Journal of Machine Learning Research 7 (2006) 2259-2301 Submitted 10/05; Revised 7/06; Published 11/06

Causal Graph Based Decomposition of Factored MDPs

Anders Jonsson ANDERS.JONSSON@UPF.EDU
Departament de Tecnologia
Universitat Pompeu Fabra
Passeig de Circumval·lació, 8
08003 Barcelona, Spain

Andrew Barto BARTO@CS.UMASS.EDU
Computer Science Department
University of Massachusetts
Amherst MA 01003, USA

Editor: Carlos Guestrin

Abstract

We present Variable Influence Structure Analysis, or VISA, an algorithm that performs hierarchical decomposition of factored Markov decision processes. VISA uses a dynamic Bayesian network model of actions, and constructs a causal graph that captures relationships between state variables. In tasks with sparse causal graphs VISA exploits structure by introducing activities that cause the values of state variables to change. The result is a hierarchy of activities that together represent a solution to the original task. VISA performs state abstraction for each activity by ignoring irrelevant state variables and lower-level activities. In addition, we describe an algorithm for constructing compact models of the activities introduced. State abstraction and compact activity models enable VISA to apply efficient algorithms to solve the stand-alone subtask associated with each activity. Experimental results show that the decomposition introduced by VISA can significantly accelerate construction of an optimal, or near-optimal, policy.

Keywords: Markov decision processes, hierarchical decomposition, state abstraction

1. Introduction

Markov decision processes, or MDPs, are widely used to model stochastic control tasks. Many researchers have developed algorithms that determine optimal or near-optimal decision policies for MDPs. However, most of these algorithms scale poorly as the size of a task grows. Much recent research on MDPs has focused on finding task structure that makes it possible to simplify construction of a useful policy. In this paper, we present Variable Influence Structure Analysis, or VISA, an algorithm that identifies task structure in factored MDPs and combines hierarchical decomposition and state abstraction to exploit task structure and simplify policy construction. VISA was first introduced in a conference paper (Jonsson and Barto, 2005); this paper provides more detail and additional insights as well as a new section on compact activity models.

Hierarchical decomposition exploits task structure by introducing stand-alone policies (also known as activities, macro-actions, temporally-extended actions, options, or skills) that can take multiple time steps to execute. We use the term activity (Harel, 1987) to denote such a stand-alone policy. Activities can exploit repeating structure by representing subroutines that are executed multiple times during solution of a task. If an activity has been learned in one task, it can be reused

in other tasks that require execution of the same subroutine. Activities also enable more efficient exploration by providing high-level behavior that enables a decision maker to look ahead to the completion of the associated subroutine. There exist three major models of activities in reinforcement learning: Hierarchical Abstract Machines, or HAMs (Parr and Russell, 1998), options (Sutton et al., 1999), and MAXQ (Dietterich, 2000a).

It may not be apparent to a system designer how to select subroutines that enable efficient hierarchical decomposition. To take full advantage of hierarchical decomposition, a system should be able to identify useful subroutines on its own. Several researchers have developed algorithms that use task-specific knowledge to identify useful subroutines. One approach is to identify useful subgoals and introduce activities that accomplish the subgoals (Digney, 1996; McGovern and Barto, 2001; Şimşek and Barto, 2004). Another approach is to solve several tasks and identify activities that are useful across tasks (Pickett and Barto, 2002; Thrun and Schwartz, 1996). There also exist algorithms that use graph theory to cluster states into regions and introduce activities for moving between regions (Menache et al., 2002; Mannor et al., 2004; Şimşek et al., 2005). Other algorithms introduce activities that cause the values of specific variables to change (Hengst, 2002; Singh et al., 2005).

Hierarchical decomposition is intimately related to state abstraction, that is, ignoring part of the available information to reduce the effective size of the state space. At each moment, only some of the information that is part of the state description may be relevant for selecting an optimal action. For example, the color of the wall is most likely irrelevant for the task of navigating to the front door of a building. State abstraction compresses the state space by grouping together states that only differ on irrelevant information. Each group of states can be treated as a single state, reducing the complexity of policy computation. Dean and Givan (1997) showed that under certain conditions, the optimal policy of an MDP is preserved under state abstraction.

Each activity can be viewed as a stand-alone subtask that can be solved independently. If each subtask is as difficult to solve as the original task, hierarchical decomposition actually increases the complexity of finding an optimal policy. However, if state abstraction is used to simplify the solution of each subtask, hierarchical decomposition can significantly accelerate policy computation. In particular, information that is relevant for one subtask may be irrelevant for another. In other words, it makes sense to perform state abstraction separately for each subtask (Dietterich, 2000b; Jonsson and Barto, 2001).

VISA uses a compact model of factored MDPs first suggested by Boutilier et al. (1995). When an action is executed, the resulting value of a state variable depends on the values of state variables prior to executing the action. In many cases, the resulting value is conditionally independent of a subset of the state variables at the previous time step. The compact model uses dynamic Bayesian networks, or DBNs (Dean and Kanazawa, 1989), to represent the effect of actions in factored MDPs. Since DBNs encode conditional independence, the model can represent the effect of actions using much less memory than the number of states. Several researchers have developed algorithms that take advantage of the DBN model to efficiently compute policies of factored MDPs (Boutilier et al., 1995; Feng et al., 2003; Guestrin et al., 2001; Hoey et al., 1999; Kearns and Koller, 1999).

2. Overview

In addition to being compact, the DBN model contains information about the preconditions necessary for an action to have the desired effect. For example, consider a task in which the objective is

to play music, described by two state variables: one representing my current location, and the other representing the current sound level. There are actions for changing my location, and an action to turn on the stereo. Being next to the stereo is a precondition for causing music to play when making a motion to turn on the stereo, a fact that is encoded in the transition probabilities of the DBN model. In other words, there is a causal relationship between the location variable and the sound level variable, conditional on the action of turning on the stereo.

To change the value of the sound level variable, it is first necessary to satisfy the precondition of being next to the stereo. Thus, a useful activity is one that causes my location to be next to the stereo. Given such an activity, it is straightforward to solve the task: first execute the activity that causes my location to be next to the stereo, and then turn on the stereo. The idea behind VISA is to use the DBN model to identify the preconditions necessary to change the value of each state variable and introduce activities for satisfying those preconditions. The result is a hierarchy of activities that can be used in a compact representation of the solution to the factored MDP. The HEX-Q algorithm (Hengst, 2002) is based on similar ideas, but does not use the DBN model to identify preconditions.

The goal of VISA is to introduce activities in such a way that their associated subtasks are easier to solve than the original task. Since the DBN model implicitly represents relationships between state variables, it is relatively easy to determine which state variables and activities are relevant for solving a particular subtask. This makes it possible to perform state abstraction for subtasks by ignoring irrelevant state variables and activities. For example, while causing my location to be next to the stereo, it is possible to ignore differences in sound level, since the sound level typically has no impact on location. If the subtasks are sufficiently easy to solve, hierarchical decomposition can lead to a significant reduction in computational complexity.

VISA, described in Section 4, uses the DBN model to construct a causal graph describing state variable relationships. If two state variables mutually influence each other, it is not possible to introduce activities that change the value of one without taking into account the value of the other. Consequently, it is not possible to perform state abstraction in a way that makes the associated subtasks easier to solve. State variables that mutually influence each other correspond to cycles in the causal graph, so VISA gets rid of cycles by identifying the strongly connected components of the graph and constructing a component graph with one node per component.

The algorithm then identifies exits (Hengst, 2002), that is, combinations of variable values and actions that cause the value of some state variable to change. For each exit, VISA introduces an activity that solves the subtask of changing the corresponding variable value. VISA uses the causal graph to identify state variables and activities that are relevant for solving the subtask, and performs state abstraction by ignoring irrelevant state variables and activities. At the top level, the algorithm introduces an activity that corresponds to the original MDP. Experimental results show that the decomposition generated by VISA can significantly accelerate construction of an optimal or near-optimal policy.

If VISA had access to compact models of activities, similar to the DBN model of primitive actions, it could apply more efficient algorithms to construct the stand-alone policies of activities. To fully model the stand-alone subtask associated with each activity, it is necessary to determine the transition probabilities of the lower-level activities used to solve the subtask. However, existing methods cannot determine transition probabilities of activities without enumerating the state space. Since the state space grows exponentially with the number of state variables, this seems like a bad idea. Instead, the implementation of VISA in Section 4 uses reinforcement learning (Sutton and

Barto, 1998), which does not require knowledge of transition probabilities, to learn the policy of each activity.

In Section 5, we describe an algorithm that constructs compact activity models without enumerating the state space. We decompose computation of an activity model by considering the conditional probabilities of one state variable at a time. The result is a DBN model for each activity identical to the DBN model of primitive actions. VISA can then apply the more efficient algorithms that take advantage of DBN models, accelerating construction of the stand-alone policies even further. Experimental results show that our algorithm can construct compact activity models without significantly decelerating solution of a task.

3. Background

In this section, we provide a background to the problem that our algorithm attempts to solve, and we introduce notation of concepts that we use throughout the paper.

3.1 Markov Decision Processes

A finite Markov decision process, or MDP (Bellman, 1957), 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. Let $A_s \equiv \{a \in A \mid (s,a) \in \Psi\}$ be the set of admissible actions in state $s \in S$. Ψ is such that for each state $s \in S$, A_s is non-empty, that is, there is at least one admissible action for each state. As a result of executing action $a \in A_s$ in state $s \in S$, the process transitions to state $s' \in S$ with probability $P(s' \mid s,a)$ and provides the decision maker with an expected reward $R(s,a)$. P is such that for each admissible state-action pair $(s,a) \in \Psi$, $\sum_{s' \in S} P(s' \mid s,a) = 1$.

For each state $s \in S$ and each action $a \in A_s$, a stochastic policy π selects action a in state s with probability $\pi(s,a)$. π is such that for each state $s \in S$, $\sum_{a \in A_s} \pi(s,a) = 1$. In the discounted case, the optimal value function V^* associated with MDP $\mathcal{M}$ is defined by the Bellman optimality equation:

$$V^*(s) = \max_{a \in A_s} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s') \right], \tag{1}$$

where γ is a discount factor. An optimal policy π^* is any stochastic policy that, in each state $s \in S$, assigns positive probabilities only to actions in the set

$$A^*(s) \equiv \arg\max_{a \in A_s} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s') \right].$$

A factored MDP is described by a set of discrete state variables $\mathbf{S}$. Each state variable $S_i \in \mathbf{S}$ takes on values in its domain $\mathcal{D}(S_i)$. The set of states $S \subseteq \times_{S_i \in \mathbf{S}} \mathcal{D}(S_i)$ is a subset of the Cartesian product of the state variable domains. A state $\mathbf{s} \in S$ is an assignment of values to the set of state variables $\mathbf{S}$. Let $f_{\mathbf{C}}$, $\mathbf{C} \subseteq \mathbf{S}$, be a projection such that if $\mathbf{s}$ is an assignment to $\mathbf{S}$, $f_{\mathbf{C}}(\mathbf{s})$ is $\mathbf{s}$'s assignment to $\mathbf{C}$. We define a *context* $\mathbf{c}$ as an assignment of values to the subset of state variables $\mathbf{C} \subseteq \mathbf{S}$.

3.2 Coffee Task

We illustrate factored MDPs using the coffee task (Boutilier et al., 1995), in which a robot has to deliver coffee to its user. The coffee task is described by six binary state variables: S_{L}, the robot's

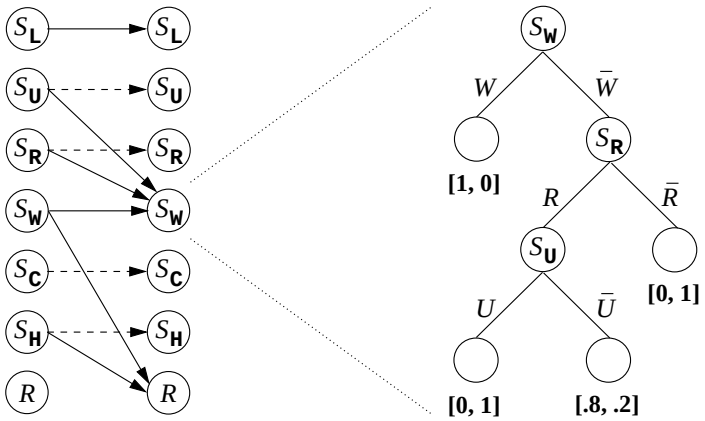

Figure 1: The DBN for action GO in the coffee task

location (office or coffee shop); $S_{\mathtt{U}}$, whether the robot has an umbrella; $S_{\mathtt{R}}$, whether it is raining; $S_{\mathtt{W}}$, whether the robot is wet; $S_{\mathtt{C}}$, whether the robot has coffee; and $S_{\mathtt{H}}$, whether the user has coffee. To distinguish between variable values we use the notation $\mathcal{D}(S_i) = \{i, \bar{i}\}$, which has obvious meaning for all state variables except $S_{\mathtt{L}}$, where we use L to denote the coffee shop and $\overline{L}$ to denote the office. 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 whenever the user has coffee plus a reward of 0.1 whenever it is dry.

3.3 DBN Model

Boutilier et al. (1995) developed a compact model of factored MDPs that uses dynamic Bayesian networks, or DBNs (Dean and Kanazawa, 1989), to represent the effect of actions. The DBN model contains one DBN for each action $a \in A$ of a factored MDP. Figure 1 illustrates the DBN for action GO in the coffee task. The DBN has two nodes for each state variable plus two nodes representing expected reward. Nodes on the left represent the values of variables prior to executing GO, and nodes on the right represent the values after executing GO. The value of a state variable S_i as a result of executing GO depends on the values of state variables that have edges to S_i in the DBN. Let $\mathbf{Pa}(S_i) \subseteq \mathbf{S}$ denote the subset of state variables with edges to S_i. A dashed line indicates that a state variable is unaffected by GO.

In the DBN for action a, each state variable S_i is associated with a conditional probability distribution P_i^a that determines the value of S_i after executing a. Like Boutilier et al. (1995), we assume that conditional probabilities are stored in trees. Figure 1 illustrates the conditional probability tree associated with state variable $S_{\mathtt{W}}$ and action GO. For example, if the robot is dry ($\overline{W}$), it is raining (R), and the robot does not have an umbrella ($\overline{U}$), the robot becomes wet with probability 0.8 after executing GO. We assume that there are no edges between state variables in the same layer of the DBN. Consequently, the DBN model cannot represent arbitrary transition probabilities. Instead, the

transition probabilities are approximated according to $P(\mathbf{s}' \mid \mathbf{s}, a) \approx \prod_{S_i \in \mathbf{S}} P_i^a(S_i = f_{\{S_i\}}(\mathbf{s}') \mid \mathbf{Pa}(S_i) = f_{\mathbf{Pa}(S_i)}(\mathbf{s}))$.

3.4 Options

We use *options* (Sutton et al., 1999) to model activities. Given an MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$, 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 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 $a \in A$ can be viewed as an option with initiation set $I = \{s \in S \mid (s,a) \in \Psi\}$ whose policy always selects a and terminates in all states with probability 1. Adding options to the action set of an MDP forms a semi-Markov decision process, or SMDP (Puterman, 1994). It is possible to construct hierarchies of options in which the options on one level selects among options on a lower level.

Ravindran (2004) showed that an option o is associated with a stand-alone task given by the option SMDP $\mathcal{M}_o = \langle S, O_o, \Psi_o, P_o, R_o \rangle$, where O_o is a set of lower-level options. The set of admissible state-option pairs $\Psi_o \subseteq S \times O_o$ is determined by the initiation sets of options in O_o. The transition probability function P_o is determined by the transition probability function P of the underlying MDP and the policies and termination functions of the options in O_o. The expected reward function R_o is independent of the expected reward function R of the underlying MDP and can be selected to reflect the desired behavior of option o. The policy π of option o can be defined as any optimal policy of the option SMDP $\mathcal{M}_o$.

SMDP Q-learning (Bradtke and Duff, 1995) maintains estimates of the optimal option-value $Q(s,o)$, representing the return for executing option o in state s. Following execution of an option o in state s, the option-value is updated using the following update rule:

$$Q(s,o) \leftarrow Q(s,o) + \alpha \left[r + \gamma^k \max_{o' \in O_o} Q(s',o') - Q(s,o) \right],$$

where s' is the state in which o terminated, k is the number of time steps elapsed during the execution of o, r is the cumulative discounted reward during this time, and α is the learning rate. Parr (1998) showed that SMDP Q-learning eventually converges to an optimal policy when the learning rate α is appropriately decreased towards 0.

3.5 State Abstraction

We use partitions to represent state abstraction in MDPs. A partition Λ of the set of states S is a collection of disjoint subsets, or blocks, $\lambda \subseteq S$ such that $\bigcup_{\lambda \in \Lambda} \lambda = S$. $[s]_\Lambda \in \Lambda$ denotes the block to which state $s \in S$ belongs. A function $f : S \to X$ from S onto an arbitrary set X induces a partition Λ_f of S such that for each pair of states $(s_i, s_j) \in S^2$, $[s_i]_{\Lambda_f} = [s_j]_{\Lambda_f}$ if and only if $f(s_i) = f(s_j)$. Let Λ_1 and Λ_2 be two partitions of S. Partition Λ_1 refines Λ_2, denoted $\Lambda_1 \leq \Lambda_2$, if and only if, for each pair of states $(s_i, s_j) \in S^2$, $[s_i]_{\Lambda_1} = [s_j]_{\Lambda_1}$ implies that $[s_i]_{\Lambda_2} = [s_j]_{\Lambda_2}$. The relation $\leq$ is a partial ordering on the set of partitions of S.

