Computational Intelligence, Volume 00, Number 000, 2012

Efficient Abstraction Selection in Reinforcement Learning

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

This article addresses reinforcement learning problems based on factored *Markov decision processes* (MDPs) in which the agent must choose among a set of *candidate abstractions*, each build up from a different combination of state components. We present and evaluate a new approach that can perform effective abstraction selection that is more resource-efficient and/or more general than existing approaches. The core of the approach is to make selection of an abstraction part of the learning agent's decision-making process by augmenting the agent's action space with internal actions that select the abstraction it uses. We prove that under certain conditions this approach results in a derived MDP whose solution yields both the optimal abstraction for the original MDP and the optimal policy under that abstraction. We examine our approach in three domains of increasing complexity: contextual bandit problems, episodic MDPs, and general MDPs with context-specific structure.

Key words: reinforcement learning, model-free learning, structure learning, abstraction selection

1. INTRODUCTION

In *reinforcement learning* (RL) (Kaelbling *et al.*, 1996; Sutton and Barto, 1998; Szepesvri, 2010), an agent learns a control policy by interaction with an initially unknown environment. The agent adapts its behaviour in response to the rewards it receives in order to maximize its expected *return*, the discounted sum over rewards. When the RL problem is modelled as a *Markov decision process* (MDP) (Bellman, 1957a), the agent's policy can be represented as a mapping from states to actions.

Learning in MDPs is challenging because of the *curse of dimensionality*: the size of the state space 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 *et al.*, 1995), wherein each state is described by a set of state component values, is the existence of irrelevant (or near-irrelevant) state components, which do not affect the next state or reward. When the irrelevant components are identified,

Address correspondence to Harm van Seijen, University of Alberta, 2-21 Athabasca Hall, Edmonton, Alberta, Canada T6G 2E8; email: harm.vanseijen@ualberta.ca

© 2012 The Authors. Journal Compilation © 2012 Wiley Periodicals, Inc.

exploiting this structure is trivial: the irrelevant state components can simply be removed from the state description, causing an exponential decrease in the state space size in the number of components that are removed. For tasks with unidentified irrelevant components, exploiting the structure is non-trivial and requires a form of structure learning. Ideally, the agent will identify, after some learning period, the *minimal component set*, i.e., the smallest subset of the total set of state components that contains all the relevant information.

The task of identifying and removing irrelevant state components has long been recognized as crucial in tackling problems with large state spaces. For example, McCallum (1995) proposed a method (U-tree) that learns a tree for efficient representation of a policy that uses only task-relevant state distinctions. However, the number of statistical tests that have to be performed to learn this tree depends of the size of the full set of state components, making the method impractical for large problems. More recently, methods have been proposed that aim to select the best *abstraction*, i.e., combination of state components, from a set of candidate abstractions (Diuk *et al.*, 2009; Konidaris and Barto, 2009).[1]

From the point of view of structure learning, the set of candidate abstractions encodes prior knowledge about (combinations of) irrelevant state components that can be leveraged to reduce the problem size. A set of candidate abstractions can model prior knowledge of the problem's structure that ranges from full knowledge (i.e., the set contains only the minimal abstraction) to no knowledge (i.e., the set contains all possible combinations of state components). When evaluating the candidate abstractions, the total number of states the agent must reason about is the sum of the state spaces induced by these abstractions. If there are many irrelevant state components and sufficient prior knowledge about it is encoded in the set of candidate abstractions, then this sum can be exponentially smaller than the state space resulting from using all components. Consequently, abstraction selection has the potential to make learning feasible on very large problems.

Diuk *et al.* (2009) propose an abstraction selection method based on their k-meteorologists algorithm, called *SCRAM-R_{max}*. Their method applies abstraction selection to select a *type* abstraction of a *relocatable action model* (Leffler *et al.*, 2007). A type abstraction clusters the state space into areas with similar transition dynamics. Besides a set of candidate abstractions their method requires as input a *next-state function*. This function requires a high degree of prior knowledge about the general transition function. For domains where this knowledge is available, learning can be performed very efficiently. On the other hand, when this knowledge is not available their method cannot be applied.

By contrast, Konidaris and Barto (2009) present an abstraction selection strategy that uses as input only a set of candidate abstractions. They propose to select abstractions based on the *Bayesian Information Criterion* (BIC) (Schwarz, 1978) and introduce a method that incrementally processes samples for computing the BIC value of a candidate abstraction. However, the space and computation costs for processing a single sample scale quadratically with the size of the abstraction. In addition, the computation cost for selection scales cubicly. Consequently, their approach is unsuitable in settings where such resources are highly constrained.

