the transition probabilites of the factored MDP are approximated as $P(s' \mid s, a) \approx \prod_{i\in D} P_i(s'_i \mid f_{D_i}(s), a)$, where P_i are the conditional probabilities associated with state variable $\mathbf{S}_i$, and $D_i \subseteq D$ indicates the state variables that have edges to $\mathbf{S}_i$ in the DBN for a.

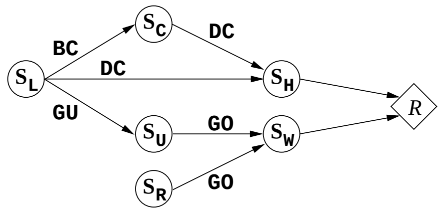

Figure 2. The SVIG of the coffee task

2.2. Options

We use the options framework (Sutton et al., 1999) to represent temporally-extended actions. In MDP $\mathcal{M}$, an option is a tuple $o = \langle I, \pi, \beta \rangle$, where $I \subseteq S$ is an initiation set, π is a policy, and β is a termination condition function. Option o can be executed in any state $s \in I$, repeatedly selects actions $a \in A$ according to π, and terminates in state $s' \in S$ with probability $\beta(s')$. An action a can be viewed as an option with initiation set $I = \{s \in S \mid (s, a) \in \Psi\}$ whose policy always selects a and that terminates in all states with probability 1. An MDP $\mathcal{M}$ together with a set of options O constitute a semi-Markov decision process, or SMDP.

An option o can be viewed as a stand-alone task given by the option SMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi_o, P_o, R_o \rangle$, where $S_o \subseteq S$ is the option state set, O_o is the set of options that o selects from, $\Psi_o \subseteq S_o \times O_o$ is the set of admissible state-option pairs, determined by the initiation sets of options in O_o, and P_o is a transition probability function, determined by the transition probability function P of the underlying MDP and the policies of the options in O_o. The expected reward function R_o associated with o can be selected to reflect the option's desired behavior. The option SMDP $\mathcal{M}_o$ implicitly defines option o's policy π as the solution to $\mathcal{M}_o$.

3. The VISA algorithm

The VISA algorithm uses causal relationships between state variables to decompose a factored MDP. The first step of the algorithm is to construct a state variable influence graph, or SVIG, indicating the causal relationships between state variables. The SVIG contains one node per state variable plus one node corresponding to reward. A directed edge between two state variables $\mathbf{S}_i$ and $\mathbf{S}_j$ (or between $\mathbf{S}_i$ and the reward node R) indicates that there is a causal relationship between $\mathbf{S}_i$ and $\mathbf{S}_j$ (R) conditional on at least one action, i.e., that there is an edge between $\mathbf{S}_i$ and $\mathbf{S}_j$ (R) in the DBN for that action. We remove reflexive edges and label each edge with the associated actions. Figure 2 illustrates the SVIG of the coffee task.

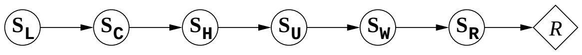

Figure 3. HEX-Q's state variable ordering in the coffee task

We are interested in determining one-way causal relationships: a state variable $\mathbf{S}_i$ causes a state variable $\mathbf{S}_j$ to change, but $\mathbf{S}_j$ does not cause $\mathbf{S}_i$ to change. In the SVIG, a causal relationship between $\mathbf{S}_i$ and $\mathbf{S}_j$ is one-way if there is a directed path between $\mathbf{S}_i$ and $\mathbf{S}_j$ but no directed path between $\mathbf{S}_j$ and $\mathbf{S}_i$. We can isolate one-way causal relationships by computing the strongly connected components, or SCCs, of the SVIG. We then compute the component graph of the SVIG, i.e., the graph with one node per SCC. The component graph is acyclic so all causal relationships are one-way. In the coffee task, each node in the SVIG is its own SCC, so the component graph is identical to the SVIG.

To introduce options we use a formalism similar to the HEX-Q algorithm (Hengst, 2002). HEX-Q determines an ordering on the state variables by randomly executing actions and counting the frequency with which the value of each state variable changes. The state variable whose value changes the most frequently becomes the lowest variable in the ordering. For each state variable $\mathbf{S}_i$ in the ordering, the HEX-Q algorithm identifies exit states $\langle s_i, a \rangle$, pairs of a state variable value $s_i \in Val(\mathbf{S}_i)$ and an action $a \in A$, that cause the value of the next state variable in the ordering to change. The HEX-Q algorithm introduces an option for each exit state, and the options on one level of the hierarchy become actions on the next level.