Dean and Givan (1997) defined two properties of partitions of the set of states S. A partition Λ has the stochastic substitution property if, for each pair of states $(s_i, s_j) \in S^2$, each action $a \in A$ and each block $\lambda \in \Lambda$, $[s_i]_\Lambda = [s_j]_\Lambda$ implies that $\sum_{s_k \in \lambda} P(s_k \mid s_i, a) = \sum_{s_k \in \lambda} P(s_k \mid s_j, a)$. Λ is reward respecting if for each pair of states $(s_i, s_j) \in S^2$ and each action $a \in A$, $[s_i]_\Lambda = [s_j]_\Lambda$ implies that $R(s_i, a) = R(s_j, a)$. A partition Λ that has the stochastic substitution property and is reward

respecting induces a reduced MDP which has fewer states and preserves optimality (Dean and Givan, 1997). Ravindran (2004) developed a theory of MDP homomorphisms and extended the above definitions to partitions of the set Ψ of admissible state-action pairs.

4. VISA

Variable Influence Structure Analysis, or VISA (Jonsson and Barto, 2005), is an algorithm that analyzes causal relationships between state variables to perform hierarchical decomposition of factored MDPs. VISA uses the DBN model of factored MDPs to compactly represent transition probabilities and expected reward. However, VISA makes additional use of the DBN model. The conditional probability distributions of the DBN model specify which preconditions have to hold for the value of a state variable to change as a result of executing an action. The aim of VISA is to facilitate variable value changes by introducing activities that satisfy those preconditions. For example, if the task is to play music, a useful activity is one that causes my location to be next to the stereo, since being next to the stereo is a precondition for successfully making a motion to turn it on.

Algorithm 1 VISA

```
1: Input: DBN model of a factored MDP
2: construct the causal graph of the task
3: identify the strongly connected components of the causal graph
4: for each strongly connected component
5:     identify a set of exits that cause the values of state variables in the component to change
6:     for each exit
7:         construct the components of an option SMDP
8:         use the causal graph to perform state abstraction for the option SMDP
9: apply reinforcement learning techniques to learn the policy of each option SMDP
```

Algorithm 1 gives a high-level description of VISA. Before decomposing the task, VISA uses the DBN model to construct a causal graph that determines how state variables influence each other. A state variable influences another if it appears in the precondition of an action that changes the value of the latter. If two state variables mutually influence each other, changing the value of one variable depends on the value of the other. Thus, it is impossible to decompose the task by introducing activities that exclusively change the value of one of the variables. State variables that mutually influence each other correspond to cycles in the causal graph. VISA gets rid of cycles by identifying the strongly connected components of the causal graph. State variables in a strongly connected component are treated as a single variable for the purpose of decomposition.

For each strongly connected component, VISA searches the conditional probability distributions of the DBN model for *exits* (Hengst, 2002), that is, pairs of a precondition and an action that cause the value of a state variable in the component to change. For each exit, VISA introduces an exit option, that is, an activity that terminates when the precondition of the exit is met and then executes the exit action. In other words, the purpose of an exit option is to change the value of a state variable by first satisfying the necessary precondition and then executing the appropriate action. To determine the policy of each exit option, VISA constructs the components of an option SMDP and defines the policy as a solution to the resulting option SMDP.

To simplify learning the policy of an exit option, VISA performs state abstraction for the option SMDP. From the causal graph it is easy to identify a set of state variables that are irrelevant, that

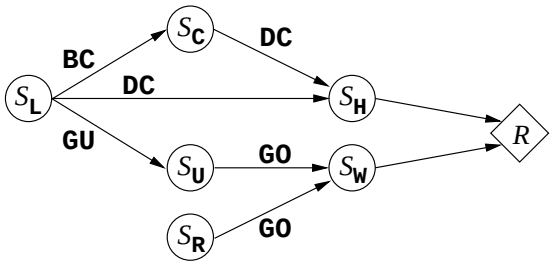

Figure 2: The causal graph of the coffee task

is, do not influence state variables whose values appear in the precondition. VISA performs state abstraction by ignoring differences in the values of irrelevant state variables. In addition, the option SMDP only needs to include actions and options that change the values of state variables that appear in the precondition. The resulting state abstraction significantly reduces the complexity of learning the exit option policies. The causal graph implicitly defines a hierarchy of options in which an exit option that changes the value of a state variable in a strongly connected component selects between options that change the values of state variables in strongly connected components with incoming edges.

4.1 Causal Graph

The first step of VISA is to construct a causal graph representing the causal relationships between state variables. The causal graph contains one node per state variable plus one node corresponding to expected reward. There is a directed edge between two state variables S_j and S_i if and only if there exists an action $a \in A$ such that there is an edge between S_j and S_i in the DBN for a. In other words, each edge in the causal graph represents a causal relationship between two state variables conditional on one or several actions. The algorithm labels each edge with the actions that give rise to the causal relationship.

Recall that Figure 1 shows the DBN for action GO in the coffee task. There are several interesting things to note. For each state variable S_i, the value of S_i as a result of executing GO depends on the value of S_i prior to executing GO. In other words, each node in the causal graph should have an associated reflexive edge. However, we are not interested in the causal relationship of a state variable onto itself, so we remove reflexive edges in the causal graph. Also, there are edges from state variable $S_{\mathtt{U}}$ to state variable $S_{\mathtt{W}}$ in the DBN, as well as from $S_{\mathtt{R}}$ to $S_{\mathtt{W}}$. Consequently, there should be an edge from $S_{\mathtt{U}}$ to $S_{\mathtt{W}}$ in the causal graph labeled GO, as well as an edge from $S_{\mathtt{R}}$ to $S_{\mathtt{W}}$ labeled GO.

The causal graph of the coffee task is shown in Figure 2. Note that the edges from the DBN for action GO have been incorporated, as well as edges from the DBNs for the other actions. Also note that there are no cycles in the causal graph. However, this is not true for arbitrary tasks, since it is possible for state variables to mutually influence each other. VISA gets rid of cycles in the causal graph by identifying the strongly connected components of the graph, each consisting of one or several state variables that are pairwise connected through directed paths. It is possible to construct a component graph in which each node is a strongly connected component, and which has an edge between two nodes if and only if there is an edge in the causal graph between a state variable of the

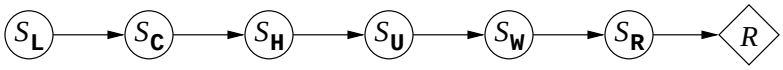

Figure 3: HEX-Q's state variable ordering in the coffee task

first component and a state variable of the second component. The component graph is guaranteed to contain no cycles. In the coffee task, each state variable in the causal graph is its own strongly connected component, so the component graph is identical to the causal graph.

The expected reward node deserves additional explanation. Just as for the other variables, there is an edge in the causal graph between a state variable S_i and the expected reward node if and only if there is an action $a \in A$ such that the expected reward as a result of executing a depends on the value of S_i. All edges to the expected reward node are incoming, since the value of a state variable never depends on the expected reward received at the previous time step. Thus, the expected reward as a result of executing any action only depends on the values of state variables with edges to the expected reward node in the causal graph. For the purpose of optimizing reward, it is only necessary to consider actions and options that change the values of those state variables.

4.2 Identifying Exits

VISA builds on ideas from the HEX-Q algorithm (Hengst, 2002), an algorithm that also performs hierarchical decomposition of factored MDPs. The HEX-Q algorithm first 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, and so on. For each state variable S_i in the ordering, the HEX-Q algorithm identifies exits $\langle k, a \rangle$, pairs of a state variable value $k \in \mathcal{D}(S_i)$ 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, 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 heuristic 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 causal graph. The ordering wrongly concludes that state variable $S_{\mathtt{W}}$ influences $S_{\mathtt{R}}$, when it is really the other way around. The ordering also fails to capture the fact that the value of $S_{\mathtt{H}}$ depends on both $S_{\mathtt{L}}$ and $S_{\mathtt{C}}$.

VISA also searches for exits that cause the values of state variables to change. However, instead of the frequency of change heuristic, VISA uses the causal graph to determine how state variables influence each other. Since the causal graph more realistically describes the causal relationships between state variables, VISA is able to successfully decompose more general tasks than the HEX-Q algorithm. Also, since the value of a state variable may depend on several other state variables, an exit $\langle \mathbf{c}, a \rangle$ in VISA is composed of a context $\mathbf{c}$ and an action $a \in A$. Recall that a context $\mathbf{c}$ is an assignment of values to a subset $\mathbf{C} \subseteq \mathbf{S}$ of the state variables.

VISA searches for exits in the conditional probability trees of the DBN model. Consider the conditional probability tree associated with state variable $S_{\mathtt{W}}$ and action `GO` in Figure 1. The third

EXIT	VARIABLE	CHANGE
$\langle(), \texttt{GO}\rangle$	$S_{\texttt{L}}$	$\overline{L} \rightarrow L$, $L \rightarrow \overline{L}$
$\langle(S_{\texttt{L}} = L), \texttt{BC}\rangle$	$S_{\texttt{C}}$	$\overline{C} \rightarrow C$
$\langle(S_{\texttt{L}} = \overline{L}), \texttt{DC}\rangle$	$S_{\texttt{C}}$	$C \rightarrow \overline{C}$
$\langle(S_{\texttt{L}} = \overline{L}, S_{\texttt{C}} = C), \texttt{DC}\rangle$	$S_{\texttt{H}}$	$\overline{H} \rightarrow H$
$\langle(S_{\texttt{L}} = \overline{L}), \texttt{GU}\rangle$	$S_{\texttt{U}}$	$\overline{U} \rightarrow U$
$\langle(S_{\texttt{U}} = \overline{U}, S_{\texttt{R}} = R), \texttt{GO}\rangle$	$S_{\texttt{W}}$	$\overline{W} \rightarrow W$

Table 1: Exits identified in the coffee task

leaf from the left is associated with states that assign $\overline{W}$ to $S_{\texttt{W}}$, R to $S_{\texttt{R}}$, and $\overline{U}$ to $S_{\texttt{U}}$. As a result of executing action GO in such states, the value of $S_{\texttt{W}}$ becomes W with probability 0.8. Since the value of state variable $S_{\texttt{W}}$ changes from $\overline{W}$ to W with non-zero probability, VISA generates an exit $\langle(S_{\texttt{U}} = \overline{U}, S_{\texttt{R}} = R), \texttt{GO}\rangle$ that causes the value of $S_{\texttt{W}}$ to change. The context of the exit is determined by the values of state variables on the path from the root to the leaf. Note that the value of $S_{\texttt{W}}$ does not appear in the exit since that is the state variable whose value the exit changes. Also note that the exit $\langle(S_{\texttt{U}} = \overline{U}, S_{\texttt{R}} = R), \texttt{GO}\rangle$ does not cause the value of $S_{\texttt{W}}$ to change with probability 1, so to effectuate the change the robot may have to execute GO multiple times in the context $(S_{\texttt{U}} = \overline{U}, S_{\texttt{R}} = R)$.

Table 1 shows a complete list of exits identified by VISA in the coffee task. The table shows which state variable is affected by each exit together with the resulting change. To generate these exits, VISA had to search through each leaf of each conditional probability tree of the DBN model. At each leaf, the algorithm examined whether the value of state variable S_i changes, where S_i is the state variable whose conditional probabilities the current tree represents. In other words, the complexity of this part of the algorithm is proportional to the number of leaves of the conditional probability trees.

4.3 Exit Transformations

Sometimes it is possible to transform exits in order to take further advantage of causality. Consider the two exits $\langle(S_{\texttt{L}} = \overline{L}), \texttt{DC}\rangle$ and $\langle(S_{\texttt{L}} = \overline{L}, S_{\texttt{C}} = C), \texttt{DC}\rangle$ in the coffee task. These are almost identical: their associated exit options both terminate in states that assign the value $\overline{L}$ to state variable $S_{\texttt{L}}$ and execute action DC following successful termination. Recall that $C \rightarrow \overline{C}$ is the exit option associated with the exit $\langle(S_{\texttt{L}} = \overline{L}), \texttt{DC}\rangle$, causing the value of $S_{\texttt{C}}$ to change from C to $\overline{C}$. The effect of the exit $\langle(S_{\texttt{L}} = \overline{L}, S_{\texttt{C}} = C), \texttt{DC}\rangle$ is equivalent to the effect of a transformed exit $\langle(S_{\texttt{C}} = C), C \rightarrow \overline{C}\rangle$, that is, reach a state that assigns C to $S_{\texttt{C}}$ 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 $\langle(S_{\texttt{L}} = \overline{L}, S_{\texttt{C}} = C), \texttt{DC}\rangle$ no longer has to care about the value of $S_{\texttt{L}}$, effectively removing an edge in the component graph of the task. After identifying an exit, VISA compares it to exits identified for ancestor strongly connected components, and performs exit transformations when possible.

4.4 Introducing Exit Options

For each exit $\langle \mathbf{c}, a\rangle$ with a non-empty context $\mathbf{c}$, VISA introduces an option $o = \langle I, \pi, \beta\rangle$. Option o terminates in any state $\mathbf{s} \in S$ whose projection $f_{\mathbf{C}}(\mathbf{s})$ onto $\mathbf{C}$ equals $\mathbf{c}$. We refer to an option

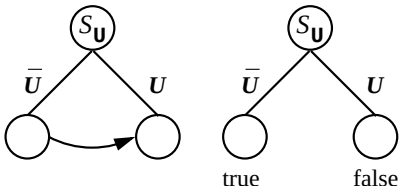

Figure 4: The transition graph (left) and reachability tree (right) of the component $S_{\mathtt{U}}$

introduced by VISA as an *exit option*. Unlike regular options, an exit option associated with an exit $\langle \mathbf{c}, a \rangle$ executes action a following termination. Note that it is not necessary to introduce options for exits with empty contexts, since these options are in fact equivalent to primitive actions. For example, VISA identifies an exit $\langle (), \mathtt{GO} \rangle$ in the coffee task. Executing action GO in any state causes the location of the robot to change, so the exit option associated with this exit is equivalent to the primitive action GO. As we shall see, it is still useful to detect exits with empty contexts.

In the coffee task example, we adopt the convention of referring to an exit option using the change that it causes, since this is an unambiguous and simple notation. For example, option $\overline{W} \rightarrow W$ is the exit option associated with the exit $\langle (S_{\mathtt{U}} = \overline{U}, S_{\mathtt{R}} = R), \mathtt{GO} \rangle$ that causes the value of $S_{\mathtt{W}}$ to change from $\overline{W}$ to W with non-zero probability. In general, several exits may cause the same change in the value of a variable, and VISA would introduce an exit option for each of these exits, so this notation is not always unambiguous.

4.4.1 INITIATION SET

The initiation set I of exit option o determines when o is admissible, that is, the subset of states in which it is possible to execute o. Two factors influence the initiation set. Option o should only be admissible in states from which it is possible to reach the associated context $\mathbf{c}$. For example, option $\overline{W} \rightarrow W$ 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. Option o should also only be admissible if it causes the value of at least one state variable to change. In our example, option $\overline{W} \rightarrow W$ 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.

VISA includes a method for constructing the initiation set of each exit option. For each strongly connected component, the algorithm 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 conditional probability trees of the DBN model. Figure 4 (left) shows the transition graph of the strongly connected component containing the state variable $S_{\mathtt{U}}$ in the coffee task. The robot can acquire an umbrella by executing the exit option $\overline{U} \rightarrow U$, so there is a corresponding edge in the transition graph between the leaf associated with states that assign $\overline{U}$ to $S_{\mathtt{U}}$ and the leaf associated with states that assign U to $S_{\mathtt{U}}$. However, the robot has no action for getting rid of an umbrella, so there is no edge going the other way.

VISA uses the transition graphs to construct a set of trees that represent the initiation set I. For each transition graph, VISA constructs a *reachability tree* that classifies states based on whether

(true) or not (false) the associated context is reachable. Figure 4 (right) shows the reachability tree for S_U associated with the context $(S_U = \overline{U}, S_R = R)$ of the exit associated with $\overline{W} \rightarrow W$. Similarly, VISA constructs a reachability tree for S_R. VISA also constructs a tree that classifies states based on whether or not the associated exit changes the value of at least one state variable in the corresponding strongly connected component. In our example, states that assign $\overline{W}$ to S_W map to a leaf labeled true, and states that assign W to S_W map to a leaf labeled false. The initiation set I of option o is implicitly defined by the trees constructed by VISA. A state $\mathbf{s} \in S$ is an element in I if and only if $\mathbf{s}$ maps to a leaf labeled true in each tree. Algorithm 2 summarizes the method used by VISA to construct the initiation set of an exit option.