We present and evaluate an alternative approach for abstraction selection that requires less prior knowledge than that of Diuk *et al.* but is more efficient than that of Konidaris and Barto. The core of our approach is to make the selection of the abstraction part of the RL agent's decision-making process by adding internal 'switch actions' to the action set that allow the agent to switch from one abstraction to another. The selected abstraction is then used for external action selection. Effectively, this process constructs from the original MDP and the set of candidate abstractions an abstract, derived RL task, which we call the *abstraction-selection task*. Due to the reasons mentioned above, the state space of this task can be exponentially smaller than that of the original MDP. In addition, off-policy learning can be employed to reduce the sample efficiency even further.

Our approach evaluates candidate abstractions in a fundamentally different way than the two approaches mentioned above. Rather than determining which abstractions best predict the observed data according to some statistical measure, our approach estimates the return each abstraction will produce, given the data seen and updates performed so far. This is achieved by trying out the different abstractions and updating the corresponding switch actions with the return produced

[1] Diuk *et al.* (2009) call this task 'feature selection'.

by these abstractions. Because abstractions are evaluated based on the quality of their current policy, small abstractions that quickly learn a decent policy will be selected in the early learning phase, while abstractions that need more data to obtain an accurate policy will be selected later. In addition, because the selection problem is transformed into a single derived task, it can be solved with resource-efficient model-free methods, making abstraction selection available in settings with strong constraints on computation and memory.

The main contribution of this article lies in specifying under which conditions the derived task is itself an MDP. In this case, basic RL methods are guaranteed to converge to the optimal abstraction and the optimal policy under that abstraction. In addition, we present model-free update strategies for off-policy updating abstractions that are not selected, in order to improve the sample efficiency in settings with more resources. We also empirically compare different model-free update strategies.

We explore our learning approach in three domains of increasing complexity. The first domain is a contextual bandit task, a variation on the regular bandit task where the expected return of an arm is conditioned on context information, represented by a set of state components. Abstraction selection in this case involves selecting an efficient combination of state components for describing the context states. After each arm pull a different abstraction can be selected.

The second domain considers abstraction selection for an episodic MDP. In this domain, abstraction selection occurs at the start of each episode and the selected abstraction is then used throughout the episode. We demonstrate that for this domain, in contrast to the contextual bandit domain, not every set of candidate abstractions will result in a Markov abstraction-selection task. Specifically, in order to obtain a Markov abstraction-selection task, all candidate abstractions have to be *Markov*. We prove that a Markov abstraction can be constructed by removing certain state component types from the set of all state components.

The third domain considers general MDPs with context-specific structure, i.e., MDPs where in different parts of the state space different state components are relevant. For this domain, partial knowledge of the structure is assumed. Specifically, knowledge is assumed about which states share the same relevant components (but not *which* components are relevant for those states). By clustering states that share the same relevant components, the state space can be divided into different regions, such that each region shares the same relevant components. Abstraction selection occurs whenever the agent moves out (or in) such a region. In this domain, to obtain a Markov abstraction-selection tasks, additional restrictions hold. Since a contextual bandit task can be seen as a special case of an MDP, the theory discussed in this domain also applies to contextual bandit tasks with context-specific structure.

2. RELATED WORK

In this section, we discuss the related work regarding abstraction selection, and more generally structure learning.

2.1. Abstraction Learning

In principle, all abstraction selection methods can achieve an exponential decrease in the problem size compared to using all state components, given that the problem contains a large number of irrelevant components and sufficient prior knowledge about them. However, abstraction selection methods differ greatly in terms of their generality and efficiency. In this section, we discuss several related abstraction selection methods and contrast them with our approach.

Konidaris and Barto (2009) evaluate each abstraction from a set of candidate abstractions by determining its *BIC value*, a standard criterion for model selection that explicitly penalizes high-dimensional abstractions. They propose an algorithm for iteratively determining the BIC value for *skills* represented using options (Sutton *et al.*, 1999). Since only a few state components may be relevant for a given skill, using only those components can greatly speed learning. The set of candidate abstractions (which they call 'library of abstractions') for a new skill can be constructed from the abstractions used for other skills. Their algorithm iteratively processes each incoming sample and computes the BIC value after m samples have been observed. Because they combine this approach with function approximation, it can be applied to continuous domains. It requires $\mathcal{O}(q_i^2)$ memory

and computation at each time step and $\mathcal{O}(q_i^3)$ computation for selection pertime step candidate abstraction i using a function approximator with q_i features.

