Efficient Abstraction Selection in Reinforcement Learning

(Extended Abstract)

Harm van Seijen
Department of Computing Science
University of Alberta
Edmonton, Canada

Shimon Whiteson
Informatics Institute
University of Amsterdam
Amsterdam, The Netherlands

Leon Kester
Distributed Sensor Systems Group
TNO Defence, Security and Safety
The Hague, The Netherlands

Abstract

This paper introduces a novel approach for abstraction selection in reinforcement learning problems modelled as factored Markov decision processes (MDPs), for which a state is described via a set of state components. In abstraction selection, an agent must choose an abstraction from a set of *candidate abstractions*, each build up from a different combination of state components.

1 Introduction

In *reinforcement learning* (RL) (Sutton and Barto 1998; Szepesvári 2010), an agent learns a control policy by interaction with an initially unknown environment, described via a set of states, while trying to optimize the (sum of) rewards it receives, resulting from its actions. An RL problem is typically modelled as a *Markov decision process* (MDP) (Bellman 1957).

One of the main obstacles for learning a good policy is the *curse of dimensionality*: the problem size grows exponentially with respect to the number of problem parameters. Consequently, finding a good policy can require prohibitive amounts of memory, computation time, and/or sample experience (i.e., interactions with the environment). Fortunately, many real-world problems have internal structure that can be leveraged to dramatically speed learning.

A common structure in *factored MDPs* (Boutilier, Dearden, and Goldszmidt 1995), wherein each state is described by a set of state component values, is the existence of irrelevant (or near-irrelevant) state components, which affect neither the next state nor the reward. Removing such components can results in a dramatic decrease in the state space size. Unfortunately, in an RL setting, where the environment dynamics are initially unknown, learning which components are irrelevant is a non-trivial task that typically requires a number of statistical tests that depends on the size of the full state space (see for example (McCallum 1995)).

More recently, methods have emerged that focus on selecting the best *abstraction*, a subset of state components, from a set of candidate abstractions (Diuk, Li, and Leffler 2009; Konidaris and Barto 2009). The complexity of these methods depends only on the size of the abstractions used, which can be exponentially smaller than the full state space. The existing methods treat abstraction selection as an instance of model selection. Consequently, an abstraction is evaluated by measuring how well it predicts the outcome of an action, using some statistical measure.

In an RL setting, the model selection approach has a number of disadvantages. First, it does not take into account the on-line nature of RL, which requires the agent to balance exploration and exploitation. In order to effectively balance exploration and exploitation, it is important to know which abstraction is *currently* the best, given the samples observed so far. For example, small, fast-learning abstractions might be preferred in the early learning phase, while larger, more informative abstractions might be preferred later on. This creates a fundamental conflict with model selection, which is based on the premise that there is a single best abstraction that needs to be found. Second, an abstraction that is selected on the basis of the accuracy of its predictions is not guaranteed to be the abstraction that results in the most reward; an abstraction can be way off in its predictions, as long as it correctly guesses what the best actions are, it will results in high total reward.

We introduce a new, intuitive approach for abstraction selection that avoids the disadvantages of model selection. Our approach evaluates an abstraction by using the abstraction for action selection for a certain period of time and determining the resulting rewards. To maintain accurate estimates for the different abstractions, the agent needs to switch the abstraction it uses for action selection frequently. A key insight behind our approach is that an agent that has to choose between abstractions faces a similar exploration-exploitation dilemma as when choosing between its actions. Therefore, we formalize the task by introducing internal actions that allow the agent to switch between the different abstractions. The value of an internal action, which estimates the sum of future rewards, can be updated using regular RL methods. We call the derived task that includes the switch actions the *abstraction-selection* task. If the Markov property holds for this derived task, which states that the outcome of an action only depends on the current state and not on the history, the derived task is an MDP itself. In this case, convergence is guaranteed to the abstraction that is asymptotically the best, as well as to the optimal policy of that abstraction.