Even though the HEX-Q algorithm achieved some early success, the frequency of change may not be an accurate indicator of how state variables influence each other. In addition, the ordering does not capture the fact that the value of a state variable may depend on multiple other state variables. Figure 3 illustrates the state variable ordering that the HEX-Q algorithm comes up with in the coffee task. There are several differences between this ordering and the SVIG. The ordering wrongly concludes that state variable $\mathbf{S}_\mathtt{W}$ influences $\mathbf{S}_\mathtt{R}$, when it is really the other way around. The ordering also fails to capture the fact that the value of $\mathbf{S}_\mathtt{H}$ depends on both $\mathbf{S}_\mathtt{L}$ and $\mathbf{S}_\mathtt{C}$.

3.1. Identifying options

The VISA algorithm uses the component graph of the SVIG to represent variable relationships. For each SCC with incoming edges, there exists a set of exits $\langle c_{D'}, a \rangle$, i.e., pairs of a context $c_{D'} \in S_{D'}$, $D' \subseteq D$, and

Table 1. Exits identified in the coffee task

SCC	Change	Exit
$\mathbf{S}_{\mathtt{C}}$	$\overline{C} \to C$	$\langle(L), \mathtt{BC}\rangle$
$\mathbf{S}_{\mathtt{C}}$	$C \to \overline{C}$	$\langle(\overline{L}), \mathtt{DC}\rangle$
$\mathbf{S}_{\mathtt{H}}$	$\overline{H} \to H$	$\langle(\overline{L}, C), \mathtt{DC}\rangle$
$\mathbf{S}_{\mathtt{U}}$	$\overline{U} \to U$	$\langle(\overline{L}), \mathtt{GU}\rangle$
$\mathbf{S}_{\mathtt{W}}$	$\overline{W} \to W$	$\langle(\overline{U}, R), \mathtt{GO}\rangle$

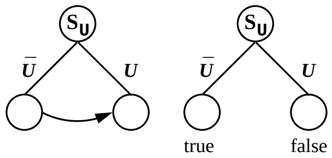

Figure 4. The transition graph and reachability tree of $\mathbf{S}_{\mathtt{U}}$

an action $a \in A$, that cause the values of state variables in the SCC to change. Here, $D' \subseteq D$ indicates a subset of the state variables in SCCs that have edges to the SCC being analyzed. VISA identifies exits by searching in the CPTs of the DBN model, and introduces an option o for each exit $\langle c_{D'}, a\rangle$. A similar causal approach to task decomposition was recently proposed in the context of deterministic planning (Helmert, 2004).

In the coffee task, two SCCs ($\mathbf{S}_{\mathtt{L}}$ and $\mathbf{S}_{\mathtt{R}}$) have no incoming edges, so VISA does not identify options for them. The SCC $\mathbf{S}_{\mathtt{W}}$ has incoming edges from $\mathbf{S}_{\mathtt{U}}$ and $\mathbf{S}_{\mathtt{R}}$. In the CPT in Figure 1, VISA identifies one leaf (third from the left) for which the value of $\mathbf{S}_{\mathtt{W}}$ changes as a result of executing `GO`. The leaf expresses the fact that if the robot is dry, it is raining, and the robot does not have an umbrella, the robot becomes wet with probability 0.8 if it executes `GO`. The exit corresponding to this change is $\langle(\overline{U}, R), \mathtt{GO}\rangle$, i.e., executing `GO` in a state s whose projection $f_{\{\mathtt{U},\mathtt{R}\}}(s)$ equals $(\overline{U}, R)$ causes the value of $\mathbf{S}_{\mathtt{W}}$ to change from $\overline{W}$ to W with non-zero probability. Table 1 shows a complete list of exits identified by VISA in the coffee task. We label each option with the change it causes; for example, $\overline{W} \to W$ is the option associated with the exit $\langle(\overline{U}, R), \mathtt{GO}\rangle$.