Algorithm 2 Initiation set

1: Input: DBN model, component graph, exit $\langle \mathbf{c}, a \rangle$
2: identify the set of components that contain state variables whose values appear in $\mathbf{c}$
3: **for each** component in this set
4: use the DBN model to construct a transition graph of the component
5: perform search in the transition graph to construct a reachability tree
6: construct a tree that determines whether $\langle \mathbf{c}, a \rangle$ changes the value of at least one variable
7: define the initiation set as the set of states that map to true in each tree

Alternatively, reachability could be computed directly using operations on trees or algebraic decision diagrams (ADDs). Feng and Hansen (2002) showed how to compute forward reachability in factored MDPs using ADDs. Since an exit represents termination of an exit option, here we are interested in computing backward reachability: from which states is it possible to reach the exit? Algorithmically, this is similar to forward reachability. However, VISA makes further use of transition graphs, and once the transition graphs have been constructed, reachability is easily computed using depth-first search on the reverse edges. For this reason we chose to stick with the above approach.

4.4.2 TERMINATION CONDITION

An exit option terminates as soon as it reaches the context $\mathbf{c}$ of its associated exit $\langle \mathbf{c}, a \rangle$, or as soon as it can no longer reach $\mathbf{c}$. Even though an exit option executes action a following termination, we can still represent termination of the option using the standard termination condition function β. For an exit option, $\beta(\mathbf{s})$ is 1 for states in the set $\{\mathbf{s} \in S \mid f_{\mathbf{C}}(\mathbf{s}) = \mathbf{c}\}$, where $\mathbf{c}$ is the associated context. $\beta(\mathbf{s})$ is also 1 for states $\mathbf{s} \notin I$, that is, when the process can no longer reach the associated context $\mathbf{c}$. In all other cases, $\beta(\mathbf{s}) = \mathbf{0}$.

4.4.3 POLICY

VISA cannot directly define the policy of an exit option since it does not know the best strategy for reaching the associated context $\mathbf{c}$. Instead, the algorithm 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. First, the algorithm defines $S_o = S$. Next, the algorithm finds all strongly connected components that contain at least one state variable whose value appears in the context $\mathbf{c}$ associated with option o. The algorithm defines O_o as the set of options that cause the values of state variables in those strongly connected components to change. For example, consider the exit option $\overline{W} \rightarrow W$ and its associated context $(S_U = \overline{U}, S_R = R)$. Two strongly connected components contain state variables whose values appear in the context: the

strongly connected component containing $S_{\mathtt{U}}$, and the strongly connected component containing $S_{\mathtt{R}}$. A single option, $\overline{U} \rightarrow U$, causes the values of state variables to change in the former component, while no option causes the values of state variables to change in the latter. In other words, the option set O_o of $\overline{W} \rightarrow W$ only needs to include the exit option $\overline{U} \rightarrow U$. Note that primitive actions may change the values of state variables in strongly connected components for which there are no options; for example, action `GO` changes the value of state variable $S_{\mathtt{L}}$.

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.

VISA defines the expected reward function R_o as -1 everywhere except when option o terminates unsuccessfully, in which case the algorithm administers a large negative reward. This ensures that the policy of option o attempts to reach the context $\mathbf{c}$ as quickly as possible. Note that this may not be optimal in terms of the expected reward of the original task; we address this issue at a later point. The set of admissible state-option pairs, Ψ_o, is determined by the initiation sets of the options in O_o. VISA 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 constructs DBN models of options without enumerating all states. Since a goal of VISA is to alleviate the curse of dimensionality, we want to avoid enumerating the states. Instead, VISA uses reinforcement learning (Sutton and Barto, 1998), which does not require explicit knowledge of the transition probabilities, to learn the policy of each option. In the next section, we develop an algorithm that constructs DBN models of options identified by VISA without enumerating all states, as an alternative to reinforcement learning. The transition graphs of strongly connected components play a part in constructing DBN models of options, but nothing prevents options from changing the values of state variables that do not appear in the associated context, which makes the issue slightly more complicated.

4.5 State Abstraction

VISA simplifies learning in the option SMDPs by performing state abstraction separately for each exit option. This is where causality really matters. Let us consider all strongly connected components that contain at least one state variable whose value appears in the context $\mathbf{c}$ associated with an option. Let $\mathbf{Z} \subseteq \mathbf{S}$ denote the subset of state variables contained in those strongly connected components. Let $\mathbf{Y} \subseteq \mathbf{S}$ denote the subset of state variables S_i such that either $S_i \in \mathbf{Z}$ or there is a directed path in the causal graph from S_i to a state variable in $\mathbf{Z}$. For example, in the case of exit option $\overline{W} \rightarrow W$, $\mathbf{Z} = \{S_{\mathtt{U}}, S_{\mathtt{R}}\}$ and $\mathbf{Y} = \{S_{\mathtt{L}}, S_{\mathtt{U}}, S_{\mathtt{R}}\}$, since there is a directed path from $S_{\mathtt{L}}$ to $S_{\mathtt{U}}$ in the causal graph of the coffee task.

Recall that the goal of an exit option o is to reach the associated context $\mathbf{c}$. We know that $\mathbf{C} \subseteq \mathbf{Z} \subseteq \mathbf{Y}$, that is, that the state variables whose values appear in the context $\mathbf{c}$ are contained in $\mathbf{Y}$. We also know that there are no edges from any state variable $S_j \notin \mathbf{Y}$ to any state variable $S_i \in \mathbf{Y}$; if there were, state variable S_j would have been included in $\mathbf{Y}$. It follows that the option SMDP $\mathcal{M}_o$ can ignore the values of state variables not in $\mathbf{Y}$ since they have no influence on the variables in $\mathbf{C}$, whose values we want to set to $\mathbf{c}$.

More formally, we can define a partition that satisfies the stochastic substitution property and is reward respecting (cf. Section 3.5), and thus guaranteed to preserve an optimal solution to the option SMDP $\mathcal{M}_o$.

Theorem 1 *The projection function $f_{\mathbf{Y}}$ induces a partition $\Lambda_{\mathbf{Y}}$ of S that has the stochastic substitution property and is reward respecting.*

Proof The projection function $f_{\mathbf{Y}}$ induces a partition $\Lambda_{\mathbf{Y}}$ of S such that two states $\mathbf{s}_1$ and $\mathbf{s}_2$ belong to the same block if and only if $f_{\mathbf{Y}}(\mathbf{s}_1) = f_{\mathbf{Y}}(\mathbf{s}_2)$, that is, if $\mathbf{s}_1$ and $\mathbf{s}_2$ assign exactly the same values to state variables in $\mathbf{Y}$. Let $\mathbf{y}_\lambda$ denote the assignment to $\mathbf{Y}$ of states in block λ of the induced partition, that is, for each state $\mathbf{s} \in \lambda$, we have that $f_{\mathbf{Y}}(\mathbf{s}) = \mathbf{y}_\lambda$. Then for each pair of states $(\mathbf{s}_1, \mathbf{s}_2) \in S^2$, each action $a \in A$, and each block $\lambda \in \Lambda_{\mathbf{Y}}$, $[\mathbf{s}_1]_{\Lambda_{\mathbf{Y}}} = [\mathbf{s}_2]_{\Lambda_{\mathbf{Y}}}$ implies that

$$\begin{aligned}
\sum_{\mathbf{s}\in\lambda} P(\mathbf{s} \mid \mathbf{s}_1, a) &= \sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_1, a) P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_1, a) = \\
&= \sum_{\mathbf{s}\in\lambda} P(\mathbf{y}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), a) P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_1, a) = \\
&= P(\mathbf{y}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), a) \sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_1, a) = \\
&= P(\mathbf{y}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_2), a) \sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_2, a) = \\
&= \sum_{\mathbf{s}\in\lambda} P(\mathbf{y}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_2), a) P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_2, a) = \\
&= \sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_2, a) P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_2, a) = \sum_{\mathbf{s}\in\lambda} P(\mathbf{s} \mid \mathbf{s}_2, a).
\end{aligned}$$

The equality $\sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_1, a) = \sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}_2, a)$ follows from the fact that as we sum over states in λ, we go through every possible assignment of values to state variables in the set $\mathbf{S} - \mathbf{Y}$, so in fact, $\sum_{\mathbf{s}\in\lambda} P(f_{\mathbf{S}-\mathbf{Y}}(\mathbf{s}) \mid \mathbf{s}', a) = 1$ for each state $\mathbf{s}' \in S$. It follows that the partition $\Lambda_{\mathbf{Y}}$ induced by $f_{\mathbf{Y}}$ has the stochastic substitution property.

In general, the partition $\Lambda_{\mathbf{Y}}$ induced by $f_{\mathbf{Y}}$ is not reward respecting with respect to the expected reward function R of the original MDP. However, recall that the expected reward function R_o of option o is independent of the expected reward function R of the original MDP. To form a reduced option SMDP it is sufficient that the partition $\Lambda_{\mathbf{Y}}$ is reward respecting with respect to R_o. R_o is defined as -1 everywhere except when the process leaves the initiation set of option o. The initiation set of option o is determined by the state variables in $\mathbf{Z} \subseteq \mathbf{Y}$, so whether or not the process leaves the initiation set depends only on those state variables. It follows that $\Lambda_{\mathbf{Y}}$ is reward respecting with respect to R_o. ■

VISA goes a step further and forms the partition $\Lambda_{\mathbf{Z}}$ induced by the projection $f_{\mathbf{Z}}$. In other words, the option SMDP of option o ignores all state variables not in strongly connected components for which the value of at least one state variable appears in the context $\mathbf{c}$ associated with option o.

Theorem 2 *The projection function $f_{\mathbf{Z}}$ induces a partition $\Lambda_{\mathbf{Z}}$ of S that is reward respecting and has the stochastic substitution property if and only if for each pair of states $\mathbf{s}_1, \mathbf{s}_2 \in S^2$, each option $o' \in O_o$, and each block $\lambda \in \Lambda_{\mathbf{Z}}$, $[\mathbf{s}_1]_{\Lambda_{\mathbf{Z}}} = [\mathbf{s}_2]_{\Lambda_{\mathbf{Z}}}$ implies that $P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), o') = P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_2), o')$, where $\mathbf{z}_\lambda$ is the assignment of values to $\mathbf{Z}$ of states in block λ.*

Proof $\Lambda_{\mathbf{Z}}$ is still reward respecting with respect to R_o. However, a state variable in $\mathbf{Y}-\mathbf{Z}$ may influence the state variables in $\mathbf{Z}$, so $\Lambda_{\mathbf{Z}}$ does not always have the stochastic substitution property. We can write the sum $\sum_{\mathbf{s}\in\lambda} P_o(\mathbf{s} \mid \mathbf{s}_1, o')$ as

$$\begin{aligned}
\sum_{\mathbf{s}\in\lambda} P_o(\mathbf{s} \mid \mathbf{s}_1, o') &= \sum_{\mathbf{s}\in\lambda} P_o(f_{\mathbf{Z}}(\mathbf{s}) \mid \mathbf{s}_1, o') P_o(f_{\mathbf{S}-\mathbf{Z}}(\mathbf{s}) \mid \mathbf{s}_1, o') = \\
&= \sum_{\mathbf{s}\in\lambda} P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), o') P_o(f_{\mathbf{S}-\mathbf{Z}}(\mathbf{s}) \mid \mathbf{s}_1, o') = \\
&= P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), o') \sum_{\mathbf{s}\in\lambda} P_o(f_{\mathbf{S}-\mathbf{Z}}(\mathbf{s}) \mid \mathbf{s}_1, o') = \\
&= P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), o').
\end{aligned}$$

Using the same calculations, we obtain $\sum_{\mathbf{s}\in\lambda} P_o(\mathbf{s} \mid \mathbf{s}_2, o') = P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_2), o')$. $\Lambda_{\mathbf{Z}}$ has the stochastic substitution property if and only if for each pair of states $(\mathbf{s}_1, \mathbf{s}_2) \in S^2$, each option $o' \in O_o$, and each block $\lambda \in \Lambda_{\mathbf{Z}}$, $[\mathbf{s}_1]_{\Lambda_{\mathbf{Z}}} = [\mathbf{s}_2]_{\Lambda_{\mathbf{Z}}}$ implies that $P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_1), o') = P_o(\mathbf{z}_\lambda \mid f_{\mathbf{Y}}(\mathbf{s}_2), o')$, where $f_{\mathbf{Y}}(\mathbf{s}_1) = f_{\mathbf{Y}}(\mathbf{s}_2)$ does not hold in general. ■

From the work of Dean and Givan (1997) and Theorem 1 it follows that the partition $\Lambda_{\mathbf{Y}}$ induces a reduced SMDP that preserves optimality. Since the reduced SMDP can have far fewer state-action pairs than the original option SMDP, it can be significantly easier to solve, resulting in an important reduction in complexity. In addition, it follows from Theorem 2 that the partition $\Lambda_{\mathbf{Z}}$ induces a reduced SMDP that preserves optimality if and only if for each exit option $o' \in O_o$, state variables in $\mathbf{Y}-\mathbf{Z}$ do not influence the state variables in $\mathbf{Z}$ as a result of executing o'. Instead of solving the option SMDP directly, VISA solves the reduced SMDP induced by the partition $\Lambda_{\mathbf{Z}}$, which can have even fewer state-action pairs than the reduced SMDP induced by $\Lambda_{\mathbf{Y}}$.

Because of the way exits are defined, the exit options discovered by VISA often satisfy Theorem 2. For example, consider the exit option $\overline{H} \rightarrow H$ in the coffee task. After exit transformations, $\mathbf{Z} = \{S_{\mathtt{C}}\}$ and $\mathbf{Y} = \{S_{\mathtt{L}}, S_{\mathtt{C}}\}$, so $\mathbf{Y}-\mathbf{Z} = \{S_{\mathtt{L}}\}$. The options in the set O_o are $\overline{C} \rightarrow C$ and $C \rightarrow \overline{C}$, with associated exits $\langle (S_{\mathtt{L}} = L), \mathtt{BC} \rangle$ and $\langle (S_{\mathtt{L}} = \overline{L}), \mathtt{DC} \rangle$, respectively. As a result of executing action BC, the resulting value of state variable $S_{\mathtt{C}}$ depends on the previous value of state variable $S_{\mathtt{L}}$. However, as a result of executing the exit option $\overline{C} \rightarrow C$, the resulting value of $S_{\mathtt{C}}$ does not depend on the previous value of $S_{\mathtt{L}}$. Regardless of the previous value of $S_{\mathtt{L}}$, option $\overline{C} \rightarrow C$ always reaches the context $(S_{\mathtt{L}} = L)$ prior to executing BC, which causes the robot to buy coffee with non-zero probability. The same is true for exit option $C \rightarrow \overline{C}$, so it follows from Theorem 2 that the partition $\Lambda_{\mathbf{Z}}$ induced by $f_{\mathbf{Z}}$ has the stochastic substitution property.

If there exists a state variable in $\mathbf{Y}-\mathbf{Z}$ that influences a state variable in $\mathbf{Z}$, the partition $\Lambda_{\mathbf{Z}}$ does not have the stochastic substitution property. In other words, an optimal solution to the option SMDP $\mathcal{M}_o$ is not preserved in the reduced SMDP induced by the partition $\Lambda_{\mathbf{Z}}$. A solution to the reduced SMDP only corresponds to an approximate solution to $\mathcal{M}_o$. However, we believe that there is still a reason to perform state abstraction this way. The size of the partition $\Lambda_{\mathbf{Y}}$ may be exponentially larger than the size of $\Lambda_{\mathbf{Z}}$, so the difference in learning complexity may be significant in the two cases. We argue that the reduction in learning complexity often outweighs the loss of exact optimality.

To take even further advantage of structure, VISA stores the policies of options in the form of policy trees. 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. At each leaf of the policy tree, VISA stores action-values or, more accurately described, option-values, which indicate the utility of executing different options in states that map to that leaf. Recall that VISA maintains a transition graph, in the form of a tree, for each strongly connected component in the causal graph. The policy tree structure of an option can be constructed by merging the transition graph trees of strongly connected components that contain state variables whose values appear in the associated context. The policy tree structure induces a partition Λ_π such that $\Lambda_Z \leq \Lambda_\pi$, that is, Λ_π is guaranteed to have at most as many blocks as Λ_Z.

Another part of abstraction is reducing the number of options in the option set O_o of the option SMDP $\mathcal{M}_o$. If there are fewer options to select from, an autonomous agent can discover more quickly which option or options result in an optimal value for each block of the state partition. As we explained above, VISA finds strongly connected components that contain at least one state variable whose value appears in the context $\mathbf{c}$ associated with option o. The algorithm fills the option set O_o with options that change the values of state variables in those strongly connected components. The algorithm also includes options that leave the initiation sets of options in O_o. It is not necessary to include other options in O_o since they do not have any impact on reaching the context $\mathbf{c}$ associated with option o. Thus, VISA can reduce the number of options of each option SMDP, further reducing the complexity of learning.

4.6 Task Option

VISA also introduces an option, which we call the *task option*, associated with the reward node in the component graph of the task. The purpose of the task option is to approximate a solution to the original MDP. However, instead of being a policy that selects among primitive actions, the policy of the task option selects among the exit options introduced by VISA. Thus, the policy of the task option represents a hierarchical solution to the task that takes advantage of the exit options to set the values of relevant state variables in such a way as to maximize expected reward.

The task option is admissible everywhere, that is, its initiation set equals S. If the task is finite-horizon, the termination condition function β is defined such that the task option terminates whenever the task is completed. If the task is infinite-horizon, β is defined such that the task option never terminates. To learn the task option policy, VISA constructs the option SMDP corresponding to the task option using the same strategy it uses for the exit options. However, the expected reward function of the task option SMDP is equal to the expected reward function of the original MDP. For determining the policy of the task option, the expected reward for executing an exit option o is defined as the sum of discounted reward of the primitive actions selected during the execution of o.