2 (Factored) Markov Decision Processes

Markov decision processes (MDPs) (Bellman 1957) are used to model sequential decision problems, where a decision maker, the *agent*, interacts with its *environment* in a sequential way. An MDP is defined by a 4-tuple $(\mathcal{X}, \mathcal{A}, \tau, \rho)$ where $\mathcal{X}$ is a finite set of states, and $\mathcal{A}$ is a finite set of actions. The *state transition function* τ gives, for each triple $(x, a, y) \in \mathcal{X} \times \mathcal{A} \times \mathcal{X}$, the probability of moving to state y, when taking action a in state x. The reward function ρ gives for each triple $(x, a, y) \in \mathcal{X} \times \mathcal{A} \times \mathcal{X}$ a probability distribution over $\mathbb{R}$. The semantics are that the reward received by the agent when taking action a in state x and moving to state y is drawn from the distribution $\rho(x, a, y)$. In general, not all actions from $\mathcal{A}$ are accessible in each state $x \in \mathcal{X}$. We denote the subset of actions accessible in x as $\mathcal{A}(x) \subseteq \mathcal{A}$.

The agent takes actions at discrete time steps $t = \{0, 1, 2, 3, ...\}$. The agent's behaviour is determined by its policy $\pi : \mathcal{X} \rightarrow \mathcal{A}$, which specifies for each state the action to take. Typically, an agent tries to find the policy that maximizes the expected value of the *return* G_t, which is defined as the (infinite) sum over future discounted rewards:

$$G_t = R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + ... \qquad (1)$$

MDPs can have *terminal* states, which divide the agent's interaction with the environment into *episodes*. When a terminal state is reached, the current episode ends and a new one is started by resetting the environment to its initial state. The infinite sum from Equation (1) does not continue across episodes. In other words, if a terminal state is reached at time step T, the sum terminates after reward R_T.

A *factored MDP* is an MDP where the set of states, $\boldsymbol{\mathcal{X}}$, is constructed from N *state components*:[1]

$$\begin{aligned} \boldsymbol{\mathcal{X}} &= \mathcal{X}^1 \times \mathcal{X}^2 \times ... \times \mathcal{X}^N \\ &= \{(x^1, x^2, ..., x^N) | x^i \in \mathcal{X}^i, 1 \leq i \leq N\}\,. \end{aligned}$$

A *context-specific* state space is a state space for which different states are described by different components. Because using a state space where the elements are vectors of different size is unintuitive, we model a context-specific state space as a factored state space, spanned by all the possible state components, with a special value added to each component, indicated by #. When a state has value # for one of its components, this indicates that this state component is actually not defined for that state.

In reinforcement learning the environment dynamics (that is, the functions τ and ρ) are unknown. Value-function based methods improve the policy by iteratively improving estimates of the optimal value function using observed samples. The optimal value of a state gives the expected return for that state when following an optimal policy.

3 Abstractions

An abstraction is a function that maps states from one state space to states from a different state space:

$$\mu : \boldsymbol{\mathcal{X}} \rightarrow \boldsymbol{\mathcal{Y}}\,.$$

[1]We use the term 'component' rather then 'feature', because our definition (based on a *set*) differs slightly from the typical definition of a feature (based on a *function*).

We consider only abstractions that correspond with ignoring certain components. We use a set-superscript to indicate the components that are used:

$$\mu(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}}\,, \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}},$$

with $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times ... \times \mathcal{X}^N$ and $\mathcal{S} \subseteq \{1, 2, ..., N\}$. For example, for $\boldsymbol{x} = (3, 5, 8, 2, 0)$ and $\mathcal{S} = \{1, 3\}$, $\mu(\boldsymbol{x}) = (3, 8)$.

A *context-specific abstraction* is an abstraction that maps states to a context-specific state space. An example of such an abstraction is:

$$\mu(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\{1,3\}} & \text{if } \boldsymbol{x}^{\{5\}} = 0 \\ \boldsymbol{x}^{\{2,4\}} & \text{otherwise}\,, \end{cases} \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}}.$$

In this case, $\boldsymbol{x}^{\mathcal{S}}$ means that all the components with an index not in $\mathcal{S}$ get the value #. For example, with $\boldsymbol{x} = (3, 5, 8, 2, 0)$ and μ as defined above, $\mu(\boldsymbol{x}) = (3, \#, 8, \#, \#)$.

An abstraction μ applied to an MDP $\mathcal{M}$ defines a derived task with state space $\boldsymbol{\mathcal{Y}}$. If for this derived task the Markov property holds, we say that abstraction μ is a *Markov abstraction* for MDP $\mathcal{M}$. If this is the case, the derived task itself is also an MDP.