By contrast, we do not consider function approximation in this article. Nonetheless, combining our abstraction-selection tasks with function approximation is straightforward and requires no additional algorithmic developments. In such a setting, our approach would require only $\mathcal{O}(q_i)$ memory and computation per time step for each candidate abstraction. Updating the switch actions and selecting an abstraction adds no significant extra cost. Thus, our approach is substantially more efficient and thus better suited for settings with strong restrictions on memory and computational resources.

Unfortunately, the work of Konidaris and Barto does not specify a complete RL method, preventing a meaningful empirical comparison between our approach and theirs. Performing such a comparison would require developing a new RL method in which their algorithm appears as a subroutine. Doing so is beyond the scope of this article.

Diuk *et al.* (2009) propose the adaptive k-meteorologists algorithm, a general method for learning a *probabilistic concept*, i.e., a function $h : X \to Y$, with $Y = [0, 1]$, where Y corresponds with a probability. In the meteorologist example, after which their algorithm is named, Y is the probability of rain and X are the state components that can be used to predict this probability. The algorithm learns which function h, from a set of functions H, best predicts a series of stochastic observations $z_t \in \{0, 1\}$, generated by an unknown h^*. Since the algorithm is based on the *KWIK* framework (Li *et al.*, 2008), it outputs $\perp$ ("I don't know") when it is not sure about its prediction. Using this algorithm, they construct a method for learning an abstraction among a set of candidate abstractions (SCRAM-R_{max}).

Despite the generality of the adaptive k-meteorologists algorithm, its use for abstraction selection is limited because it compares predictions about the *same* variable. Since the transition dynamics of different candidate abstractions make predictions about the values of *different* state components, the adaptive k-meteorologists algorithm cannot effectively compare them. Hence, SCRAM-R_{max} does not apply to the problem of selecting among *state* abstractions. Therefore, their method can only be used for selecting among *type* abstractions for a *relocatable action model* (Leffler *et al.*, 2007). This requires a different, more advanced form of prior knowledge of the transition function.

More specifically, a type is a mapping $\kappa : S \to C$ that clusters the state space into regions that share the same transition dynamics. This transition dynamics is expressed using $t : C \times A \to P(O)$, which maps a type $c \in C$ and action $a \in A$ to a probability distribution over possible *outcomes*. These outcomes are a state-independent specification of an action's effects (e.g. "agent moves one step to the left"). By combining this outcome with a *next-state function* $\eta : S \times O \to S$, which maps states and observations to next states, the next state can be computed. SCRAM-R_{max} requires as input a set of type mappings κ and the next-state function η. The function η requires a high degree of prior knowledge about the transition function. In fact, in a deterministic environment, this function is similar to the transition function, making the problem more of a planning problem than an RL problem. The core property that allows for the k-meteorologists algorithm to be used is that each type mapping can be evaluated based on predictions about the same outcome variable O.

The task on which SCRAM-R_{max} is evaluated is similar to our extended Mars rover task (see Section 7.5). Both are navigation tasks and require a robot to find its way from start to goal locations, while encountering different terrain types. A difference is that in the SCRAM-R_{max} domain, terrain types share equal action effects, while in the extended Mars rover task they share equal irrelevant components (and can have different action effects). The key difference, however, is that the learning problem presented to SCRAM-R_{max} is much smaller than the learning problem captured by the extended Mars rover task, since the function η is assumed to be known in the SCRAM-R_{max} domain. The candidate abstractions used for SCRAM-R_{max} consists of only single binary components, i.e., these abstractions have a size of only 2. By contrast, the size of a candidate abstraction in the extended Mars rover domain, which consists of a position component and a structural component, is 900. Note that, since Diuk's method relies on restrictive assumptions that do not hold for our extended Mars rover task, we do not compare to it directly.

2.2. Other Methods Exploiting Structure

Whereas abstraction selection seeks to identify state components that can be excluded from the state space completely, related approaches seek to identify conditional independence between components, typically expressed using a *dynamic Bayesian network (DBN)* (Dean and Kanazawa, 1989; Murphy, 2002), also referred to as a two-slice temporal Bayesian network. In planning problems, DBNs enable efficient solution methods that do not require explicit enumeration of the state space (Boutilier *et al.*, 1995). Similarly, in learning problems where the structure of the DBN is known (but not its parameter values), near-optimal performance can be obtained using only samples and computation polynomial in the number of parameters of the DBN, which may be exponentially smaller than the number of states (Kearns and Koller, 1999).