VISA also performs state abstraction for the task option SMDP in the same way it does for exit options. First, VISA finds the set of state variables $\mathbf{Z} \subseteq \mathbf{V}$ in strongly connected components with edges to the expected reward node in the component graph. VISA performs state abstraction by ignoring the values of state variables not in $\mathbf{Z}$, and it constructs a policy tree structure to further reduce the number of states. In addition, the option set of the task option SMDP only includes exit options that change the values of state variables in $\mathbf{Z}$.

The task option is the only reward-dependent component of VISA. If several tasks share the same set of state variables and actions, the same set of exit options apply to all of these tasks. For example, this would apply to a workshop environment with a fixed number of objects where a robot may be instructed to perform several tasks, such as moving objects. VISA can construct a causal graph that is common to all tasks by excluding the expected reward node, and use the graph to

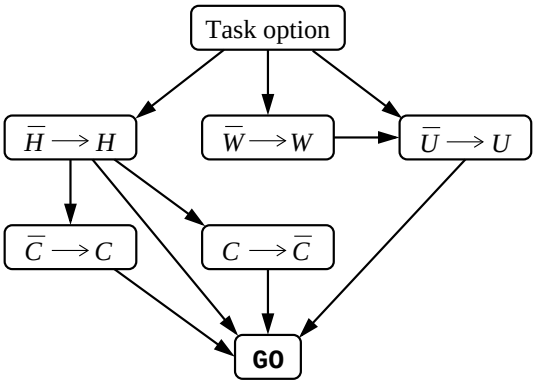

Figure 5: The hierarchy of options discovered by VISA in the coffee task

introduce exit options as before. When provided with the expected reward function of a specific task, VISA can construct the task option by overlaying the expected reward node onto the existing causal graph. This way, VISA just needs to learn the policies of the exit options once and can reuse them throughout all tasks. This facilitates transfer of knowledge between tasks in the same environment.

4.7 Option Hierarchy

The task option together with the exit options introduced by VISA implicitly define a hierarchy of options in which the options on one level selects options on the next lower level. Recall that for an option associated with a node in the component graph, the option SMDP only includes options that change state variables in components that have edges to that node. In other words, the component graph determines the structure of the option hierarchy. Since the component graph is guaranteed to contain no cycles, the option hierarchy is well-defined, and it is not possible for an option to execute itself, either directly or indirectly.

Figure 5 shows the hierarchy of options that VISA comes up with in the coffee task. The option hierarchy is determined by the component graph of the coffee task, illustrated in Figure 2. The task option always sits at the top level of the hierarchy. There are two components with edges to the expected reward node, namely $S_{\mathtt{H}}$ and $S_{\mathtt{W}}$. Option $\overline{H} \to H$ changes the value of $S_{\mathtt{H}}$, and option $\overline{W} \to W$ changes the value of $S_{\mathtt{W}}$. In addition, option $\overline{U} \to U$ causes the process to leave the initiation set of $\overline{W} \to W$. In other words, the task option selects among the three options $\overline{H} \to H$, $\overline{W} \to W$, and $\overline{U} \to U$.

In turn, there are two components with edges to the component $S_{\mathtt{H}}$, namely $S_{\mathtt{L}}$ and $S_{\mathtt{C}}$. The primitive action `GO` changes the value of $S_{\mathtt{L}}$, while options $\overline{C} \to C$ and $C \to \overline{C}$ change the value of $S_{\mathtt{L}}$. Consequently, option $\overline{H} \to H$ selects among `GO`, $\overline{C} \to C$, and $C \to \overline{C}$. There are also two components with edges to $S_{\mathtt{W}}$, namely $S_{\mathtt{U}}$ and $S_{\mathtt{R}}$. Option $\overline{U} \to U$ changes the value of $S_{\mathtt{U}}$, while no option changes the value of $S_{\mathtt{R}}$. Thus, $\overline{W} \to W$ can only select $\overline{U} \to U$. Finally, there are edges between $S_{\mathtt{L}}$ and $S_{\mathtt{C}}$ as well as between $S_{\mathtt{L}}$ and $S_{\mathtt{U}}$, so options $\overline{C} \to C$, $C \to \overline{C}$, and $\overline{U} \to U$ all select among the primitive action `GO` that changes the value of $S_{\mathtt{L}}$.

4.8 Merging Strongly Connected Components

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 strongly connected components that are linked by too many exits. After VISA identifies exits for a strongly connected component, the algorithm counts the number of exits identified. If the number of exits is larger than a threshold, VISA merges the strongly connected component with one or several of its parents. The merge operation places all state variables in the strongly connected components into a single component and recomputes the exits of the new component. As a result, the complexity of solving an associated subtask increases because there are more state variables in the set $\mathbf{Z}$. However, the number of subtasks decreases since there are fewer exits as a result.

4.9 Summary of the Algorithm

In summary, VISA first constructs the causal graph to determine how state variables are related. If there are cycles in the causal graph, it is not possible to decompose the task, so VISA gets rid of cycles by identifying the strongly connected components. For each strongly connected component, VISA uses the DBN model to identify exits, that is, pairs of variable values and actions that cause the value of some state variable in the component to change. For each exit, VISA constructs the components of an exit option, whose purpose it is to bring about the corresponding variable value change using a minimum number of options. At the top level, VISA constructs a task option that uses the exit options to approximate a solution to the original MDP. VISA uses reinforcement learning techniques to learn a policy of each option introduced. Algorithm 3 provides pseudo-code for VISA.

4.10 Limitations of the Algorithm

VISA only decomposes a task if there are two or more strongly connected components in the causal graph of the task. Otherwise, VISA cannot exploit conditional independence between state variables to identify options. Since the option SMDPs are stand-alone, the hierarchy discovered by VISA enables recursive optimality at best, as opposed to hierarchical optimality (Dietterich, 2000a). In addition, VISA works best when there are relatively few exits that cause the values of state variables in a strongly connected component to change.

Furthermore, the option-specific state abstraction performed by VISA is independent of the way options are formed. Given access to the causal graph, VISA makes it possible to efficiently perform state abstraction for any option whose goal is to reach a context specified by an assignment of values to a subset of the state variables. For the purpose of state abstraction, it does not matter how an autonomous agent determines that it is useful to reach that specific context. In other words, the state abstraction part of VISA could be combined with other techniques for discovering useful activities, as long as they are of the required form.

State abstraction for the task option is particularly efficient in tasks for which the expected reward depends on only a few state variables. If most state variables influence reward, learning the task option policy requires almost the same effort as learning a policy over primitive actions. Also note that exit options attempt to change the value of a state variable using as few primitive actions as possible. In terms of expected reward, such a behavior may not be optimal, since each action does not necessarily incur the same expected reward. In some tasks, it would be necessary to choose a

Algorithm 3 VISA

Input: DBN model of a factored MDP $\mathcal{M}$ with set of state variables **S**
construct the causal graph of the task
compute the strongly connected components of the causal graph
perform a topological sort of the strongly connected components
for each strongly connected component **SC** $\subseteq$ **S** in topological order
    identify exits that cause the values of state variables in **SC** to change
    while the number of exits exceeds a threshold
        merge **SC** with a parent strongly connected component
        label the new strongly connected component **SC** and recompute the exits
    for each exit $\langle \mathbf{c}, a \rangle$ of the strongly connected component **SC**
        perform any possible exit transformations
        compute the set **Z** of influencing state variables
        construct an initiation set I
        construct a termination function β using the context **c**
        construct a policy tree by merging transition graphs of parent components
        let S_o be the leaves of the policy tree
        let O_o be the set of options that changes values of state variables in **Z**
        let Ψ_o be defined by the initiation sets of options in O_o
        define R_o as -1 everywhere except when the context **c** is unreachable
        let P_o be undefined
        construct the option SMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi_o, P_o, R_o \rangle$
        construct an exit option $o = \langle I, \pi, \beta \rangle$, where $\pi =$ optimal policy of $\mathcal{M}_o$
    construct the transition graph of the strongly connected component **SC**
construct a task option corresponding to the original task
use reinforcement learning techniques to learn the policy of each option

different expected reward function for exit option SMDPs to avoid large negative rewards. However, VISA is most efficient in tasks for which few state variables influence reward. In such tasks, lower-level variables do not influence reward, so the optimal behavior is to achieve the precondition of an exit as quickly as possible. For example, in the coffee task, the option hierarchy discovered by VISA enables optimality, since none of the exit options choose suboptimal actions.

4.11 Experiments

We ran several experiments to test the performance of VISA. Since VISA uses the DBN model of factored MDPs, it would be unfair to compare it to algorithms that begin with less prior knowledge. Instead, we compared VISA to two algorithms that also assume knowledge of the DBN model: SPUDD (Hoey et al., 1999) and symbolic Real-Time Dynamic Programming, or sRTDP (Feng et al., 2003). SPUDD is a more efficient version of policy iteration that takes advantage of the compactness of the DBN model to compute the value function in the form of an algebraic decision diagram, or ADD.

sRTDP is an online planning algorithm that, at each step, constructs a set of states that are similar to the current state according to one of two heuristics, called value and reach. The algorithm uses the DBN model to determine the set of possible next states, and performs a masked backup of the value function restricted to the set of current and next states. The algorithm then selects for execution one of the actions whose current action-value estimate is highest. sRTDP stores the value function in the form of ADDs, and uses SPUDD to perform the masked value backup at each step. SPUDD includes a mechanism that limits the size of the ADDs, divides the state variables into subsets, and decomposes the value backup into several smaller computations. In our implementation, we did not allow the size of the ADDs to exceed 10,000 nodes.

We performed experiments with each algorithm in four tasks: the coffee task, the Taxi task, the Factory task (Hoey et al., 1999), and a simplified version of the autonomous guided vehicle (AGV) task of Ghavamzadeh and Mahadevan (2001). In the Taxi task (Dietterich, 2000a), a taxi has to pick up a passenger from their location and deliver them to their destination. The Taxi task has 600 states and 6 actions. In the Factory task (Hoey et al., 1999), a robot has to assemble a component made of two objects. Before assembly is possible, the robot has to perform various operations on each of the two objects, such as shaping, smoothing, polishing and painting. The objects can then be connected either by drilling and bolting or by gluing. The task is described by 17 binary variables, for a total of approximately 130,000 states, and the robot has 14 actions.

The Factory task was designed as a infinite-horizon task whose reward function assigns partial reward in many states. When the component has been assembled, the optimal policy repeatedly selects the same action in the same state for maximal reward. However, when using reinforcement learning to learn a policy, it is necessary to reset the state once in a while to ensure that a policy is learned for all states. When the state is reset, the positive reward as a result of assembling the component is not large enough to prevent the learning agent from exploiting the partial reward in other states. For this reason, we redefined the reward function of the Factory task to only assign positive reward when the component has been assembled. This neither affects the optimal policy nor the set of state variables that influence reward.

In the AGV task (Ghavamzadeh and Mahadevan, 2001), an autonomous guided vehicle (AGV) has to transport parts between machines in a manufacturing workshop. We simplified the task by reducing the number of machines from 4 to 2 and setting the processing time of machines to 0 to

make the task fully observable. The resulting task is illustrated in Figure 6 and has approximately 75,000 states. The goal of the AGV is to proceed to the load station, pick up a random part i, transport it to the drop-off location D_i of machine M_i, drop it off, then proceed to the pick-up location P_i of machine M_i, pick up the processed part, transport it to the warehouse, and finally drop it off. The AGV is restricted to move unidirectionally along the arrows in the figure, and has to ensure that at least one part of each type is stored in the warehouse. The set of state variables describing the task is $\mathbf{S} = \{S_x, S_y, S_f, S_h, S_{d1}, S_{p1}, S_{d2}, S_{p2}, S_{a1}, S_{a2}\}$, where S_x and S_y represent the location of the AGV, S_f the direction it is facing, S_h the part it is holding, S_{di} the number of parts at the drop-off location D_i of machine M_i, S_{pi} the number of parts at the pick-up location P_i, and S_{ai} whether a part of type i is present in the warehouse. The AGV has 6 actions: move in the direction it is facing, turn left or right, drop off a part, pick up a part, and idle. Even though we simplified the AGV task, its size still presents a challenge for algorithms that discover activities.

Each graph in the results illustrates the average reward over 100 learning runs with each algorithm. Since the algorithms are fundamentally different, we compared the actual running time in milliseconds. The graphs for VISA include 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 among primitive actions. We set the discount factor to $\gamma = 0.9$ and initially used a step-size parameter $\alpha = 0.05$, which we decayed at regular time intervals by multiplying the current step-size parameter by 0.9. The policies of all options, including the task option, were learned in parallel. Prior to executing, sRTDP computes action ADDs; the graphs include the time it takes to do this. We report results of both heuristics (value and reach) used by sRTDP to construct the set of similar states. All algorithms were coded in Java, except that the CUDD library (written in C) was used to manipulate ADDs through the Java Native Interface.

SPUDD is conceptually different from VISA and sRTDP in that it does not require actual experience in the domain. Instead, it uses the transition probabilities and expected reward of the DBN model to repeatedly update the policy off-line. Consequently, it is not possible to measure the reward received as a result of executing actions in the environment. To evaluate the running time of SPUDD, we first recorded the time elapsed between each iteration of the algorithm. After each iteration, we recorded the current policy and stored it in memory. When policy iteration converged, we retrieved each stored policy from memory. For each policy, we ran experiments in the domain and selected actions according the policy. The average reward of each experiment appears at the time at which the policy was recorded. Just as for VISA and sRTDP, we ran 100 trials and plotted the average reward across trials.

In the results, we also present a comparison of the size of the state partitions produced by SPUDD and VISA. The size of the state partition produced by SPUDD equals the number of leaves in the ADD used to represent the value function. In contrast, the size of the state partition produced by VISA equals the total number of leaves in the policy trees of exit options, including the task option. Since the state partition produced by VISA does not necessarily preserve optimality, it is often smaller than that of SPUDD.

4.12 Results

Figure 7 illustrates the results of the experiments in the coffee task. Since the coffee task is very small, all algorithms converge quickly to an optimal policy, although SPUDD has a slight edge over the others. The state partition produced by SPUDD contains 48 aggregated states, as compared to

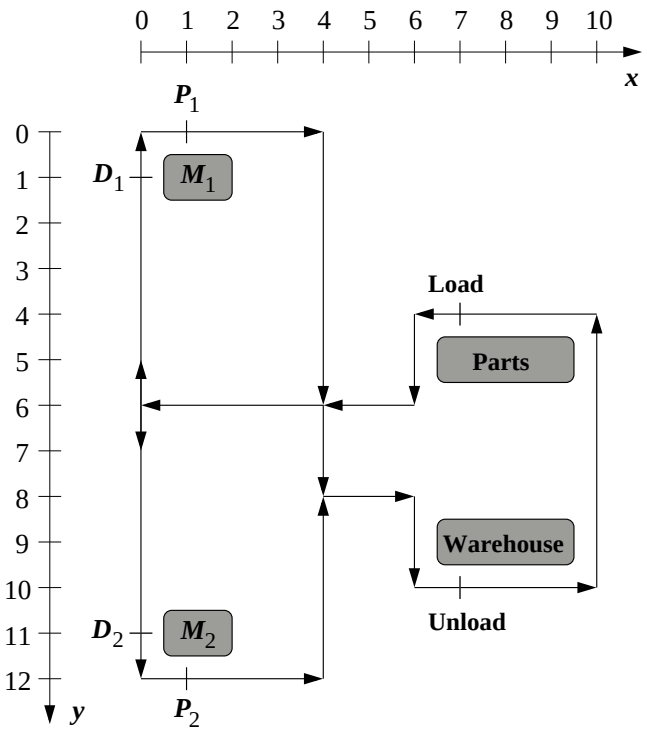

Figure 6: Illustration of the AGV task

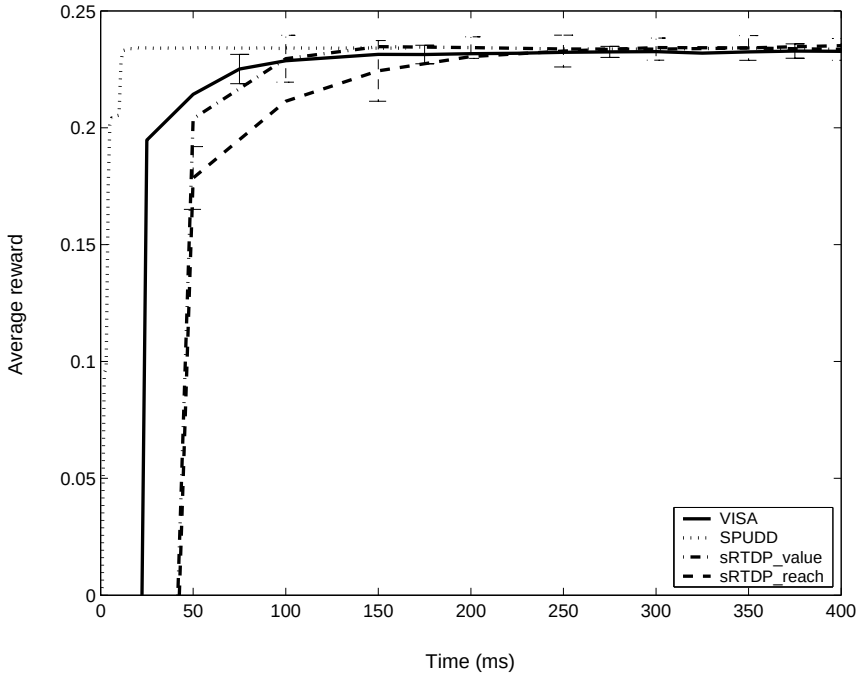

Figure 7: Results of learning in the coffee task

the $2^6 = 64$ total states of the task. In contrast, the state partition produced by VISA contains a total of 20 aggregated states.