One way to construct a Markov abstraction is by removing *irrelevant* components from $\boldsymbol{\mathcal{X}}$. These are components that neither affect the reward received by the agent, nor the value of any other component (besides itself). But also removing relevant components can yield a Markov abstraction. It can be shown that removing *independent* components from $\boldsymbol{\mathcal{X}}$ also results in a Markov abstraction, even when the removed components contained relevant information. An independent component is a component whose value is drawn, at each time step, from the same, fixed, probability distribution; hence, its next value is not affected by current component values or values from the past. An independent component can model a seemingly random, but relevant environment process. For example, such a component could represent a smartphone app that provides real-time information to a daily commuter on the early-morning traffic conditions. The agent will experience the removal of a relevant, independent component as increased environment stochasticity. Therefore, the performance will be lower in general.

4 Abstraction Selection for a Contextual Bandit

We now demonstrate how to construct the abstraction-selection task of a contextual bandit problem. A contextual bandit task can be modelled as an episodic MDP that terminates after a single action. Each arm of the contextual bandit task corresponds with an action, while each context corresponds with a state.

As a motivating example for abstraction selection in this domain, consider the task of placing the most relevant ads on a large number of different websites. If websites are described by 50 binary state-components, there are $2^{50} \approx 10^{15}$ states. Because for each state, the click-through rate of each ad needs to be learned, using all state components is not practical. Instead, the best combination of three components could be learned. This requires evaluation of $\binom{50}{3} = 19,600$

abstractions, each consisting of $2^3 = 8$ states. So, the total number of states is reduced to $19,600 \times 8 = 160,000$, a difference of a factor 10^{10}.

4.1 Abstraction-Selection Task

Consider a contextual bandit problem modelled by the MDP $\mathcal{M} = (\boldsymbol{\mathcal{X}}, \mathcal{A}, \tau, \rho)$ and a set of K candidate abstractions $\boldsymbol{\mu} = \{\mu^1, \dots, \mu^K\}$. The abstraction-selection task for $\mathcal{M}$ is a task resulting from applying a context-specific abstraction to an extended version of $\mathcal{M}$ that includes switch actions and an extra state component, indicating the candidate abstractions currently selected. We indicate this extended version by $\mathcal{M}^+$, and the context-specific abstraction applied to it by μ^+.

First, we define the extended version of $\mathcal{M}$: $\mathcal{M}^+ = (\boldsymbol{\mathcal{X}}^+, \mathcal{A}^+, \tau^+, \rho^+)$. The state space $\boldsymbol{\mathcal{X}}^+$ extends $\boldsymbol{\mathcal{X}}$ by adding a special abstraction component:

$$\boldsymbol{\mathcal{X}}^+ = \mathcal{X}^{abs} \times \boldsymbol{\mathcal{X}} \,,$$

with $\mathcal{X}^{abs} = \{0, 1, \dots, K\}$. The values in $\mathcal{X}^{abs}$ refer to the indices of the candidate abstractions. The value 0 means that there is currently no candidate abstraction selected. The initial value of $\mathcal{X}^{abs}$ is always 0.

The action set $\mathcal{A}^+$ is created by adding K switch action to $\mathcal{A}$, one corresponding to each candidate abstraction:

$$\mathcal{A}^+ = \mathcal{A} \cup \{a^{sw,1}, \dots, a^{sw,K}\}$$

The switch actions are only available in states for which component $\mathcal{X}^{abs}$ has value 0 (i.e., $\boldsymbol{x}^{\{abs\}} = 0$). In such states, no regular actions are available. Hence, for all $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+$:

$$\mathcal{A}^+(\boldsymbol{x}) = \begin{cases} \{a^{sw,1}, \dots, a^{sw,K}\} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ \mathcal{A} & \text{if } \boldsymbol{x}^{\{abs\}} \neq 0 \,, \end{cases}$$

The effect of taking switch action $a^{sw,i}$ is that the value of component $\mathcal{X}^{abs}$ is set to i for $1 \leq i \leq K$. Because a switch action is an internal action it does not affect any of the other component values. In addition, the agent receives no reward for taking an internal action. Taking a regular action has no effect on the value of $\mathcal{X}^{abs}$. The effect of a regular action on the other component values is defined by $\mathcal{M}$.

While $\mathcal{M}^+$ contains switch actions and a component to keep track of the selected abstraction, no abstractions have been applied yet. To get the abstraction-selection task, a context-specific abstraction μ^+ has to be applied to $\mathcal{M}^+$. μ^+ is defined, for all $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+$, as:

$$\mu^+(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\{abs\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ (\boldsymbol{x}^{\{abs\}}, \mu^i(\boldsymbol{x}^{\{1,\dots,N\}})) & \text{if } \boldsymbol{x}^{\{abs\}} = i, i \neq 0 \end{cases}$$

The following theory holds:

Theorem 1 *The abstraction-selection task of a contextual bandit problem obeys the Markov property.*

This theory holds, because the history of an abstraction state just consists of the initial state and the switch action, which is always the same for the same state. Because the task is Markov, standard RL methods can be used to solve it.