However, when the structure of the DBN is not known in advance, the problem becomes much harder. Learning is still possible in a sample-efficient way, either by requiring prior knowledge of the maximum degree of the DBN (Strehl *et al.*, 2007; Diuk *et al.*, 2009; Kroon and Whiteson, 2009) or by placing restrictions on the planning horizon (Chakraborty and Stone, 2011). However, the time and space requirements of such methods are linear in the number of states, making them impractical for large problems (as an example, the largest problem Diuk's DBN learning method is tested on consists of only 512 states). While abstraction selection requires more prior knowledge (a set of candidate abstractions), this knowledge can be leveraged to tackle problems with huge state spaces: we show good results for large size bandit problems (10^{45} states, see Section 5.4) as well as large size MDPs (10^{20} states, see Section 6.3). Such tasks are far beyond the reach of DBN learning methods.

Also related to our approach is the work on automatically finding good state abstractions (instead of selecting the best abstraction among a set of candidate abstractions). In a planning context, such methods can be roughly divided in exact and approximate methods. Exact methods aggregate states for which the transition and reward functions are equal (Givan *et al.*, 2003; Boutilier *et al.*, 2000; Ravindran and Barto, 2003). By contrast, approximate methods aggregate states for which the transition and reward functions are similar according to some metric (Dean *et al.*, 1997; Ferns *et al.*, 2004). These methods differ from ours in that they focus on planning and thereby assume complete knowledge of the transition dynamics, whereas we focus on learning, in which such knowledge is absent.

There also exist methods for automatically finding good abstractions for the learning setting. For example, Chapman and Kaelbling, L.P. (1991) propose a method for on-line state abstraction of states with the same reward and Q-value for each action. Similarly, Jong and Stone (2005) propose a method that can learn to aggregate states with the same optimal action. Since these approaches learn the structure without relying on prior knowledge, they typically require large amounts of data, limiting their application to *transfer learning* (Taylor and Stone, 2009), where abstractions learned in one task can be used to speed learning in other, related tasks. By contrast, our methods are effective in an on-line setting by exploiting because they exploit prior knowledge about irrelevant components.

Abstractions with context-specific structure are related to hierarchical learning. Dietterich (2000) presents a method, called MAX-Q, which decomposes a task is into subtasks which are solved separately. It has the advantage that a subtask does not have to use all state components. Instead, it can only use the components that are relevant to the specific subtask, reducing the overall problem size. A context-specific abstraction can also achieve this problem reduction by dividing the task into regions with different relevant components. A difference is that for MAX-Q the solution of a subtask is independent of other subtasks. This has a number of advantages. For example, the policy of a subtask will converge more quickly and it is easier to apply transfer learning. A disadvantage of MAX-Q is that in general only *recursively optimal* solutions can be guaranteed. By contrast, the solution of an abstraction with context-specific structure is globally optimal (assuming each 'region' contains all the local relevant components). The options framework (Sutton *et al.*, 1999) also provides task hierarchy. Options combine multiple primitive actions (actions that finish after a single time step) into extended actions that control the agent for multiple time steps until some termination condition is met. By adding options to a task, while leaving in the primitive actions, hierarchical optimality of the solution can be guaranteed in the limit, instead of only recursive optimality. The downside of leaving in the primitive actions is that the overall problem size increases instead of decreases.

3. MARKOV DECISION PROCESSES

In this section, we discuss the standard framework for describing MDPs and factored MDPs and discuss basic solution strategies. But first we discuss some notational conventions used throughout this article.

3.1. Notation

We use capital letters to denote random variables (e.g., X) and small letters to denote their values (e.g., x). We use a calligraphic font to denote a set (e.g., $\mathcal{X}$), and small letters in greek font to denote functions (e.g., μ). We use boldface to denote a multivariate random variable (e.g., $\boldsymbol{X} = (X^1, X^2)$) or product set (e.g., $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \mathcal{X}^2$), and superscripts to denote different elements in a multivariate variable or a product set. Subscripts are reserved to denote the time index of a random variable (e.g., X_t). If P is a distribution or probability measure, then $X \sim P$ means X is a random variable drawn from P. $X \in \mathcal{X}$ means that the values the random variable X can have are the elements from $\mathcal{X}$.