Figure 8 illustrates the results of the experiments in the Taxi task. In this task, VISA and SPUDD perform significantly better than sRTDP, with SPUDD slightly faster than VISA. The state partition of SPUDD contains 525 aggregated states, almost as many as the 600 states of the original task. In comparison, the state partition produced by VISA contains a total of 106 aggregated states.

Figure 9 illustrates the results of the experiments in the Factory task. Again, SPUDD and VISA have a similar convergence times, although it appears as if VISA converges to a slightly suboptimal policy. The Factory task poses a significant challenge to online learning algorithms since a lot of actions undo the effect of other actions, making it difficult to achieve the objective. We believe this is the reason that sRTDP is struggling to converge quickly. The reason VISA does so well is that the hierarchical decomposition restricts the policy to select between options that achieve relevant subgoals, guiding the process towards the ultimate objective. The state partition of SPUDD contains 4,550 states, significantly less than the $2^{17} \approx 130{,}000$ states of the task. The state partition produced by VISA is even smaller, containing 2,620 aggregated states.

Figure 10 illustrates the results of the experiments in the AGV task of VISA and sRTDP using the reach heuristic. VISA decomposes the task in roughly 6 seconds and learning converges after 20 seconds. In comparison, it took SPUDD more than 4 minutes to converge to an optimal policy, and its performance is not shown in Figure 10. sRTDP using the reach heuristic completes the task a few times within the first minute of running time but convergence is much slower than for VISA. During our experiments, sRTDP using the value heuristic failed to complete the task even once within the

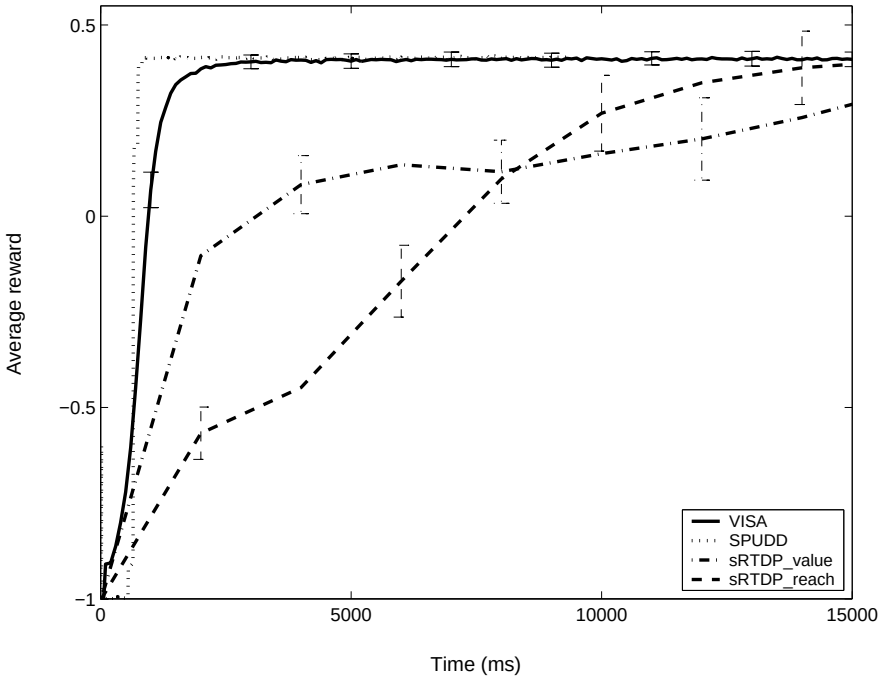

Figure 8: Results of learning in the Taxi task

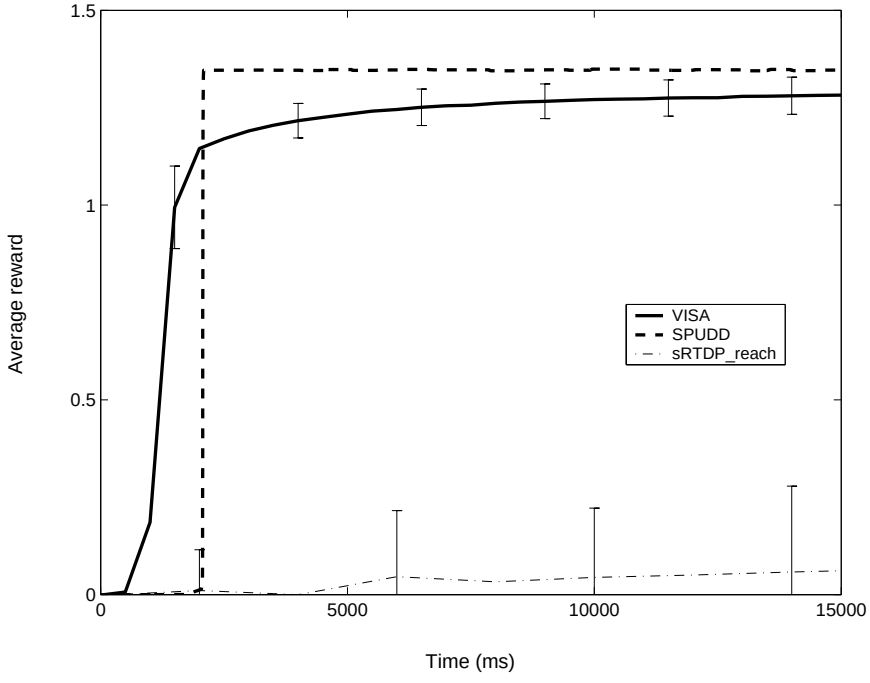

Figure 9: Results of learning in the Factory task

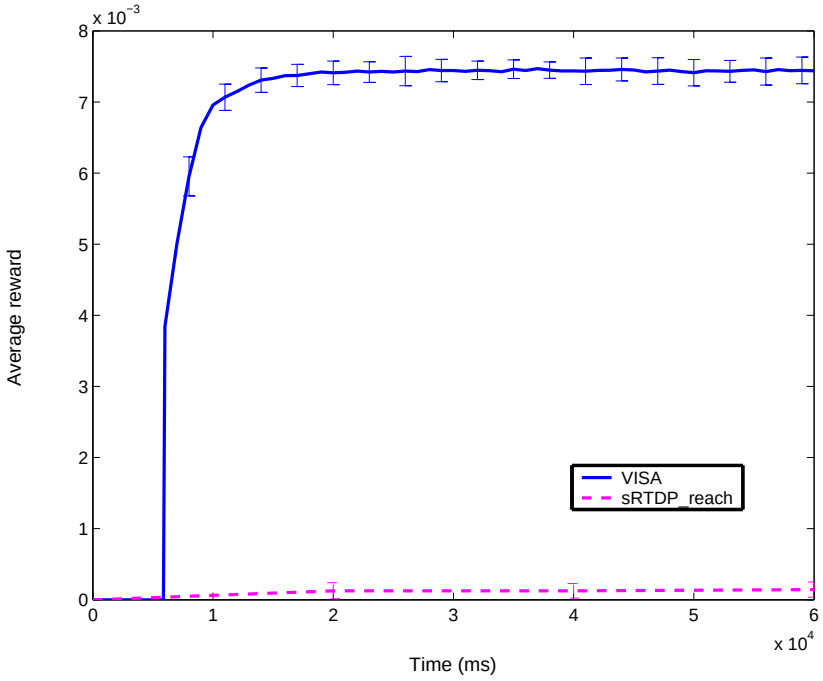

Figure 10: Results of learning in the AGV task

first 15 minutes. The state partition of SPUDD contains 11,096 states, compared to the 75,000 states of the task, while the state partition of VISA contains 5,996 aggregated states.

The results of the experiments illustrate the power of hierarchical decomposition when combined with option-specific abstraction. Even though SPUDD and sRTDP take advantage of task structure and are empirically faster than regular reinforcement learning algorithms, they still suffer from the curse of dimensionality as the size of the state space grows. On the other hand, VISA decomposes the original tasks into smaller, stand-alone tasks that are easier to solve without ever enumerating the state space. Instead, the complexity of the decomposition is polynomial in the size of the conditional probability trees of the DBN model. Each stand-alone task only distinguishes among values of a subset of the state variables, which means that the complexity of learning does not necessarily increase with the number of state variables. Evidently, the advantage offered by VISA varies between tasks and is dependent on the causal graph structure.

5. Constructing Compact Option Models

VISA computes option policies by first constructing the option SMDP, $\mathcal{M}_o$, of each exit option o. Since VISA does not have access to an estimate of the transition probability function, P_o, it cannot use a planning algorithm to solve the option SMDP. Instead, VISA uses SMDP Q-learning to learn the option policies, which does not require knowledge of transition probabilities as long as the algorithm has access to a real system and enough time. If VISA had access to DBN models that compactly describe the transition probabilities as a result of executing options, it would be possible to apply existing planning algorithms that exploit the DBN models to efficiently solve the

option SMDPs. Access to DBN models of options would open up new possibilities for learning and planning with options.

In this section, we develop ideas for constructing compact models of the exit options introduced by VISA. Current techniques for constructing option models require the state space to be enumerated. Since the goal of VISA is to reduce the complexity of learning by ignoring a subset of state variables, we want to avoid enumerating the state space. Instead, we define theoretical properties of partitions that preserve the transition probabilities and expected reward of options. We then discuss how to construct representations of transition probabilities and expected reward using partitions with these properties. In many cases, the partitions contain far fewer blocks than the number of states, resulting in a compact representation.

5.1 Multi-Time Option Models

Sutton et al. (1999) defined the multi-time model of an option $o = \langle I, \pi, \beta \rangle$ as

$$P(s' \mid s, o) = \sum_{t=1}^{\infty} \gamma^t P(s', t \mid s, o), \tag{2}$$

$$R(s, o) = E\{ \sum_{k=1}^{t} \gamma^{k-1} R(s_k, a_k) \mid s_1 = s \}, \tag{3}$$

where t is the random duration until o terminates and $P(s', t \mid s, o)$ is the probability that o terminates in state $s' \in S$ after t time steps when executed in state $s \in S$. The expectation in Equation 3 is taken over the distribution of state-action pairs (s_k, a_k), $k \in [1, t]$. This distribution is determined by the functions $P(s_{k+1} \mid s_k, a_k)$, $\pi(s_k, a_k)$, and $\beta(s_k)$. We refer to the terms $P(s' \mid s, o)$ as *discounted probabilities* since they do not sum to 1 for $\gamma < 1$. However, the multi-time model enables learning and planning with options as single units, which Sutton et al. (1999) call SMDP value learning and SMDP planning, respectively.

It is possible to use dynamic programming to compute the multi-time model in Equations 2 and 3. We can set up the Bellman form of the equations in which each term is a function of the terms at the next time step:

$$P(s' \mid s, o) = \gamma \sum_{a \in A} \pi(s, a) \left[P(s' \mid s, a) \beta(s') + \sum_{s'' \in S} P(s'' \mid s, a)(1 - \beta(s'')) P(s' \mid s'', o) \right], \tag{4}$$

$$R(s, o) = \sum_{a \in A} \pi(s, a) \left[R(s, a) + \gamma \sum_{s' \in S} P(s' \mid s, a)(1 - \beta(s')) R(s', o) \right]. \tag{5}$$

Let us label each state with a unique subscript $i \in \{1, \ldots, |S|\}$. Let P^a, $a \in A$, be the transition matrix for action a whose entry (i, j) is $P(s_j \mid s_i, a)$, and let P^o be the corresponding matrix for option o. Let Π^a, $a \in A$, be the diagonal matrix whose entry (i, i) is $\pi(s_i, a)$, and let B be the diagonal matrix whose entry (i, i) is $\beta(s_i)$. Let R^a, $a \in A$, be the vector whose ith entry is $R(s_i, a)$, and let R^o be the corresponding vector for option o. To avoid confusion with the option initiation set I, we use E to denote the identity matrix. Then we can write Equations 4 and 5 respectively in the following forms:

$$P^o = \gamma \sum_{a \in A} \Pi^a P^a (B + (E - B) P^o), \tag{6}$$

$$R^o = \sum_{a \in A} \Pi^a (R^a + \gamma P^a (E - B) R^o). \tag{7}$$

5.2 Multi-Time Models for Exit Options

Recall that an exit option o is associated with an exit $\langle \mathbf{c}, a \rangle$, composed of a context $\mathbf{c}$ and an action a. Unlike regular options, the exit option executes action a following termination in context $\mathbf{c}$. In addition, if o is executed in a state that already satisfies context $\mathbf{c}$, o immediately terminates and executes action a. As a consequence, it is necessary to modify the multi-time model to suit exit options. The multi-time model of an exit option o has the following form:

$$P^o = \gamma(BP^a + (E - B) \sum_{a' \in A} \Pi^{a'} P^{a'} P^o), \tag{8}$$

$$R^o = BR^a + (E - B) \sum_{a' \in A} \Pi^{a'} (R^{a'} + \gamma P^{a'} R^o), \tag{9}$$

where a is the action of the exit associated with o. The above definition assumes that o always terminates in a state that satisfies context $\mathbf{c}$, so that a is always executed following termination.

Because we modified the multi-time model to handle the case of exit options, we need to show that the discounted probabilities, P^o, and expected reward, R^o, associated with an exit option o are well-defined under the condition that o is guaranteed to eventually terminate.

Definition 3 *An option o is proper if for each state $s_i \in I$, o eventually terminates with probability 1 when executed in s_i.*

Definition 3 imposes a restriction on the policy π and termination condition function β of an option o.

Theorem 4 *For a proper option o, the systems of linear equations in Equations 8 and 9 are consistent and have unique solutions.*

The unknown quantities that we want to solve for are P^o and R^o. If we move the unknowns to the left-hand side of Equations 8 and 9 we obtain the following systems of equations:

$$\left[E - \gamma(E - B) \sum_{a' \in A} \Pi^{a'} P^{a'} \right] P^o = \gamma BP^a, \tag{10}$$

$$\left[E - \gamma(E - B) \sum_{a' \in A} \Pi^{a'} P^{a'} \right] R^o = BR^a + (E - B) \sum_{a' \in A} \Pi^{a'} R^{a'}. \tag{11}$$

Note that the unknown quantities P^o and R^o are multiplied by the same matrix $M = E - \gamma(E - B) \sum_{a' \in A} \Pi^{a'} P^{a'}$. The systems of linear equations in Equations 10 and 11 are consistent and have unique solutions if and only if matrix M is invertible, that is, if and only if the determinant of M is non-zero. The proof of Theorem 4 appears in Appendix A.

Since Equations 6 and 7 resemble the Bellman optimality equation in Equation 1, it is possible to use algorithms similar to value iteration and policy iteration to solve for P^o and R^o. Note, however, that we do not want to represent the matrices explicitly, since their size is proportional to the number of states. Instead, we can use decision trees or ADDs to compactly represent the matrices. The policy of an exit option is already in the form of a tree, and it is easy to construct a tree that represents the termination condition function β. SPUDD (Hoey et al., 1999) contains an efficient subroutine

that uses the DBN model to perform multiplication of a matrix with the transition probability matrix P^a without explicitly representing P^a. For a proper option o, iteratively performing the calculations on the right-hand side of Equations 8 and 9 will eventually converge to a compact representation of P^o and R^o.

5.3 Decomposition of the Option Model

In Section 4, we compared VISA with several algorithms that take advantage of the DBN model to compactly represent transition probabilities and expected reward. Even though these algorithms construct compact representations of the value function, VISA outperformed these algorithms in several tasks. The reason for this is that VISA introduces a set of subtasks and performs state abstraction for each subtask by ignoring irrelevant state variables, making each subtask easier to solve than the original task. It is possible to decompose computation of the multi-time option model in a similar way.

Recall that we approximate the transition probabilities of primitive actions as products of the conditional probabilities of each state variable $S_d \in \mathbf{S}$:

$$P(\mathbf{s}' \mid \mathbf{s}, a) \approx \prod_{S_d \in \mathbf{S}} P_d(f_{\{S_d\}}(\mathbf{s}') \mid f_{\mathbf{Pa}(S_d)}(\mathbf{s}), a).$$

The expression is an approximation for tasks in which there are dependencies between state variables at a same time step because our formalism does not account for such synchronous dependencies.

It is possible to approximate the terms of the multi-time model in a similar way. However, the multi-time model of an option o has two distributions that resemble transition probabilities: $P(\mathbf{s}' \mid \mathbf{s}, o)$, the discounted probability of transitioning from $\mathbf{s}$ to $\mathbf{s}'$ as a result of executing o; and $P(\mathbf{s}', t \mid \mathbf{s}, o)$, the exact probability of transitioning from $\mathbf{s}$ to $\mathbf{s}'$ in t time steps as a result of executing o. We can choose which of the two distributions to approximate.