4.2 Example Abstraction-Selection Task

Consider a simple contextual bandit problem with two actions, a^1 and a^2, and a state space spanned by two binary components: $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \mathcal{X}^2$, with $\mathcal{X}^1 = \{true, false\}$ and $\mathcal{X}^2 = \{true, false\}$. All four possible states have the same probability of occurring. The expected value of the rewards of the two actions, conditioned on the state, is shown in Table 1. Because a contextual-bandit problem has a trivial next state (a terminal state), the reward function is expressed only as function of a state-action pair. From this table, the optimal policy can be easily deduced: action a^2 should be taken in states where $X^1 = true$ and action a^1 should be taken in the other states. The expected reward of this policy is $\sum P_0(x^1, x^2) \cdot \max_a \mathbb{E}[\rho((x^1, x^2), a)] = 1.5$. Although the state space of this task is small enough to use both components, for illustrative purposes we assume that the agent must choose between using either component $\mathcal{X}^1$ or component $\mathcal{X}^2$. In other words, its set of candidate abstractions is $\boldsymbol{\mu} = \{\mu^1, \mu^2\}$, where $\mu^1(\boldsymbol{x}) = \boldsymbol{x}^{\{1\}}$ and $\mu^1(\boldsymbol{x}) = \boldsymbol{x}^{\{2\}}$ for $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$.

Table 1: Expected rewards and initial state probability P_0 for $\boldsymbol{X} = (X^1, X^2) \in \boldsymbol{\mathcal{X}}$

X^1	X^2	$P_0(\boldsymbol{X})$	$\mathbb{E}[\,\rho(\boldsymbol{X}, a^1)\,]$	$\mathbb{E}[\,\rho(\boldsymbol{X}, a^2)\,]$
true	true	0.25	0	+4.0
true	false	0.25	0	+2.0
false	true	0.25	0	-2.0
false	false	0.25	0	-4.0

Given the contextual-bandit task described above, the extended state space and action space are defined as follows:

$$\begin{aligned} \mathcal{A}^+ &= \{a^1, a^2, a^{sw,1}, a^{sw,2}\}\,, \\ \boldsymbol{\mathcal{X}}^+ &= \mathcal{X}^{abs} \times \mathcal{X}^1 \times \mathcal{X}^2\,, \end{aligned}$$

with $\mathcal{X}^{abs} = \{0, 1, 2\}$. Action $a^{sw,i}$ sets the value-component of a state corresponding with $\mathcal{X}^{abs}$ to i for $i \in \{1, 2\}$.

In addition, the mapping μ^+ is defined, for all $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+$, as follows:

$$\mu^+(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\{abs\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ \boldsymbol{x}^{\{abs,1\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 1 \\ \boldsymbol{x}^{\{abs,2\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 2\,, \end{cases}$$

The complete MDP resulting from applying μ^+ to $\mathcal{M}^+$ is visualized in Figure 1. The expected rewards for actions a^1 and a^2 can be derived from Table 1. For example, the expected value of $\rho(\boldsymbol{X}, a^2)$ given $X^1 = true$ is $(0.25 * 4 + 0.25 * 2)/0.5 = +3$. Overall, abstraction μ^1 is the better choice, because selecting it results in an expected reward of +1.5. By contrast, selecting abstraction μ^2 results in an expected reward of only +0.5.

The switch actions are key to ensure that the best abstraction is selected. To understand why, consider an abstraction selection approach that does not use switch action, but simply select the abstraction whose current abstract state predicts the highest reward (note that the agent can observe the