3.2. Initiation set

Two factors influence the initiation set I of option o. Option o should only be admissible in states from which it is possible to reach the context $c_{D'}$. Option o should also only be admissible in states for which its associated exit causes the value of at least one state variable in the corresponding SCC to change. For example, option $\overline{W} \to W$ should only be admissible in states that assign $\overline{U}$ to $\mathbf{S}_{\mathtt{U}}$ and R to $\mathbf{S}_{\mathtt{R}}$. The robot has no action for getting rid of an umbrella, and it cannot affect whether it is raining, so it can only get wet if it does not have an umbrella and it is raining. In addition, option $\overline{W} \to W$ should only be admissible in states that assign $\overline{W}$ to $\mathbf{S}_{\mathtt{W}}$, since otherwise the option cannot cause the value of $\mathbf{S}_{\mathtt{W}}$ to change from $\overline{W}$ to W.

Existing techniques that identify temporally-extended actions usually ignore the problem of determining initiation sets. In contrast, the VISA algorithm uses a sophisticated method to construct the initiation set I of an option o. For each SCC, VISA constructs a transition graph that represents possible transitions between contexts in the joint value set of its state variables. Each transition graph is in the form of a tree in which possible transitions are represented as directed edges between the leaves. Possible transitions are determined using the CPTs of the DBN model. VISA uses the transition graphs to construct a tree that classifies states on the basis of whether or not the context $c_{D'}$ of the exit associated with option o is reachable. Figure 4 illustrates the transition graph of the SCC $\mathbf{S}_{\mathtt{U}}$ in the coffee task as well as the corresponding reachability tree indicating whether the context $(\overline{U}, R)$ of the exit $\langle(\overline{U}, R), \mathtt{GO}\rangle$ is reachable (true) or not (false).

VISA also builds a tree that classifies states on the basis of whether or not the associated exit changes the value of at least one state variable in the corresponding SCC. This tree can also be constructed from the CPTs of the DBN model. In our example, states that assign $\overline{W}$ to $\mathbf{S}_{\mathtt{W}}$ map to a leaf labeled true, and states that assign W to $\mathbf{S}_{\mathtt{W}}$ map to a leaf labeled false, since the exit $\langle(\overline{U}, R), \mathtt{GO}\rangle$ does not cause the value of $\mathbf{S}_{\mathtt{W}}$ to change if its current value is W. The initiation set I of option o is implicitly defined by the two trees constructed by VISA. A state $s \in S$ is an element in I if and only if s maps to a leaf labeled true in both trees.

3.3. Termination condition function

The termination condition function β is defined as $\beta(s) = 1$ for each state s whose projection $f_{D'}(s)$ onto $S_{D'}$ equals $c_{D'}$. $\beta(s)$ is also 1 for states $s \notin I$, i.e., when the process can no longer reach the context $c_{D'}$. In all other cases, $\beta(s) = 0$. In other words, option o terminates as soon as the process reaches the context $c_{D'}$ or as soon as it becomes impossible to reach $c_{D'}$. We refer to options discovered by VISA as exit options since they are slightly different than regular options. If option o successfully terminates in the context $c_{D'}$, action a of its associated exit is always executed.

3.4. Policy

VISA cannot directly define the policy π of option o since it does not know the best strategy for reaching the context $c_{D'}$. Instead, VISA constructs an option SMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi_o, P_o, R_o \rangle$ for option o that implicitly defines its policy π. We let $S_o = S$ and define O_o as the set of options that affect state variables in SCCs that have edges to the SCC being analyzed. For example, the option set O_o of the exit option $\overline{W} \rightarrow W$ only needs to include the exit option $\overline{U} \rightarrow U$, since that is the only option that affects the SCCs $\mathbf{S}_{\mathtt{U}}$ or $\mathbf{S}_{\mathtt{R}}$ that have edges to $\mathbf{S}_{\mathtt{W}}$. Note that primitive actions may affect state variables for which there are no options; for example, action GO affects state variable $\mathbf{S}_{\mathtt{L}}$.

If there are lower-level options that cause the process to leave the initiation set of an option in O_o, VISA includes these options in O_o as well. For example, the exit option $\overline{U} \rightarrow U$ causes the process to leave the initiation set of the exit option $\overline{W} \rightarrow W$. If the robot does not have an umbrella and it is raining, the exit option $\overline{W} \rightarrow W$ will no longer be admissible as a result of executing the exit option $\overline{U} \rightarrow U$ causing the robot to hold an umbrella. In other words, an option whose option set O_o includes the exit option $\overline{W} \rightarrow W$ should include the exit option $\overline{U} \rightarrow U$ as well.