If we choose to approximate $P(\mathbf{s}' \mid \mathbf{s}, o)$, we obtain the following approximation:

$$P(\mathbf{s}' \mid \mathbf{s}, o) \approx \prod_{S_d \in \mathbf{S}} P_d(f_{\{S_d\}}(\mathbf{s}') \mid f_{\mathbf{Pa}(S_d)}(\mathbf{s}), o).$$

Since we do not (yet) have access to a DBN model of option o, we assume that all state variables are parents of S_d, so $f_{\mathbf{Pa}(S_d)}(\mathbf{s}) = f_{\mathbf{S}}(\mathbf{s}) = \mathbf{s}$. We can compute the terms $P_d(f_{\{S_d\}}(\mathbf{s}') \mid \mathbf{s}, o)$ in the same way as the multi-time model:

$$P_d(f_{\{S_d\}}(\mathbf{s}') \mid \mathbf{s}, o) = \sum_{t=1}^{\infty} \gamma^t P_d(f_{\{S_d\}}(\mathbf{s}'), t \mid \mathbf{s}, o).$$

As a result, we obtain the following final approximation of Equation 2:

$$P(\mathbf{s}' \mid \mathbf{s}, o) \approx \prod_{S_d \in \mathbf{S}} \sum_{t=1}^{\infty} \gamma^t P_d(f_{\{S_d\}}(\mathbf{s}'), t \mid \mathbf{s}, o). \tag{12}$$

Equation 12 enables us to compute the conditional probabilities $P_d(v_d \mid \mathbf{s}, o)$ of the multi-time model separately for each state variable S_d. Here, $v_d \in \mathcal{D}(S_d)$ denotes one of the values of state variable S_d.

If we instead choose to approximate $P(\mathbf{s}', t \mid \mathbf{s}, o)$, we obtain the following alternative approximation of Equation (2):

$$P(\mathbf{s}' \mid \mathbf{s}, o) = \sum_{t=1}^{\infty} \gamma^t P(\mathbf{s}', t \mid \mathbf{s}, o) \approx \sum_{t=1}^{\infty} \prod_{S_d \in \mathbf{S}} \gamma^t P_d(f_{\{S_d\}}(\mathbf{s}'), t \mid \mathbf{s}, o). \tag{13}$$

Note that the difference between Equations 12 and 13 is the order of the summation and the product. As a result, Equation 12 assigns non-zero probability to events that could never occur, such as "the value of state variable $S_{\mathtt{L}}$ becomes L in 2 time steps, and the value of state variable $S_{\mathtt{W}}$ becomes W in 3 time steps." This event could never occur because an option cannot simultaneously terminate after 2 time steps and 3 time steps. In this sense, Equation 13 is a better approximation of $P(\mathbf{s}' \mid \mathbf{s}, o)$, but on the other hand, it does not enable us to compute a multi-time model of option o separately for each state variable. As we shall see, the ability to decompose the computation significantly reduces the complexity of computing the multi-time model. We believe that the reduction in complexity justifies the loss of accuracy, although we currently have no bounds on the approximation error. For this reason, we use Equation 12 as our approximation of Equation 2.

For each state variable S_d, each state $\mathbf{s} \in S$, and each value $v_d \in \mathcal{D}(S_d)$, we seek the term $P_d(v_d \mid \mathbf{s}, o)$ representing the probability of transitioning into a state that assigns v_d to S_d when o is executed in state $\mathbf{s}$. $P_d(v_d \mid \mathbf{s}, o)$ is given by the following equation:

$$P_d(v_d \mid \mathbf{s}, o) = \gamma(\beta(\mathbf{s}) P_d(v_d \mid \mathbf{s}, a) + (1 - \beta(\mathbf{s})) \sum_{a' \in A} \pi(\mathbf{s}, a') \sum_{\mathbf{s}' \in S} P(\mathbf{s}' \mid \mathbf{s}, a') P_d(v_d \mid \mathbf{s}', o)). \tag{14}$$

Let P_d^o be a $|S| \times |\mathcal{D}(S_d)|$ matrix whose entry (i, j) equals $P_d(j \mid \mathbf{s}_i, o)$. We can solve for P_d^o using the following system of equations:

$$P_d^o = \gamma(BP_d^a + (E - B) \sum_{a' \in A} \Pi^{a'} P^{a'} P_d^o), \tag{15}$$

where P_d^a is the equivalent of P_d^o for exit action a.

Lemma 5 *For a proper option o, the system of linear equations in Equation 15 is consistent and has a unique solution.*

Let us again move all unknowns to the left side of the equation to obtain

$$\left[E - \gamma(E - B) \sum_{a' \in A} \Pi^{a'} P^{a'} \right] P_d^o = \gamma B P_d^a. \tag{16}$$

The proof of Lemma 5 follows directly from the proof of Theorem 4 since the matrix $M = \left[E - \gamma(E - B) \sum_{a' \in A} \Pi^{a'} P^{a'} \right]$ that we need to invert to solve Equation 16 is the same as the matrix in Equations 10 and 11.

Instead of a single system of equations (8), we now have to solve one system of equations per state variable (15) to approximate the discounted probabilities P^o. Since the system of equations are similar, it appears as if little is gained, but as it turns out, the complexity of the computation may be dramatically lower. The matrix P_d^o only has one column per value in $\mathcal{D}(S_d)$, which is considerably less than the number of columns of P^o, even for a very compact state representation. This means that matrix multiplications can be carried out more efficiently. In addition, some state variables may be irrelevant for computing P_d^o, making the representation even more compact throughout the computation. Specifically, decomposing computation of the option model makes it possible to construct a different compact representation for each state variable.

5.4 Partitions

We formalize the ability to construct compact representations for each state variable using partitions. Recall that a partition Λ of the state set S that has the stochastic substitution property and is reward respecting induces a reduced MDP that preserves optimality. We define three more properties of partitions of S with respect to a factored MDP $\mathcal{M}$ and an option $o = \langle I, \pi, \beta \rangle$:

Definition 6 *A partition Λ of S is policy respecting if for each pair of states $(\mathbf{s}_i, \mathbf{s}_j) \in S^2$ and each action $a \in A$, $[\mathbf{s}_i]_\Lambda = [\mathbf{s}_j]_\Lambda$ implies that $\pi(\mathbf{s}_i, a) = \pi(\mathbf{s}_j, a)$.*

Definition 7 *A partition Λ of S is termination respecting if for each pair of states $(\mathbf{s}_i, \mathbf{s}_j) \in S^2$, $[\mathbf{s}_i]_\Lambda = [\mathbf{s}_j]_\Lambda$ implies that $\beta(\mathbf{s}_i) = \beta(\mathbf{s}_j)$.*

Definition 8 *A partition Λ of S is probability respecting of state variable S_d if for each pair of states $(\mathbf{s}_i, \mathbf{s}_j) \in S^2$, each action $a \in A$, and each value $v_d \in \mathcal{D}(S_d)$, $[\mathbf{s}_i]_\Lambda = [\mathbf{s}_j]_\Lambda$ implies that $P_d(v_d \mid \mathbf{s}_i, a) = P_d(v_d \mid \mathbf{s}_j, a)$.*

Using these definitions, it is possible to define partitions of S that preserve the multi-time model of an option o, which we prove in the following two theorems:

Theorem 9 *Let o be a proper option and let Λ_d be a partition of S that has the stochastic substitution property, is policy respecting, termination respecting, and probability respecting of S_d. Then for each pair of states $(\mathbf{s}_i, \mathbf{s}_j) \in S^2$ and each value $v_d \in \mathcal{D}(S_d)$, $[\mathbf{s}_i]_{\Lambda_d} = [\mathbf{s}_j]_{\Lambda_d}$ implies that $P_d(v_d \mid \mathbf{s}_i, o) = P_d(v_d \mid \mathbf{s}_j, o)$.*

The proof of Theorem 9 appears in Appendix B. As a consequence of Theorem 9, it is possible to ignore some state variables while computing the discounted probability model of an exit option. Take the example of computing the discounted probability $P^o_{\mathtt{W}}$ associated with state variable $S_{\mathtt{W}}$ and exit option $\overline{C} \to C$ in the coffee task. The policy and termination condition function of $\overline{C} \to C$ only distinguish between values of state variable $S_{\mathtt{L}}$, that is, any partition of S that distinguishes between values of $S_{\mathtt{L}}$ is policy respecting and termination respecting. The value of $S_{\mathtt{W}}$ as a result of executing any action is determined by the previous values of $S_{\mathtt{U}}$, $S_{\mathtt{R}}$, and $S_{\mathtt{W}}$, that is, any partition of S that distinguishes between values of $S_{\mathtt{U}}$, $S_{\mathtt{R}}$, and $S_{\mathtt{W}}$ is probability respecting of $S_{\mathtt{W}}$. From Theorem 1 we know that a partition of S has the stochastic substitution property and is reward respecting if it distinguishes between values of all state variables that influence relevant state variables. It follows from Theorem 9 that a partition of S that distinguishes between values of $S_{\mathtt{L}}$, $S_{\mathtt{U}}$, $S_{\mathtt{R}}$, and $S_{\mathtt{W}}$ preserves the discounted probability $P^o_{\mathtt{W}}$, that is, state variables $S_{\mathtt{C}}$ and $S_{\mathtt{H}}$ are irrelevant for computing $P^o_{\mathtt{W}}$.

Theorem 10 *Let o be a proper option and let Λ_R be a partition of S that has the stochastic substitution property, is reward respecting, policy respecting, and termination respecting. Then for each pair of states $(\mathbf{s}_i, \mathbf{s}_j) \in S^2$, $[\mathbf{s}_i]_{\Lambda_R} = [\mathbf{s}_j]_{\Lambda_R}$ implies that $R(\mathbf{s}_i, o) = R(\mathbf{s}_j, o)$.*

The proof of Theorem 10 appears in Appendix C. Usually, all state variables indirectly influence reward, so normally it is not possible to ignore the values of any state variables while computing the expected reward model R^o associated with exit option o.

5.5 Distribution Irrelevance

Dietterich (2000b) defined a condition that he calls result distribution irrelevance: a subset of the state variables may be irrelevant for the resulting distribution of a temporally-extended activity. This condition only exists in the undiscounted case, that is, for $\gamma = 1$. Otherwise, the time it takes the activity to terminate influences subsequent reward. We can take advantage of distribution irrelevance to compute the multi-time model of an exit option when $\gamma = 1$. Let o be the exit option associated with the exit $\langle \mathbf{c}, a \rangle$. Since o terminates in the context $\mathbf{c}$, we know the value of each state variable in the set $\mathbf{C} \subseteq \mathbf{S}$ immediately before action a is executed. In other words, the values of state variables in the set $\mathbf{C}$ prior to executing o are irrelevant for the resulting distribution of o.

Because of distribution irrelevance, we do not need to solve Equation 15 for state variables in the set $\mathbf{C}$. Instead, the conditional probabilities associated with state variable $S_d \in \mathbf{C}$ and option o are given by the conditional probabilities associated with S_d and the exit action a, restricted to states $s \in S$ such that $f_{\mathbf{C}}(s) = \mathbf{c}$. For example, as a result of executing the exit option associated with the exit $\langle (S_{\mathtt{L}} = L), \mathtt{BC} \rangle$ in the coffee task, the value of state variable $S_{\mathtt{L}}$ is L immediately before executing BC. Executing the exit action BC has no influence on the value of $S_{\mathtt{L}}$. As a result of executing the option that acquires coffee, the location of the robot is always the coffee shop, regardless of its previous location.

We can also simplify computation of conditional probabilities for state variables that are unaffected by actions that the policy selects. Let $\mathbf{U}^o \subseteq \mathbf{S}$ denote the subset of state variables whose values do not change as a result of executing any action selected by the policy π of exit option o. For the exit option o associated with exit $\langle (S_{\mathtt{L}} = L), \mathtt{BC} \rangle$ in the coffee task, $\mathbf{U}^o = \{S_{\mathtt{U}}, S_{\mathtt{R}}, S_{\mathtt{C}}, S_{\mathtt{H}}\}$, since the values of these state variables do not change as a result of executing GO, the only action selected by the policy of o. Thus, the conditional probabilities associated with state variables in the set $\mathbf{U}^o$ can be computed without solving Equation 15.

5.6 Summary of the Algorithm

In summary, to compute the multi-time model of an exit option one should first solve Equation 9 to compute the expected reward. In addition, for each state variable, one should solve Equation 15 to compute the discounted probability model associated with that state variable. If $\gamma = 1$ and it is possible to take advantage of distribution irrelevance, it is not necessary to solve Equation 15 for that state variable. The computation is most efficient if matrices are represented as trees or ADDs; in that case, the resulting models are also trees.

When we have computed the conditional probability tree associated with each state variable for an exit option, as well as a tree representing expected reward, we can construct a DBN for the option in the same way that we can for primitive actions. Figure 11 shows the DBN for the exit option associated with the exit $\langle (S_{\mathtt{L}} = L), \mathtt{BC} \rangle$ in the coffee task when $\gamma = 1$, taking advantage of distribution irrelevance. Note that there is no edge to state variable $S_{\mathtt{L}}$, which indicates that the resulting location does not depend on any of the state variables.

Since the DBN model of an option is in the same form as the DBN models of primitive actions, we can treat the option as a single unit and apply any of the algorithms that take advantage of compact representations. In addition, the DBN model makes it possible to apply our technique to nested options, that is, options selecting between other options. Once the policy of an option has been learned, we can construct its DBN model and use that model both to learn the policy of a higher-level option and later to construct a DBN model of the higher-level option.

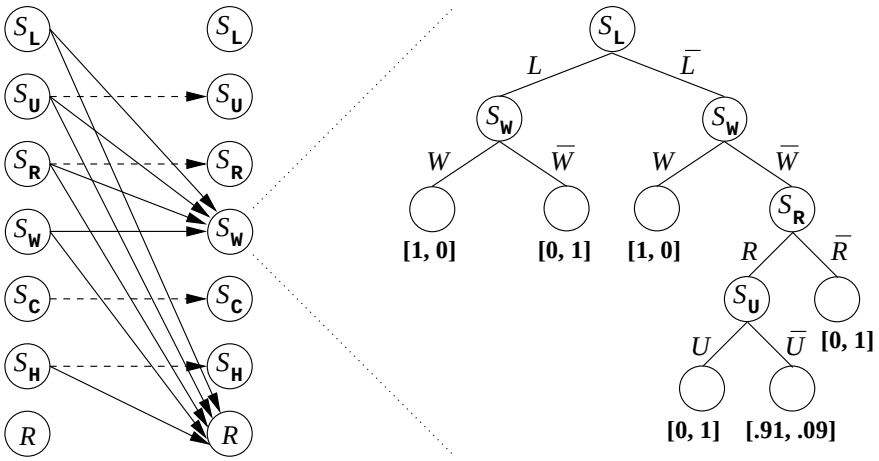

Figure 11: DBN for the option associated with $\langle (S_L = L), \texttt{BC} \rangle$

TASK	SPUDD	VISA	VISA-D
Coffee	75 ± 3	100 ± 33	18 ± 14
Taxi	$4{,}965 \pm 20$	$2{,}220{,}102 \pm 1{,}861$	$7{,}465 \pm 139$

Table 2: Comparison of time (ms) to convergence in two tasks

5.7 Experimental Results

We conducted a set of experiments to test the complexity of computing multi-time models for exit options. First, we ran VISA on the coffee task and used Equations 8 and 9 to compute the multi-time model of each exit option introduced. For each exit option, including the task option at the top level, we used SPUDD to compute an optimal policy. In a second experiment, we ran VISA again, but this time used Equation 15 to compute a separate transition probability model for each state variable. We set $\gamma = 1$ and used distribution irrelevance whenever possible to simplify computation of the transition probability model. We repeated these experiments in the Taxi task. For comparison, we also computed the time it takes SPUDD to converge in these two tasks.

Table 2 presents the results of the experiments, averaged over 100 trials. VISA-D denotes the VISA algorithm with the decomposed transition probability model. In all cases, the algorithms converged to an optimal policy for the task. Since the coffee task is very small, all algorithms converged relatively quickly. However, note that in the Taxi task, the convergence time of VISA is orders of magnitudes larger than that of SPUDD and VISA-D, while the convergence time of VISA-D is almost on par with that of SPUDD, even though it includes the time it took to compute the compact option models. Evidently, distribution irrelevance and simplified computation of the transition probability models can have a huge impact on the complexity of computing multi-time models for exit options.

6. Related Work

There exist several algorithms that decompose tasks into a hierarchy of activities. We have already mentioned the HEX-Q algorithm (Hengst, 2002) and its relation to our work. Dean and Lin (1995) used a fixed partition of the state space to decompose a factored MDP into regions. The authors developed an algorithm for solving the decomposed task by constructing activities for moving between regions. At the top level, the algorithm forms an abstract MDP with the regions as states and the activities as actions to approximate a global solution. Hauskrecht et al. (1998) extended this idea by suggesting several ways of constructing the set of activities given the decomposition. Most of their techniques rely on partial knowledge of the value function at different states to decide which activities to introduce. These techniques rely on prior knowledge of a useful partition, while our algorithm relies on the DBN model to decompose a task.

Nested Q-learning (Digney, 1996) introduces an activity for each value of each state variable. The goal of each activity is to reach the context described by the single state variable value. McGovern and Barto (2001) use diverse density to locate bottlenecks in successful solution paths, and introduce activities that reach these bottlenecks. Şimşek and Barto (2004) measure the relative novelty of each visited state, and introduce activities that reach states whose relative novelty exceeds a threshold value. Recent work on intrinsic motivation (Singh et al., 2005) tracks salient changes in variable values and introduces activities that cause salient changes to occur.