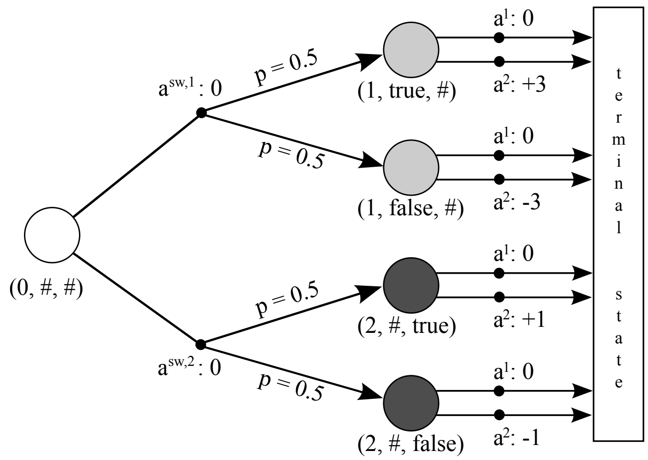

Figure 1: Abstraction-selection task for the contextual bandit task example. Circles indicate states. States with the same colour, use the same candidate abstraction. The small black dots indicate actions. p refers to the transition probabilities for a stochastic action. The value after each action identifier is the expected value of the reward when taking that action from the corresponding state.

abstract states from all abstractions simultaneously, because it has access to all state-components). If this strategy would be applied to the example task described above and the current state would be $(X^1 = false, X^2 = true)$, then, after learning the correct expected rewards for each state, abstraction μ^1 predicts action a^1 is best, yielding a reward of 0 (see Figure 1). On the other hand, abstraction μ^2 predicts action a^2 is best, yielding an expected reward of +1. Hence, the agent would choose abstraction μ^2 and select action a^2. However, from Table 1 is can be observed that a^1 is actually the better action (a^2 results in an expected reward of -2). This conflict occurs because this strategy implicitly uses both components to select the switch action while the abstraction uses only a single component, resulting in an underlying task that is non-Markov.

5 Abstraction Selection for MDPs

The approach outlined for the contextual bandit task also applies to episodic MDPs with longer episodes. The agent chooses at the start of each episode which abstraction to use; it does not change its abstraction within an episode. In contrast to the contextual bandit case, the abstraction-selection task of a general episodic MDP is not always Markov. To ensure that the abstraction-selection task of a general episodic MDP is Markov, all involved candidate abstractions must be Markov. The presence of non-Markov candidate abstractions makes the abstraction-selection task also non-Markov. However, the negative effects of this can be mitigated by using Monte-Carlo backups (see Section 6.2).

Figure 2 shows results for a simple navigation task: a robot has to find its way from a start state to a goal state in a 15×15 square grid, using 4 directional actions. The start state is in one corner; the goal state is in the opposite corner. Besides a component specifying the agent's position in the grid, the agent has access to n independent, 'structural' components, each consisting of 4 values. All structural components are independent (that is, their values are drawn from a fixed probability distribution at each time step — see Section 3). Besides that, they are all irrelevant, except for one. The agent does not know which structural component is relevant. The relevant structural component determines which direction each action corresponds with. Hence, an abstraction that ignores this component produces random behavior.

We compared our switching strategy, which uses n candidate abstractions, each consisting of the position feature plus one of the structural features, with a strategy that simply uses all components, for $n = 5$ and $n = 10$. In addition, we compared against a method that only uses the position component, as well as a method that knows ahead of time which component is relevant, and ignores all irrelevant components. We have results only for $n = 5$ for the method using all components, because the state space size for $n = 30$ was infeasible (the full state space has a size of 10^{20} in this case). We used two strategies for updating the values of switch actions. One strategy updates switch actions using regular Q-learning backups, while the other strategy uses Monte-Carlo backups. The advantage of the first strategy is that a value for the switch actions can be learned off-policy (requiring less exploration). The advantage of the second strategy is that the value of switch actions is not affected by the values of a specific abstraction; it is only affected by the actual return produced by an abstraction. The second strategy is especially useful if the candidate set contains non-Markov abstractions (see Section 6.2).

The results show that our switching method can obtain a performance only slightly less than optimal (i.e., knowing a priori which component is relevant), and, as expected, much higher than using all components.

By allowing the agent to switch abstraction in the middle of an episode, our approach can also be applied to non-episodic MDPs. In this case, extra information has to be provided to the agent. Specifically, an extra component has to be provided whose values divide the state space into *context regions*. The semantics of these regions is that states in the same region share the same relevant components. Instead of learning one abstraction for the complete state space, the agent learns a separate abstraction for each region. The agent chooses a new abstraction whenever it crosses the border between two regions.

6 Discussion

In this section, we discuss the main advantage of candidate abstractions, and discuss how to back up switch-action values when dealing with non-Markov abstractions or abstractions of different size.