We define the expected reward function R_o as -1 everywhere except when option o terminates unsuccessfully, in which case we administer a large negative reward. This ensures that the policy π of option o attempts to reach the context $c_{D'}$ as quickly as possible. Ψ_o is determined by the initiation sets of the options in O_o. The VISA algorithm does not represent the transition probability function P_o explicitly. It is possible to construct a DBN model for each option similar to the DBN model for the primitive actions. However, there is currently no technique that enables us to do so without enumerating all states. Since the whole point of VISA is to alleviate the curse of dimensionality, we want to avoid enumerating the states. Instead, we will use reinforcement learning techniques, which do not require explicit knowledge of the transition probabilities, to learn the policy π of option o.

3.5. State abstraction

To achieve our goal of decomposing the original MDP $\mathcal{M}$ into smaller tasks, the option SMDP $\mathcal{M}_o$ should be significantly easier to solve than $\mathcal{M}$. This is where causality really matters. Because of one-way causal relationships, the option SMDP can ignore all state variables that do not influence state variables in SCCs that have edges to the SCC being analyzed. For example, the option SMDP of the exit option $\overline{W} \rightarrow W$ can ignore state variables $\mathbf{S}_{\mathtt{C}}$, $\mathbf{S}_{\mathtt{H}}$ and $\mathbf{S}_{\mathtt{W}}$, since neither of these influence the state variables $\mathbf{S}_{\mathtt{U}}$ and $\mathbf{S}_{\mathtt{R}}$ that have edges to $\mathbf{S}_{\mathtt{W}}$. Intuitively, the values of these state variables do not matter for the purpose of reaching the context $c_{D'}$ of the associated exit. It is trivial to show that this reduction preserves optimality of $\mathcal{M}_o$.

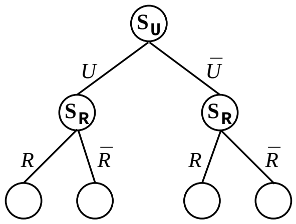

Figure 5. The policy tree of the exit option $\overline{W} \rightarrow W$

VISA reduces the complexity of the option SMDP even further by ignoring all state variables that are not in immediate parent SCCs of the SCC being analyzed. For example, the option SMDP of exit option $\overline{W} \rightarrow W$ ignores state variable $\mathbf{S}_{\mathtt{L}}$, since that is not an immediate parent of $\mathbf{S}_{\mathtt{W}}$. If SCCs with edges to the SCC being analyzed have no common ancestor SCCs in the component graph, it is possible to show that this reduction preserves optimality of $\mathcal{M}_o$ as well (we omit the proof for lack of space). If there are common ancestor SCCs in the component graph, the resulting solution to the option SMDP will only be approximately optimal. However, as the algorithm scales to increasingly large tasks, we believe that the reduction in complexity will be worth the loss of exact optimality.

Boutilier et al. (1995) introduced the use of policy trees to represent stochastic policies. The benefit of using a policy tree is that the number of leaves in the tree may be smaller than the actual number of states. The VISA algorithm uses a policy tree to represent the policy π of an exit option o. VISA constructs the policy tree by merging the transition graphs of SCCs that have edges to the SCC being analyzed. In other words, the policy tree only distinguishes between state variables in SCCs that have edges to the SCC being analyzed. Figure 5 shows the policy tree of the exit option $\overline{W} \rightarrow W$. VISA reduces the number of effective states in the option SMDP of the exit option $\overline{W} \rightarrow W$ from $2^6 = 64$ to 4.

3.6. Task option

The VISA algorithm also introduces an option, which we call the task option, associated with the reward node in the component graph of the SVIG. VISA uses the same strategy to construct the task option as the other options. The option SMDP of the task option

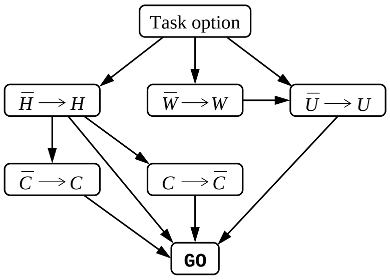

Figure 6. The hierarchy of options in the coffee task

only considers SCCs that have edges to the reward node. However, the expected reward function of the task option SMDP is the same as the expected reward function of the original MDP $\mathcal{M}$. Solving the task option gives us a (possibly approximate) solution of the original MDP which uses the other options discovered by VISA. Figure 6 shows the hierarchy of options that VISA comes up with in the coffee task.