Furthermore, a set-superscript for a vector or product set indicates that only certain components are used. For example, let $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \mathcal{X}^2 \times \cdots \times \mathcal{X}^N = \times_{i=1}^N \mathcal{X}^i$ be a product set, and $\mathcal{S} \subseteq \{1, 2, \ldots, N\}$ be a subset of component indices. Then $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ indicates the product set spanned by the components in $\boldsymbol{\mathcal{X}}$ with an index in $\mathcal{S}$: $\boldsymbol{\mathcal{X}}^{\mathcal{S}} = \times_{i \in \mathcal{S}} \mathcal{X}^i$. For example, $\boldsymbol{\mathcal{X}}^{\{1,3\}} = \mathcal{X}^1 \times \mathcal{X}^3$. In addition, for the multivariate random variable $\boldsymbol{X} = (X^1, \ldots, X^N)$, with $X^i \in \mathcal{X}^i$, $\boldsymbol{X}^{\mathcal{S}} = (X^i : i \in \mathcal{S})$. For example, $\boldsymbol{X}^{\{1,3\}} = (X^1, X^3)$. Similarly, for $\boldsymbol{x} = (x^1, \ldots, x^N) \in \boldsymbol{\mathcal{X}}$, with $x^i \in \mathcal{X}^i$, $\boldsymbol{x}^{\mathcal{S}} = (x^i : i \in \mathcal{S})$.

3.2. Markov Decision Processes

Markov decision processes (MDPs) (Bellman, 1957a) 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 of size $|\mathcal{X}|$, 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 interaction with the environment occurs at discrete time steps $t = \{0, 1, 2, 3, ...\}$. This interaction happens as follows. Let $X_t \in \mathcal{X}$ and $A_t \in \mathcal{A}$ be random variables that denote the state of the environment and the action taken by the agent at time step t. Once the action is selected, it is sent to the environment, which makes the transition:

$$X_{t+1} \sim \tau(X_t, A_t, \cdot) \tag{1}$$

$$R_{t+1} \sim \rho(X_t, A_t, X_{t+1}) \tag{2}$$

After the agent observes the next state X_{t+1} and reward R_{t+1}, the agentstate space chooses a new action $A_{t+1} \in \mathcal{A}$ and the process is repeated.

The agent can select its actions at a certain time step based on the complete history. An MDP $\mathcal{M}$, the action-selection strategy of the agent and some random initial state X_0 together define a random state-action-reward sequence $((X_t, A_t, R_{t+1}); t \geqslant 0)$. For this sequence, the following holds, for all $U \subseteq \mathcal{X} \times \mathbb{R}$:

$$Pr((X_{t+1}, R_{t+1}) \in U | X_t, A_t) = Pr((X_{t+1}, R_{t+1}) \in U | X_t, A_t, R_t, ..., R_1, X_0, A_0)$$

In other words, the future and history are conditionally independent, given the current state and action. This is called the *Markov property.*

The *return* for time step t, G_t, is defined as the discounted sum over future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}\,, \tag{3}$$

where $0 \leqslant \gamma \leqslant 1$ is the *discount factor*. In general, at time step t, an agent will try to select action A_t in such a way that the expected value of G_t is maximized.

Some MDPs 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 the initial state. The infinite sum from Equation (3) 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.

3.3. Factored Markov Decision Processes

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

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

Occasionally, we say something like "for this state, the value of $\mathcal{X}^k$ is 0". This is shorthand for saying "the value of the k-th component of this state, which is an element of state component $\mathcal{X}^k$, is 0".

The *size* of a factored state space, which we indicate by $|\boldsymbol{\mathcal{X}}|$, is the number of distinct component value combinations that can potentially be observed by the agent. This number is in general smaller than the product of the state component sizes (i.e., $|\boldsymbol{\mathcal{X}}| \leqslant \prod_{i=1}^{N} |\mathcal{X}^i|$), because of correlation between component values. For example, a state space spanned by two identical state components of size K, also has a size of K, and not of K^2.