6.1 Why Candidate Abstractions?

Learning the best abstraction from a set of candidate abstractions can be interpreted as a form of structure learning. However, in contrast to many other approaches to structure learning, it assumes a high degree of prior knowledge about the structure (represented by the set of candidate abstractions). This might seem like a disadvantage, but it is

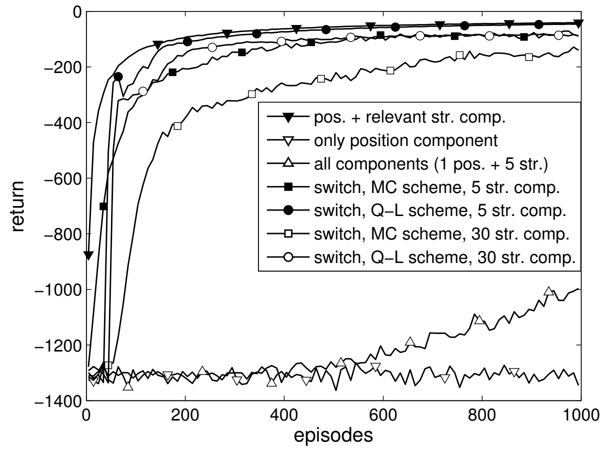

Figure 2: Performance of different methods on a small navigation task.

a deliberate choice. While it restricts the application to domains where such prior knowledge is either available or easily obtainable, exploiting partial prior knowledge about the structure allows us to tackle huge problems that would be otherwise infeasible to solve (see the ad-placement example at the beginning of Section 4). The main problem with structure learning methods that try to learn problem structure from scratch is that it often takes as much effort (in terms of samples and computation) to learn the structure and then solve the task using this structure, as it would to solve the task without learning the structure. Therefore, such methods are mainly limited to transfer-learning scenarios, where the high initial cost for learning the structure can be offset against many future applications of this structure. Extending our method by 'inventing' and evaluating new abstractions on-the-fly, would potentially cause similar issues.

Part of the appeal of our method is its simplicity: the agent simply decides what the best abstraction is through trial and error. This brings up the question: is it even necessary to construct an abstraction-selection task? Why not simply evaluate the abstractions one-by-one? To address this question, we refer back to the ad-placement example from Section 4. The primary goal is to optimize the *on-line* performance, that is, the performance *during* learning. The switch actions give the agent the ability to decide whether to continue exploring an abstraction based on the performance of other abstractions. If the relative performance of one abstraction is clearly below average, the agent might decide to stop using it (or just update it by off-policy learning), even when its performance has not converged yet. Of course, in order for this to work, a robust performance measure is required, which we discuss next.

6.2 Robust Abstraction Selection with Monte-Carlo Backups

Figure 2 shows that, for the simple navigational task considered, backing up the switch action values using Q-learning backups performs (slightly) better than using Monte-Carlo backups. However, in many domains, Monte-Carlo backups will have the upper hand. In the considered task, all candidate abstractions were Markov and of the same size, which means that the state values of the different abstractions form a good indicator of the quality of an abstraction. However, if some of the candidate abstractions are non-Markov, this is no longer true. In this case, Monte-Carlo backups are the better choice, because they use the complete return to update the switch-action values, and do not rely on abstraction state values. Hence, the switch-action values cannot get corrupted by incorrect abstraction values. By contrast, with Q-learning backups, the agent could be misled into thinking a bad abstraction is good.

A second scenario where Monte-Carlo backups are a better choice is when the set of candidate abstractions is a mixture of small and large abstractions. Small abstraction converge fast and are typically a better choice in the early learning phase. However, for an agent to recognize this, it should not only have accurate switch-action values for the small abstractions, but also for the large abstractions. If the switch-action values depend on the abstraction state values, as is the case with Q-learning backups, reliable abstraction selection can only occur after *all* abstractions have more or less converged, when it is no longer beneficial to use an abstraction of lower resolution. With Monte-Carlo backups, the small, quickly learning abstractions can be quickly recognized as better (in terms of return) than the larger, more informative, slowly learning abstractions. Hence, the agent can boost its initial performance by using the small abstractions, and only using the large abstractions once they have sufficiently converged so that their return is larger than that of the smaller abstractions.

References

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

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *International Joint Conference on Artificial Intelligence*, 1104–1113.

Diuk, C., Li, L., and Leffler, B.R. (2009). The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*.

Konidaris, G. and Barto, A. (2009). Efficient skill learning using abstraction selection. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*, 1107–1112.

McCallum, A.K. (1995). *Reinforcement Learning with Selective Perception and Hidden States*. Ph.D. Dissertation, University of Rochester.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, Massachussets: MIT Press.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103.