3.7. Exit transformations

Sometimes it is possible to transform exits in order to take further advantage of causality. Consider the two exits $\langle(\overline{L}), \mathtt{DC}\rangle$ and $\langle(\overline{L}, C), \mathtt{DC}\rangle$ in the coffee task. These exits are almost identical: their associated exit options both terminate in states that assign $\overline{L}$ to $\mathbf{S}_{\mathtt{L}}$ and execute action DC following successful termination. Recall that $C \rightarrow \overline{C}$ is the exit option associated with the exit $\langle(\overline{L}), \mathtt{DC}\rangle$, causing the value of $\mathbf{S}_{\mathtt{C}}$ to change from C to $\overline{C}$. We can transform the exit $\langle(\overline{L}, C), \mathtt{DC}\rangle$ to $\langle(C), C \rightarrow \overline{C}\rangle$, i.e., reach a state that assigns C to $\mathbf{S}_{\mathtt{C}}$ and execute option $C \rightarrow \overline{C}$ following termination. The benefit of this transformation is that the exit option $\overline{H} \rightarrow H$ associated with the exit $\langle(\overline{L}, C), \mathtt{DC}\rangle$ no longer has to consider the value of $\mathbf{S}_{\mathtt{L}}$, effectively removing an edge in the component graph of the SVIG.

3.8. Limitations of the algorithm

VISA only decomposes a task if there are two or more SCCs in the component graph of the SVIG, i.e., if there is at least one instance of one-way causality. In addition, VISA works best when there are relatively few exits that cause the values of state variables in an SCC to change. If there are many context-action pairs that cause changes, it is not particularly useful to introduce an option for each of them. Instead, VISA merges two SCCs if they are linked by too many exits. Since the option SMDPs are stand-alone, the hierarchy discovered by VISA enables recursive optimality at best, as opposed to hierarchical optimality (Dietterich, 2000).

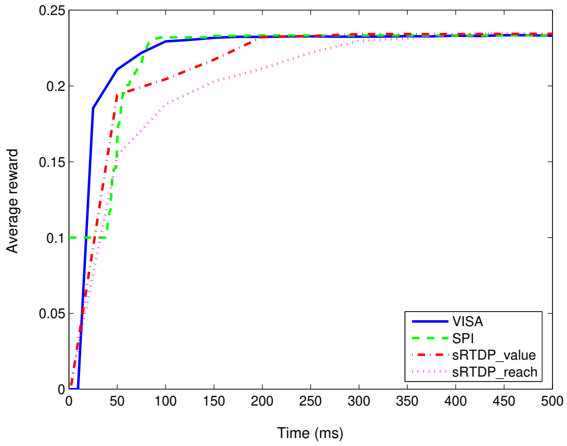

Figure 7. Results in the coffee task

4. Results

We compared the VISA algorithm to two algorithms that also use the DBN model: Structured Policy Iteration, or SPI (Boutilier et al., 1995), and symbolic Real-Time Dynamic Programming, or sRTDP (Feng et al., 2003). We performed experiments with each algorithm in four tasks: the coffee task, the Taxi task (Dietterich, 2000), the Factory task (Hoey et al., 1999), and a simplified version of the autonomous guided vehicle (AGV) task of Ghavamzadeh & Mahadevan (2001).

Figure 7 shows the results in the coffee task. The graph for each algorithm illustrates the average reward over 100 trials. The graph for VISA includes the time it takes to decompose the factored MDP. We used SMDP Q-learning to learn the option policies, which reduces to regular Q-learning for policies that select between primitive actions. sRTDP uses algebraic decision diagrams, or ADDs, to store conditional probabilities. Prior to executing, sRTDP computes complete action ADDs; the graphs include the time it takes to do this. sRTDP uses two heuristics, value and reach, to group states into abstract states. We report results of both heuristics. All algorithms were coded in Java, except that the CUDD library (written in C) was used to manipulate ADDs through the Java Native Interface.