Sometimes the structure of a problem dictates that a state component is only meaningful depending on the value of other state components. For example, suppose we have a component *TransportVehicle* = { *'car', 'boat'* } and a component *WheelSize* = { *'small','large'* }. Because a boat does not have any wheels, the wheel-size component is meaningless in this context. We call a state space where the number of components that describe a state varies across the state space a *context-specific state space*. Because it is conceptually unintuitive to use a state space where the elements are vectors of different size, we will 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. For example, a boat in the state space *TransportVehicle* × *WheelSize* is described by the vector ('boat', #). With this way of modelling context-specific structure, formally, each state has the same number of components. Note that the size of a context-specific state space is not affected by this approach (to see why, enumerate the states for the example above; keep in mind that the size is determined by the number of component-value combinations that can potentially be observed).

Context-specific state spaces are related to hierarchical RL. We discussed this relation in the related work section (Section 2).

3.4. Solution Strategies

Actions are selected according to a *policy*. A stationary, deterministic policy π is a mapping $\pi : \boldsymbol{\mathcal{X}} \to \mathcal{A}$, where $\pi(\boldsymbol{x})$ is the action the agent takes in state $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$. Each policy π is associated with a state-value function $V^\pi : X \to \mathbb{R}$ that maps a state $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ to the expected return from that state when following policy π:

$$V^\pi(\boldsymbol{x}) = \mathbb{E}\big[G_t | \boldsymbol{x}_t = \boldsymbol{x}, \pi\big]$$

Related to the state-value function is the action-value function $Q_\pi : \boldsymbol{\mathcal{X}} \times \mathcal{A} \to \mathbb{R}$, which gives the expected return when taking action a in state $\boldsymbol{x}$ and following policy π thereafter:

$$Q^\pi(\boldsymbol{x}, a) = \mathbb{E}\big[G_t | \boldsymbol{x}_t = \boldsymbol{x}, a_t = a, \pi\big].$$

Terminal states (in episodic MDPs) have by definition a value of 0. This can be related to the infinite sum in Equation 3 by interpreting them as states with only a single action with zero reward that points to themselves.

In a *planning* context, the transition and reward functions of the MDP are known to the agent. The goal in this case is to compute an *optimal policy* π^*, which maximizes the expected return for

each state. In a *reinforcement learning* context, the transition and reward functions are not (fully) known by the agent, preventing direct computation of an optimal policy. Instead, the agent has to interact with the environment to learn about the environment and improve its policy. The goal of the agent in an RL setting is to maximize its on-line performance, i.e., the return accrued while learning. Underlying the RL problem is the *exploration/exploitation dilemma*: the agent can either exploit its current knowledge by taking the action that is best according to its current value estimates, or it can explore by taking an action currently deemed suboptimal in order to improve the corresponding value estimate.

RL problems are often solved by iteratively improving the state-value or action-value function. This can be done in a *model-based* way(Sutton, 1990; Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002; Kearns and Singh, 2002), by using the interactions with the environment to estimate the transition and reward functions and then computing the optimal state-value or action-value function for the estimated model via off-line planning techniques such as dynamic programming (Bellman, 1957b; Puterman and Shin, 1978). Alternatively, an RL problem can be solved in a *model-free* way, in which case experience is used to directly update the state-values or action-values. A common form for updating the action-values, or Q-values, is:

$$Q(\boldsymbol{x}_t, a_t) \leftarrow (1-\alpha)Q(\boldsymbol{x}_t, a_t) + \alpha\, v_t\,, \tag{4}$$

where α is the learning rate, $\boldsymbol{x}_t$ and a_t are the state and action at time step t, and v_t is the update target. Many different update targets are possible, e.g., a Monte Carlo (MC) update uses the complete return: $v_t = G_t$. *Temporal-difference* (TD) methods (Sutton, 1988) use an update target that is based on the Q-values of other state-actions pairs. An example is the Q-learning (Watkins and Dayan, 1992) update target:

$$v_t = r_{t+1} + \gamma \max_a Q(\boldsymbol{x}_{t+1}, a)\,, \tag{5}$$

where r_{t+1} is the reward received after taking action a_t in state $\boldsymbol{x}_t$. Once the optimal state-value or action-value function has been learned, an optimal policy can easily be derived.

4. ABSTRACTIONS

The size of the state space can grow exponentially in the number of state components. Therefore, when the number of state components is large, the state space can become prohibitively large. To overcome this, the agent can choose to ignore certain components, for example, those that it knows are irrelevant. Ignoring certain state components is an example of an *abstraction*. More generally, an abstraction is a function that maps states from one state space to states from a different state space. Effectively, an abstraction defines a different task that the agent can interact with. Under certain conditions, this task also obeys the Markov property, i.e., it forms an MDP by itself, in which case standard RL methods can be used to solve it. If this is the case, we call the abstraction a *Markov abstraction*. In this section, we prove that Markov abstractions can be constructed by removing certain types of state components.