Other researchers use graph-theoretic approaches to decompose tasks. Menache et al. (2002) construct a state transition graph and introduce activities that reach states on the border of strongly connected regions of the graph. The authors use a max-flow/min-cut algorithm to identify border states in the transition graph. Mannor et al. (2004) use a clustering algorithm to partition the state space into different regions and introduce activities for moving between regions. Şimşek et al. (2005) identify subgoals by partitioning local state transition graphs that represent only the most recently recorded experience.

Another approach is to track learning in several related tasks and identify activities that are useful across tasks. SKILLS (Thrun and Schwartz, 1996) identifies activities that minimize a function of the performance loss induced by the resulting hierarchy and the total description length of all actions. PolicyBlocks (Pickett and Barto, 2002) identifies regions in the state space for which the policy is identical across tasks, and introduces activities that represent the policy of each region. Each activity is only admissible within its region of the state space.

Helmert (2004) developed an algorithm that constructs a causal graph similar to that of VISA and uses the graph to decompose deterministic planning tasks. The algorithm assumes a STRIPS formulation of actions (Fikes and Nilsson, 1971), which is similar to the DBN model of factored MDPs. Just like the DBN model, the STRIPS formulation expresses actions in terms of causes and effects on the state variables, except that the causes and effects are deterministic. Helmert (2004) uses the STRIPS action formulation to construct a causal graph in a special class of deterministic tasks in which the causal graph has one absorbing state variable with edges from each of the other state variables. The author shows that his algorithm efficiently solves a set of standard planning tasks using activities to represent the stand-alone tasks of the resulting decomposition.

There are several efficient algorithms for solving factored MDPs that use the DBN model to compactly describe transition probabilities and expected reward. Structured Policy Iteration, or SPI (Boutilier et al., 1995), stores the policy and value function in the form of trees. The algorithm performs policy iteration by intermittently updating the policy and value function, possibly changing

the structure of the trees in the process. Hoey et al. (1999) modified SPI to include algebraic decision diagrams, or ADDs, which store conditional probabilities more compactly than trees. Symbolic Real-Time Dynamic Programming, or sRTDP (Feng et al., 2003), also assumes that the conditional probabilities of the DBN model are stored using ADDs. sRTDP is an extension of Real-Time Dynamic Programming, or RTDP (Barto et al., 1995), that clusters states into abstract states based on two criteria, and performs an efficient backup of the value of the current abstract state following each execution of an action in the environment.

The DBN-E^3 algorithm (Kearns and Koller, 1999) is based on the assumption that there exists an approximate planning algorithm for the task, and that the structure of the DBN model is given. Using the planning procedure as a subroutine, the algorithm explores the state space and fills in the parameters of the DBN model. The running time of the algorithm is polynomial in the number of parameters of the DBN model, generally much smaller than the number of states. Guestrin et al. (2001) developed an algorithm based on linear programming that combines the DBN model with max-norm projections to solve factored MDPs. The algorithm is based on the assumption that there is a set of basis functions for representing the value function. It is guaranteed to converge to an approximately optimal solution.

Sutton et al. (1999) developed the multi-time model of options that we used to represent the effect of activities. The multi-time model includes an estimate of the transition probabilities and expected reward of options. Using the multi-time model of an option, it is possible to treat the option as a single unit during learning and planning. SMDP value learning (Sutton et al., 1999) uses the multi-time model to learn values or action-values in an SMDP. SMDP planning (Sutton et al., 1999) uses the multi-time model to perform planning in an SMDP, similar to policy iteration.

7. Conclusion

We presented Variable Influence Structure Analysis, or VISA, an algorithm that decomposes factored MDPs into hierarchies of options. VISA uses a DBN model of the factored MDP to construct a causal graph describing how state variables are related. The algorithm then searches in the conditional probability trees of the DBN model for exits, that is, combinations of state variable values and actions that cause the values of other state variables to change. VISA introduces an option for each exit and uses sophisticated techniques to construct the components of each option. The result is a hierarchy of options in which the policy of an option selects among options at a lower level in the hierarchy. Experimental results in a series of tasks show that the performance of VISA is comparable to that of state-of-the-art algorithms that exploit the DBN model, and in one task (AGV) VISA significantly outperforms the other algorithms.

VISA is based on the assumption that the values of key state variables change relatively infrequently. This is the same assumption made by Hengst (2002), Helmert (2004), and Singh et al. (2005). Just like the HEX-Q algorithm (Hengst, 2002), VISA decomposes a task into activities by detecting the combinations of state variable values and actions that cause key variable value changes. However, as we already discussed, VISA uses the causal graph to represent how state variables are related, which is a more realistic model than that used by HEX-Q. Unlike the work of Helmert (2004), VISA can handle any configuration of the causal graph.

Many existing algorithms need to accumulate extensive experience in the environment to decompose a task into activities, and they usually store quantities for each state. Assuming that the DBN model is given, VISA does not need to accumulate experience in the environment to perform

the decomposition. In addition, VISA only stores quantities proportional to the size of the conditional probability trees of the DBN model. Although we do not provide any comparisons, it is likely that VISA uses less memory and performs decomposition of a task in less time than these other algorithms.

Our second algorithm is a method for computing compact models of the options discovered by VISA. Existing methods for computing compact option models do not scale well to large tasks. For this reason, the first implementation of VISA uses reinforcement learning to approximate an optimal policy of each option. If VISA had access to compact option models, it could use dynamic programming techniques to compute the option policies without interacting with the environment. Our algorithm constructs partitions with certain properties to reduce the complexity of computing compact option models. The algorithm computes a DBN model for each option identical to the DBN model for primitive actions. This makes it possible to apply existing algorithms that use the DBN model to efficiently approximate option policies.

For VISA to successfully decompose a task, the causal graph needs to contain at least two separate strongly connected components. In tasks for which each state variable indirectly influences each other state variable, decomposition using this strategy is not possible. In other words, VISA is limited to function well in tasks with relatively sparse relationships between state variables. We believe that a non-trivial number of realistic tasks fall within this category. For example, in most navigation tasks, location influences the value of variables representing stationary objects, which in turn have no impact on location. Moreover, constructing the causal graph is polynomial in the size of the DBNs, so it is relatively inexpensive to test whether or not VISA can successfully decompose a task.

7.1 Future Work

Hoey et al. (1999) pioneered the use of algebraic decision diagrams, or ADDs, to store the conditional probability distributions of the DBN model. Since ADDs are a more compact representation than trees, they require less memory. More importantly, several operations can be executed faster on ADDs than on trees. Although VISA uses trees to represent the conditional probability distributions, it would be relatively straightforward to change the representation to ADDs. Possibly, this modification could speed up decomposition and construction of compact option models.

It is also possible to combine VISA with other techniques that facilitate scaling. For example, once VISA has decomposed a task into options, we can apply reinforcement learning with function approximation to learn the option policies. Another possibility is to use existing algorithms to detect bottlenecks in the transition graph of a strongly connected component in the causal graph. This would enable further decomposition of the option SMDPs into even smaller subtasks.

Recall that VISA performs state abstraction for an option SMDP by constructing the partition $\Lambda_{\mathbf{Z}}$, where $\mathbf{Z} \subseteq \mathbf{S}$ is the set of state variables in strongly connected components whose variable values appear in the context of the associated exit. As a result of state abstraction, the option policy may be suboptimal. The problem occurs when an option selected by the option policy changes the value of a state variable not in $\mathbf{Z}$ that indirectly influences state variables in $\mathbf{Z}$. This problem would be alleviated if we merge strongly connected components whose state variables are affected by the same actions. The resulting decomposition would be less efficient in terms of learning complexity but would guarantee recursive optimality.

Our formal analysis of constructing compact option models requires partitions with a set of established properties. The requirement that the partitions should have all of these properties is quite strong. A possible line of future research is to relax or approximate the required properties of partitions, which could lead to even more efficient computation of option models, albeit with some loss of accuracy. An analysis of the resulting approximation could help determine a tradeoff between the complexity of computing compact option models and the accuracy of the resulting model.

We also made a strong independence assumption in order to reduce the complexity of computing a compact option model. Our algorithm assumes that the value of a state variable that results from executing an option is independent of the resulting values of other state variables. Since an option takes variable time to execute, the option passes through many states during execution. The independence assumption only holds if the resulting values of state variables are independent regardless of which state the option is currently in. In many cases, our independence assumption induces an approximation error. If possible, we would like to establish bounds on this approximation error to analyze the accuracy of our algorithm.

It is unrealistic to assume that a DBN model is always given prior to learning. If no DBN model is available, it is necessary to learn a DBN model from experience prior to executing VISA. There exist algorithms for active learning of Bayesian networks that can be applied to factored MDPs (Murphy, 2001; Steck and Jaakkola, 2002; Tong and Koller, 2001). However, these algorithms assume that it is possible to sample the MDP at arbitrary states. If we assume that it is only possible to sample the MDP along trajectories, it becomes necessary to develop novel algorithms for active learning of DBN models. Such algorithms would select actions with the goal of learning a DBN model describing the effect of actions on the state variables as quickly as possible.

Acknowledgments

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.

Appendix A. Proof of Theorem 4

In this appendix we prove Theorem 4 from Section 5. Equations 10 and 11 are consistent and have unique solutions if and only if the matrix $M = E - \gamma(E - B)\sum_{a\in A}\Pi^a P^a$ is invertible, that is, if $\det(M) \neq 0$. Each element of P^a is in the range $[0,1]$, and each row of P^a sums to 1. Because of the properties of π, it follows that $\sum_{a\in A}\Pi^a = E$ and that $\sum_{a\in A}\Pi^a P^a$ has the same properties as P^a. $(E - B)$ is a diagonal matrix whose elements are in the range $[0,1]$. Then $\gamma(E - B)\sum_{a\in A}\Pi^a P^a$ is a matrix such that each element is in the range $[0,1]$ and such that the sum of each row is in the range $[0,1]$. In other words, M has the following properties, where $n = |S|$:

1. for each $i = 1,\ldots,n$: $0 \leq m_{ii} \leq 1$,
2. for each $i = 1,\ldots,n$, $j \neq i$: $-m_{ii} \leq m_{ij} \leq 0$,
3. for each $i = 1,\ldots,n$: $0 \leq \sum_{j=1}^{n} m_{ij} \leq m_{ii}$.

Lemma 11 *An element m_{ii} on the diagonal of M equals 0 if and only if*

1. $\gamma = 1$,
2. $\beta(s_i) = 0$,
3. *for each action $a \in A$ such that $\pi(s_i, a) > 0$, $P(s_i \mid s_i, a) = 1$.*

Proof $m_{ii} = 1 - \gamma(1 - \beta(s_i)) \sum_{a \in A} \pi(s_i, a) P(s_i \mid s_i, a)$. The only solution to $m_{ii} = 0$ is $\gamma = 1$, $\beta(s_i) = 0$, and $P(s_i \mid s_i, a) = 1$ for each action $a \in A$ such that $\pi(s_i, a) > 0$. ■

An option is proper if and only if there is no set of absorbing states S' such that $\beta(s) = 0$ for each state $s \in S'$. A set of states S' is absorbing if and only if the probability of transitioning from any state in S' to any state outside S' is 0. A special case occurs when S' contains a single state s_i such that $\beta(s_i) = 0$ and such that $P(s_i \mid s_i, a)$ for each action $a \in A$ such that $\pi(s_i, a) > 0$. From Lemma 11 it follows that an element m_{ii} on the diagonal of M equals 0 if and only if s_i is an absorbing state such that $\beta(s_i) = 0$. Since no such state exists for a proper option o, we conclude that all elements on the diagonal of M are larger than 0 for a proper option o. Then it is possible to multiply each row of M by its diagonal element $1/m_{ii}$ to obtain a matrix A with the following properties:

1. for each $i = 1, \ldots, n$: $a_{ii} = 1$,
2. for each $i = 1, \ldots, n$, $j \neq i$: $-1 \leq a_{ij} \leq 0$,
3. for each $i = 1, \ldots, n$: $0 \leq \sum_{j=1}^{n} a_{ij} \leq 1$.

Since matrix A is obtained by multiplying each row of M by a scalar, the determinant of M equals 0 if and only if the determinant of A equals 0. We can write A as

$$A = \begin{pmatrix} 1 & a_{12} & \cdots & a_{1n} \\ a_{21} & 1 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 1 \end{pmatrix} = \begin{pmatrix} - & \mathbf{r}_1 & - \\ - & \mathbf{r}_2 & - \\ & \vdots & \\ - & \mathbf{r}_n & - \end{pmatrix},$$

where $\mathbf{r}_i$ is the ith row of A. It is possible to eliminate an element a_{ij}, $j < i$, by subtracting $a_{ij}\mathbf{r}_j$ from row $\mathbf{r}_i$:

$$\mathbf{r}_i - a_{ij}\mathbf{r}_j = \begin{pmatrix} a_{i1} - a_{ij}a_{j1} & \cdots & a_{ij} - a_{ij} \cdot 1 & \cdots & 1 - a_{ij}a_{ji} & \cdots & a_{in} - a_{ij}a_{jn} \end{pmatrix}.$$

Lemma 12 $0 \leq 1 - a_{ij}a_{ji} \leq 1$, *and* $1 - a_{ij}a_{ji} = 0$ *if and only if* $a_{ij} = a_{ji} = -1$.

Proof Follows immediately from the properties of A. ■

Lemma 13 *If* $1 - a_{ij}a_{ji} > 0$, *elimination of* a_{ij} *preserves the properties of* A.

Proof Since $1 - a_{ij}a_{ji} > 0$, we can multiply $\mathbf{r}_i - a_{ij}\mathbf{r}_j$ by $1/(1 - a_{ij}a_{ji})$:

$$\bar{\mathbf{r}}_i = \frac{1}{1 - a_{ij}a_{ji}} [\mathbf{r}_i - a_{ij}\mathbf{r}_j] = \begin{pmatrix} \frac{a_{i1} - a_{ij}a_{j1}}{1 - a_{ij}a_{ji}} & \cdots & 0 & \cdots & 1 & \cdots & \frac{a_{in} - a_{ij}a_{jn}}{1 - a_{ij}a_{ji}} \end{pmatrix}.$$

It follows immediately that element i of row $\bar{\mathbf{r}}_i$ equals $(1-a_{ij}a_{ji})/(1-a_{ij}a_{ji}) = 1$ and that element j equals $(a_{ij}-a_{ij}\cdot 1)/(1-a_{ij}a_{ji}) = 0$. For each $k = 1,\ldots,n$, $k \neq i, j$, compute bounds on element k of $\bar{\mathbf{r}}_i$:

$$\begin{aligned}
\frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} &\leq \frac{a_{ik}-0}{1-a_{ij}a_{ji}} \leq \frac{0-0}{1-a_{ij}a_{ji}} = 0, \\
\frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} &= \frac{1-a_{ij}a_{ji}+a_{ik}-a_{ij}a_{jk}-(1-a_{ij}a_{ji})}{1-a_{ij}a_{ji}} = \\
&= \frac{1+a_{ik}-a_{ij}(a_{ji}+a_{jk})}{1-a_{ij}a_{ji}} - 1 \geq \\
&\geq \frac{1+a_{ik}+a_{ij}}{1-a_{ij}a_{ji}} - 1 \geq \frac{1-1}{1-a_{ij}a_{ji}} - 1 = -1.
\end{aligned}$$

Also compute bounds on the sum of the elements of $\bar{\mathbf{r}}_i$:

$$\begin{aligned}
\sum_{k=1}^{n} \frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} &= \sum_{k=1}^{j-1} \frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} + \frac{a_{ij}-a_{ij}\cdot 1}{1-a_{ij}a_{ji}} + \sum_{k=j+1}^{i-1} \frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} + \\
&+ \frac{1-a_{ij}a_{ji}}{1-a_{ij}a_{ji}} + \sum_{k=i+1}^{n} \frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} \leq \\
&\leq 0+0+0+1+0 = 1, \\
\sum_{k=1}^{n} \frac{a_{ik}-a_{ij}a_{jk}}{1-a_{ij}a_{ji}} &= \frac{1}{1-a_{ij}a_{ji}} \left[\sum_{k=1}^{n} a_{ik} - a_{ij} \sum_{k=1}^{n} a_{jk} \right] \geq \frac{0+0}{1-a_{ij}a_{ji}} = 0.
\end{aligned}$$

It follows that row $\bar{\mathbf{r}}_i$ satisfies the properties of A. ■

From Lemma 12 and Lemma 13 it follows that the properties of A are preserved under elimination unless the element on the diagonal equals 0. We can compute the determinant of A by repeatedly performing elimination until A is an upper triangular matrix. If any element on the diagonal becomes 0 during elimination, $\det(A) = 0$. Otherwise, the determinant of A equals the inverse of the product of the coefficients by which we multiplied rows during elimination. Since each coefficient is larger than 0, it follows that $\det(A) > 0$.