Figure 8 shows the results in the Taxi task. The graphs illustrate the average reward over 100 trials. The reason VISA outperforms the other algorithms is that VISA decomposes the task into smaller, stand-alone tasks that are easier to solve without ever enumerating the entire state space. VISA reduces the number of state-action pairs from 3,000 to approximately 800.

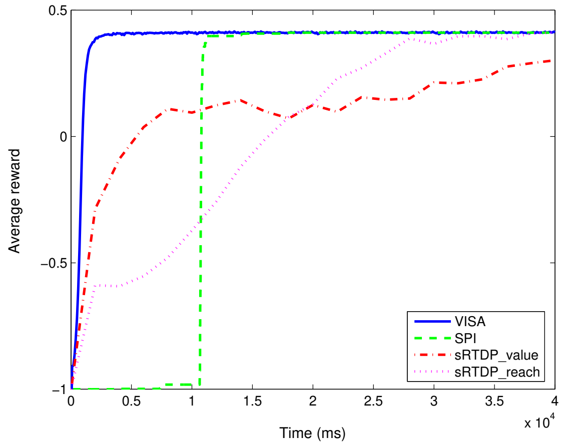

Figure 8. Results in the Taxi task

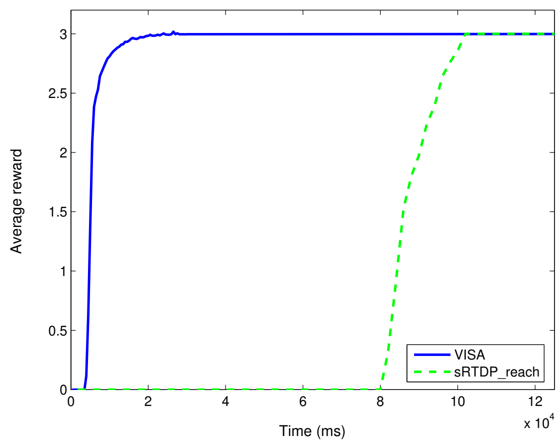

Figure 9. Results in the Factory task

In the Factory task, a robot has to assemble a component made of two objects. The task is described by 17 binary variables for a total of 130,000 states, and the robot has 14 actions. Figure 9 shows the results in the Factory task of the VISA algorithm and sRTDP using the reach heuristic. The VISA algorithm decomposes the task in 5 seconds and learning converges after 20 seconds. In comparison, it takes sRTDP 80 seconds to compute complete action ADDs. Each subsequent iteration of the value heuristic takes 20-60 seconds, which causes convergence to be very slow. The reach heuristic performs better and is included in the figure. SPI ran out of memory after running for several hours.

In the AGV task, an AGV agent has to transport pieces between machines in a manufacturing workshop. We simplified the task by reducing the number of machines to 2 and setting the processing time of machines to 0. Figure 10 shows the result of the VISA algorithm in the AGV task, averaged over 100 trials. In this case, VISA reduces the number of state-action pairs from 450,000 to approximately 16,000. VISA decomposes the task in roughly 6 seconds and learning converges after 20 seconds. In comparison, SPI ran out of memory after 3 hours. It takes sRTDP 4 minutes to compute complete action ADDs, and each subsequent iteration takes 20-60 seconds. The shortest solution path requires 89 actions, and sRTDP performs one iteration per action, so it takes sRTDP more than half an hour to complete the task once, let alone converge.

5. Conclusion

We have presented VISA, an algorithm that dynamically decomposes a factored MDP when a DBN model of the MDP is given. The VISA algorithm determines one-way causal relationships between state variables and identifies exits that cause the value of state variables to change. For each exit, VISA uses sophisticated tree manipulations to construct an associated exit option, i.e., an option that executes an additional action following successful termination. Instead of learning a policy for the original MDP, VISA constructs a solution by learning the policies of the exit options. Because of causality, the policies of the exit options are significantly easier to learn than the policy of the original MDP, reducing complexity.

We compared the VISA algorithm to two other algorithms that also assume that a DBN model of the MDP is given. In smaller tasks, the advantage of VISA algorithm is not apparent, but as the size of a task grows, the decomposition identified by VISA provides a significant reduction in learning time.

It is not realistic to assume that a DBN model of a factored MDP is always given prior to learning. An important research topic is to devise algorithms for learning the DBN model from experience. There exist algorithms in the literature for learning DBNs from experience. However, these algorithms usually fix the values of a subset of the variables in order to determine variable correlations. Unless there is a generative model, it is not possible to fix the values of state variables in an MDP. In other words, we believe that algorithms for learning DBN models of factored MDPs have to take into account the specific nature of MDPs.

We would also like to determine bounds on the quality of the approximation when there are common ancestor SCCs in the component graph. In other words, we

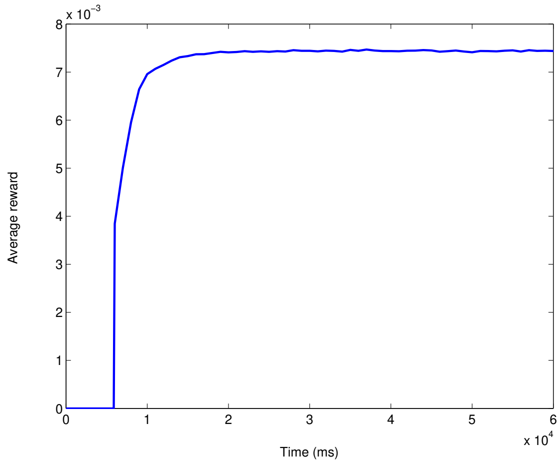

Figure 10. Result of the VISA algorithm in the AGV task

would like to determine the tradeoff between the reduction in complexity and the loss of optimality. This sort of analysis may help us decide when to reduce the size of an option SMDP and when to maintain a larger size that preserves a higher degree of optimality.

Finally, we are working on a method for constructing a DBN model of each exit option, similar to the DBN model of individual actions. We hope to be able to construct DBN models for the options without exhaustively enumerating all states. If successful, it will be possible to apply planning algorithms, such as policy iteration, to learn the policies of the options, in addition to reinforcement learning.

Acknowledgements

The authors would like to thank Alicia "Pippin" Wolfe and Mohammad Ghavamzadeh for useful comments on this paper. This work was partially funded by NSF grants ECS-0218125 and CCF-0432143.

References

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995) Exploiting structure in policy construction. *IJCAI*, 14: 1104–1113.

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

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303.

Digney, B. (1996) Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. *From animals to animats*, 4: 363–372.

Feng, Z., Hansen, E., & Zilberstein, Z. (2003) Symbolic generalization for on-line planning. *UAI*, 19: 209–216.

Ghavamzadeh, M., & Mahadevan, S. (2001) Continuous-time hierarchical reinforcement learning. *ICML*, 18: 186–193.

Guestrin, C., Koller, D., & Parr, R. (2001) Max-norm projections for factored MDPs. *IJCAI*, 17: 673–680.

Helmert, M. (2004) A planning heuristic based on causal graph analysis. *ICAPS*, 16: 161–170.

Hengst, B. (2002) Discovering hierarchy in reinforcement learning with HEXQ. *ICML*, 19: 243–250.

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999) SPUDD: Stochastic Planning using Decision Diagrams. *UAI*, 15: 279–288.

Kearns, M., & Koller, D. (1999) Efficient reinforcement learning in factored MDPs. *IJCAI*, 16: 740–747.

Mannor, S., Menache, I., Hoze, A., & Klein, U. (2004) Dynamic abstraction in reinforcement learning via clustering. *ICML*, 21: 560–567.

McGovern, A., & Barto, A. (2001) Automatic discovery of subgoals in reinforcement learning using diverse density. *ICML*, 18: 361–368.

Menache, I., Mannor, S., & Shimkin, N. (2002) Q-Cut – Dynamic discovery of sub-goals in reinforcement learning. *ECML*, 14: 295–306.

Parr, R., & Russell, S. (1998) Reinforcement learning with hierarchies of machines. *NIPS*, 10: 1043–1049.

Pickett, M., & Barto, A. (2002) PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. *ICML*, 19: 506–513.

Şimşek, Ö., & Barto, A. (2004) Using relative novelty to identify useful temporal abstractions in reinforcement learning. *ICML*, 21: 751–758.

Sutton, R., Precup, D., & Singh, S. (1999) Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112: 181–211.

Thrun, S., & Schwartz, A. (1995) Finding structure in reinforcement learning. *NIPS*, 8: 385–392.