Lemma 14 *Let $C = \{c_1,\ldots,c_m\}$ be a set of m indices, and let $S(C,\mathbf{r}_i) = \sum_{k=1}^{m} a_{ic_k}$ be the sum of elements of row $\mathbf{r}_i$ whose column indices are elements of C. Assume that $i \in C$ and that $S(C,\bar{\mathbf{r}}_i) = 0$ after elimination of an element a_{ij}, $j < i$. Then $S(C\cup\{j\},\mathbf{r}_i) = 0$ and $S(C\cup\{j\},\mathbf{r}_j) = 0$ prior to elimination of a_{ij}.*

Proof When we eliminate an element a_{ij}, $j < i$, the sum of elements of row $\bar{\mathbf{r}}_i$ whose column indices are elements of C is

$$\begin{aligned}
S(C,\bar{\mathbf{r}}_i) &= S(C,\bar{\mathbf{r}}_i) + 0 = \\
&= \sum_{k=1}^{m} \frac{a_{ic_k}-a_{ij}a_{jc_k}}{1-a_{ij}a_{ji}} + \frac{a_{ij}-a_{ij}\cdot 1}{1-a_{ij}a_{ji}} = \\
&= \frac{1}{1-a_{ij}a_{ji}} \left[\left(\sum_{k=1}^{m} a_{ic_k} + a_{ij} \right) - a_{ij} \left(\sum_{k=1}^{m} a_{jc_k} + 1 \right) \right].
\end{aligned}$$

Since i is one of the indices in C, it follows from the properties of A that $S(C,\bar{\mathbf{r}}_i) = 0$ if and only if $\sum_{k=1}^{m} a_{ic_k} + a_{ij} = 0$ and either $a_{ij} = 0$ or $\sum_{k=1}^{m} a_{jc_k} + 1 = 0$. If $a_{ij} = 0$, there was no reason to perform elimination, so it follows that $S(C \cup \{j\}, \mathbf{r}_i) = \sum_{k=1}^{m} a_{ic_k} + a_{ij} = 0$ and that $S(C \cup \{j\}, \mathbf{r}_j) = \sum_{k=1}^{m} a_{jc_k} + 1 = 0$. ■

Lemma 15 *If $S(C,\mathbf{r}_k) = 0$ for each row $\mathbf{r}_k$, $k \in C$, after elimination of an element a_{ij}, $j \notin C$, in row $\mathbf{r}_i$, $i \in C$, it follows that $S(C \cup \{j\}, \mathbf{r}_k) = 0$ for each row $\mathbf{r}_k$, $k \in C \cup \{j\}$ prior to elimination of a_{ij}.*

Proof If $S(C,\mathbf{r}_i) = 0$ following elimination of a_{ij}, it follows from Lemma 14 that $S(C \cup \{j\}, \mathbf{r}_i) = 0$ and that $S(C \cup \{j\}, \mathbf{r}_j) = 0$ prior to elimination of a_{ij}. For $k \in C - \{i\}$, $S(C \cup \{j\}, \mathbf{r}_k) = S(C,\mathbf{r}_k) + a_{kj} = a_{kj}$. Since $a_{kj} \leq 0$ and $S(C \cup \{j\}, \mathbf{r}_k) \geq 0$, it follows that $S(C \cup \{j\}, \mathbf{r}_k) = a_{kj} = 0$. ■

Lemma 16 *If* $\det(A) = 0$, *it is possible to rearrange the rows and columns of A to obtain*

$$\begin{pmatrix} X & 0 \\ Y & Z \end{pmatrix},$$

where X is a $k \times k$ matrix such that for each $i = 1,\ldots,k$, $\sum_{j=1}^{k} x_{ij} = 0$.

Proof If $\det(A) = 0$, there exists $i, j < i$ such that a_{ii} becomes 0 during elimination of a_{ij}. From Lemma 12 it follows that $a_{ij} = a_{ji} = -1$ prior to elimination of a_{ij}, so $S(\{i,j\}, \mathbf{r}_i) = a_{ij} + a_{ii} = -1 + 1 = 0$ and $S(\{i,j\}, \mathbf{r}_j) = a_{jj} + a_{ji} = 1 - 1 = 0$. Let $C = \{i,j\}$. Recursively find each index l such that elimination of element a_{kl} occurred prior to this round in row $\mathbf{r}_k$, $k \in C$. Then it follows from Lemma 15 that $S(C \cup \{l\}, \mathbf{r}_k) = 0$ for each $k \in C \cup \{l\}$ prior to elimination of a_{kl}. Add each such index l to C. Prior to elimination of any element, it is possible to rearrange the rows and columns of A to obtain

$$\begin{pmatrix} a'_{11} & \cdots & a'_{1m} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a'_{m1} & \cdots & a'_{mm} & 0 & \cdots & 0 \\ a'_{(m+1)1} & \cdots & a'_{(m+1)m} & a'_{(m+1)(m+1)} & \cdots & a'_{(m+1)n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a'_{n1} & \cdots & a'_{nm} & a'_{n(m+1)} & \cdots & a'_{nn} \end{pmatrix},$$

where the first m rows and columns are those whose indices are elements of C. Since $S(C,\mathbf{r}_k) = 0$ for each row $\mathbf{r}_k$, $k \in C$, it follows that the sum of row $\mathbf{r}_k$ equals 0 and that for each $l \notin C$, element a_{kl} equals 0. ■

From the definition of M it follows that it is only possible to rearrange the rows and columns to obtain

$$\begin{pmatrix} X & 0 \\ Y & Z \end{pmatrix},$$

such that the sum of each row of X equals 0, if there is an absorbing set of states S' such that $\beta(s) = 0$ for each state $s \in S'$ and if $\gamma = 1$. For a proper option o, it is not possible to rearrange M that way. Since the sum of one row of M equals 0 if and only if the sum of the same row of A equals 0, it is not possible to rearrange A that way either. It follows from the contrapositive of Lemma 16 that $\det(A) \neq 0$, which also means that $\det(M) \neq 0$. This concludes the proof of Theorem 4.

Appendix B. Proof of Theorem 9

Assume that for each block λ and each value $v_d \in \mathcal{D}(S_d)$, the probability $P_d(v_d \mid \mathbf{s}_k, o)$ is identical for each state $\mathbf{s}_k \in \lambda$. Let P^o_{λ,v_d} denote that probability. We will show that $P_d(v_d \mid \mathbf{s}_i, o) = P_d(v_d \mid \mathbf{s}_j, o)$ checks under this assumption if $[\mathbf{s}_i]_{\Lambda_d} = [\mathbf{s}_j]_{\Lambda_d}$.

From Equation 14, the expression for $P_d(v_d \mid \mathbf{s}_i, o)$ is given by

$$\gamma \left[\beta(\mathbf{s}_i) P_d(v_d \mid \mathbf{s}_i, a) + (1 - \beta(\mathbf{s}_i)) \sum_{a' \in A} \pi(\mathbf{s}_i, a') \sum_{\mathbf{s}' \in S} P(\mathbf{s}' \mid \mathbf{s}_i, a') P_d(v_d \mid \mathbf{s}', o) \right].$$

We can expand the sum $\sum_{\mathbf{s}' \in S}$ by first summing over blocks λ of partition Λ_d and then over states $\mathbf{s}_k$ in block λ, replacing $P_d(v_d \mid \mathbf{s}_k, o)$ with P^o_{λ,v_d}:

$$\gamma \left[\beta(\mathbf{s}_i) P_d(v_d \mid \mathbf{s}_i, a) + (1 - \beta(\mathbf{s}_i)) \sum_{a' \in A} \pi(\mathbf{s}_i, a') \sum_{\lambda \in \Lambda_d} \sum_{\mathbf{s}_k \in \lambda} P(\mathbf{s}_k \mid \mathbf{s}_i, a') P^o_{\lambda,v_d} \right].$$

Since P^o_{λ,v_d} does not depend on $\mathbf{s}_k$, we can move it outside the summation to obtain

$$\gamma \left[\beta(\mathbf{s}_i) P_d(v_d \mid \mathbf{s}_i, a) + (1 - \beta(\mathbf{s}_i)) \sum_{a' \in A} \pi(\mathbf{s}_i, a') \sum_{\lambda \in \Lambda_d} P^o_{\lambda,v_d} \sum_{\mathbf{s}_k \in \lambda} P(\mathbf{s}_k \mid \mathbf{s}_i, a') \right].$$

We can expand the expression for $P_d(v_d \mid \mathbf{s}_j, o)$ in the same way to obtain

$$\gamma \left[\beta(\mathbf{s}_j) P_d(v_d \mid \mathbf{s}_j, a) + (1 - \beta(\mathbf{s}_j)) \sum_{a' \in A} \pi(\mathbf{s}_j, a') \sum_{\lambda \in \Lambda_d} P^o_{\lambda,v_d} \sum_{\mathbf{s}_k \in \lambda} P(\mathbf{s}_k \mid \mathbf{s}_j, a') \right].$$

If $[\mathbf{s}_i]_{\Lambda_d} = [\mathbf{s}_j]_{\Lambda_d}$, it follows immediately from the definitions of stochastic substitution property, policy respecting, termination respecting, and probability respecting of S_d that $P_d(v_d \mid \mathbf{s}_i, o) = P_d(v_d \mid \mathbf{s}_j, o)$. Lemma 5 states that the solution to the equations in Equation 15 is unique. Since we know that $P_d(v_d \mid \mathbf{s}_i, o) = P_d(v_d \mid s_j, o)$ is a solution, it follows from Lemma 5 that it is the only solution. This concludes the proof.

Appendix C. Proof of Theorem 10

Assume that for each block $\lambda \in \Lambda_R$ and each state $\mathbf{s}_k \in \lambda$, the expected reward $R(\mathbf{s}_k, o)$ as a result of executing option o is equal, and let R^o_λ denote that expected reward. We will show that $R(\mathbf{s}_i, o) = R(\mathbf{s}_j, o)$ checks under this assumption if $[\mathbf{s}_i]_{\Lambda_R} = [\mathbf{s}_j]_{\Lambda_R}$.

From Equation 9, the expression for $R(\mathbf{s}_i, o)$ is given by

$$R(\mathbf{s}_i, o) = \beta(\mathbf{s}_i) R(\mathbf{s}_i, a) + (1 - \beta(\mathbf{s}_i)) \sum_{a' \in A} \pi(\mathbf{s}_i, a') \left[R(\mathbf{s}_i, a') + \sum_{\mathbf{s}' \in S} P(\mathbf{s}' \mid \mathbf{s}_i, a') R(\mathbf{s}', o) \right].$$

We can expand the sum $\sum_{s' \in S}$ by first summing over blocks λ of partition Λ_R and then over states $\mathbf{s}_k$ in block λ, replacing $R(\mathbf{s}_k, o)$ with R^o_λ:

$$R(\mathbf{s}_i, o) = \beta(\mathbf{s}_i) R(\mathbf{s}_i, a) + (1 - \beta(\mathbf{s}_i)) \sum_{a' \in A} \pi(\mathbf{s}_i, a') \left[R(\mathbf{s}_i, a') + \sum_{\lambda \in \Lambda_R} \sum_{\mathbf{s}_k \in \lambda} P(\mathbf{s}_k \mid \mathbf{s}_i, a') R^o_\lambda \right].$$

Since R^o_λ does not depend on $\mathbf{s}_k$, we move it outside the summation to obtain

$$R(\mathbf{s}_i, o) = \beta(\mathbf{s}_i) R(\mathbf{s}_i, a) + (1 - \beta(\mathbf{s}_i)) \sum_{a' \in A} \pi(\mathbf{s}_i, a') \left[R(\mathbf{s}_i, a') + \sum_{\lambda \in \Lambda_R} R^o_\lambda \sum_{\mathbf{s}_k \in \lambda} P(\mathbf{s}_k \mid \mathbf{s}_i, a') \right].$$

We expand the expression for $R(\mathbf{s}_j, o)$ in the same way to obtain

$$R(\mathbf{s}_j, o) = \beta(\mathbf{s}_j) R(\mathbf{s}_j, a) + (1 - \beta(\mathbf{s}_j)) \sum_{a' \in A} \pi(\mathbf{s}_j, a') \left[R(\mathbf{s}_j, a') + \sum_{\lambda \in \Lambda_R} R^o_\lambda \sum_{\mathbf{s}_k \in \lambda} P(\mathbf{s}_k \mid \mathbf{s}_j, a') \right].$$

If $[\mathbf{s}_i]_{\Lambda_d} = [\mathbf{s}_j]_{\Lambda_d}$, it follows immediately from the definitions of stochastic substitution property, reward respecting, policy respecting, and termination respecting that $R(\mathbf{s}_i, o) = R(\mathbf{s}_j, o)$. Theorem 4 states that the solution to the equations in Equation 9 is unique. Since we know that $R(\mathbf{s}_i, o) = R(\mathbf{s}_j, o)$ is a solution, it follows from Theorem 4 that it is the only solution. This concludes the proof.

References

A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence, Special Volume on Computational Research on Interaction and Agency*, 72(1):81–138, 1995.

R. Bellman. A Markov decision process. *Journal of Mathematical Mechanics*, 6:679–684, 1957.

C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. *Proceedings of the International Joint Conference on Artificial Intelligence*, 14:1104–1113, 1995.

S. Bradtke and M. Duff. Reinforcement learning methods for continuous-time Markov decision problems. *Advances in Neural Information Processing Systems*, 7:393–400, 1995.

Ö. Şimşek and A. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proceedings of the International Conference on Machine Learning*, 21:751–758, 2004.

Ö. Şimşek, A. Wolfe, and A. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proceedings of the International Conference on Machine Learning*, 22, 2005.

T. Dean and R. Givan. Model minimization in Markov decision processes. *Proceedings of the National Conference on Artificial Intelligence*, 14:106–111, 1997.

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

T. Dean and S. Lin. Decomposition techniques for planning in stochastic domains. *Proceedings of the International Joint Conference on Artificial Intelligence*, 14:1121–1129, 1995.

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

T. Dietterich. State Abstraction in MAXQ Hierarchical Reinforcement Learning. *Advances in Neural Information Processing Systems*, 12:994–1000, 2000b.

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

Z. Feng and E. Hansen. Symbolic Heuristic Search for Factored Markov Decision Processes. *Proceedings of the National Conference on Artificial Intelligence*, 18:455–460, 2002.

Z. Feng, E. Hansen, and S. Zilberstein. Symbolic generalization for on-line planning. *Proceedings of Uncertainty in Artificial Intelligence*, 19:209–216, 2003.

R. Fikes and N. Nilsson. Strips: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

M. Ghavamzadeh and S. Mahadevan. Continuous-Time Hierarchical Reinforcement Learning. *Proceedings of the International Conference on Machine Learning*, 18:186–193, 2001.

C. Guestrin, D. Koller, and R. Parr. Max-norm Projections for Factored MDPs. *International Joint Conference on Artificial Intelligence*, 17:673–680, 2001.

D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical Solution of Markov Decision Processes using Macro-actions. *Uncertainty in Artificial Intelligence*, 14:220–229, 1998.

M. Helmert. A planning heuristic based on causal graph analysis. *Proceedings of the International Conference on Automated Planning and Scheduling*, 14:161–170, 2004.

B. Hengst. Discovering Hierarchy in Reinforcement Learning with HEXQ. *Proceedings of the International Conference on Machine Learning*, 19:243–250, 2002.

J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic Planning using Decision Diagrams. *Proceedings of Uncertainty in Artificial Intelligence*, 15:279–288, 1999.

A. Jonsson and A. Barto. Automated State Abstractions for Options Using the U-Tree Algorithm. *Advances in Neural Information Processing Systems*, 13:1054–1060, 2001.

A. Jonsson and A. Barto. A Causal Approach to Hierarchical Decomposition of Factored MDPs. *Proceedings of the International Conference on Machine Learning*, 22:401–408, 2005.

M. Kearns and D. Koller. Efficient Reinforcement Learning in Factored MDPs. *Proceedings of the International Joint Conference on Artificial Intelligence*, 16:740–747, 1999.

S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. *Proceedings of the International Conference on Machine Learning*, 21:560–567, 2004.

A. McGovern and A. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. *Proceedings of the International Conference on Machine Learning*, 18:361–368, 2001.

I. Menache, S. Mannor, and N. Shimkin. Q-Cut – Dynamic Discovery of Sub-Goals in Reinforcement Learning. *Proceedings of the European Conference on Machine Learning*, 13:295–306, 2002.

K. Murphy. *Active Learning of Causal Bayes Net Structure*. Technical Report, University of California, Berkeley, USA, 2001.

R. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. Ph.D. Thesis, University of California at Berkeley, 1998.

R. Parr and S. Russell. Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Information Processing Systems*, 10:1043–1049, 1998.

M. Pickett and A. Barto. Policyblocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. *Proceedings of the International Conference on Machine Learning*, 19: 506–513, 2002.

M. Puterman. *Markov Decision Processes*. Wiley Interscience, New York, USA, 1994.

B. Ravindran. *An Algebraic Approach to Abstraction in Reinforcement Learning*. Ph.D. Thesis, Department of Computer Science, University of Massachusetts, Amherst, USA, 2004.

S. Singh, A. Barto, and N. Chentanez. Intrinsically Motivated Reinforcement Learning. *Advances in Neural Information Processing Systems*, 18:1281–1288, 2005.

H. Steck and T. Jaakkola. Unsupervised Active Learning in Large Domains. *Proceedings of Uncertainty in Artificial Intelligence*, 18:469–476, 2002.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, USA, 1998.

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

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. *Advances in Neural Information Processing Systems*, 8:385–392, 1996.

S. Tong and D. Koller. Active learning for parameter estimation in Bayesian networks. *Advances in Neural Information Processing Systems*, 13:647–653, 2001.