4.1. Markov Abstractions

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

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

Before we define what a *Markov* abstraction is, we reiterate what the Markov property of an MDP is based on. In Section 3.2, we showed that an MDP $\mathcal{M} = (\boldsymbol{\mathcal{X}}, \mathcal{A}, \tau, \rho)$, an action-selection strategy based on history values, and an initial state $\boldsymbol{X}_0$ together define an (infinite) sequence of random variables of the form:

$$(\boldsymbol{X}_0, A_0, R_1, \boldsymbol{X}_1, A_1, R_2, \dots)\,,$$

with $\boldsymbol{X}_t \in \boldsymbol{\mathcal{X}}$, $A_t \in \mathcal{A}$ for $t \geqslant 0$ and $R_t \in \mathbb{R}$ for $t \geqslant 1$. The Markov property is defined based on this sequence.

Given $\mathcal{M}$, an action-selection stategy, an initial state $\boldsymbol{X}^0$, *and* an abstraction μ, we can define

a different sequence of random variables.

$$(\boldsymbol{Y}_0, A_0, R_1, \boldsymbol{Y}_1, A_1, R_2, \dots)\,, \tag{6}$$

with $\boldsymbol{Y}_t = \mu(\boldsymbol{X}_t)$ for $t \geqslant 0$, and $\boldsymbol{X}_t$ and R_t for $t \geqslant 1$ generated according to equations (1) and (2). A_t is determined by the agent's action-selection strategy and the (abstracted) history: $(\boldsymbol{Y}_0, A_0, R_1, \dots, R_t, \boldsymbol{Y}_t)$. We call μ a Markov abstraction if the Markov property applies to the sequence specified by (6).

Definition 1: The abstraction $\mu : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{Y}}$ is a Markov abstraction if for the sequence of random variables defined by (6) the following holds, for all $U \subseteq \boldsymbol{\mathcal{Y}} \times \mathbb{R}$:

$$Pr((\boldsymbol{Y}_{t+1}, R_{t+1}) \in U | \boldsymbol{Y}_t, A_t) = Pr((\boldsymbol{Y}_{t+1}, R_{t+1}) \in U | \boldsymbol{Y}_t, A_t, R_t, ..., R_1, \boldsymbol{Y}_0, A_0)\,.$$

In this article, we only consider two types of abstractions. The first type corresponds with ignoring certain components. In other words, it is an abstraction of the form:

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

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 $\mathcal{S} = \{1, 3\}$, $\mu(\boldsymbol{x}) = (3, 8)$.

The second type of abstraction is a mapping to a context-specific state space (see Section 3.3), where different states are described by different state components of $\boldsymbol{\mathcal{X}}$. This state space is spanned by all the components of $\boldsymbol{\mathcal{X}}$, with for each component the element # added, which value indicates the component is not active. An example of such a mapping 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, $\mu(\boldsymbol{x}) = (3, \#, 8, \#, \#)$. We call an abstraction that maps to a context-specific state space a *context-specific abstraction*.

Next, we show how Markov abstractions can be constructed by ignoring certain component types.

4.2. State Component Types

In this section, we define several types of state components. We use a dynamic Bayesian network (Dean and Kanazawa, 1989; Murphy, 2002), shown in Figure 1, as a running example to illustrate the different types. At the end of this section, we show that a Markov abstraction can be constructed by ignoring certain component types.

Throughout this section, we consider the MDP $\mathcal{M} = (\boldsymbol{\mathcal{X}}, \mathcal{A}, \tau, \rho)$, with $\boldsymbol{\mathcal{X}}$ consisting of N state components: $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times ... \times \mathcal{X}^N$. In addition, $\mathcal{S} \subseteq \{1, 2, ..., N\}$ is a subset of component indices. $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ is the state space spanned by the components of $\boldsymbol{\mathcal{X}}$ with an index in the set $\mathcal{S}$, and $\boldsymbol{X}^{\mathcal{S}}$ is a vector of random variables corresponding with the components specified by $\mathcal{S}$. The definitions are specified for $\mathcal{X}^k$, the k-th state component of $\boldsymbol{\mathcal{X}}$.

We start by defining irrelevant state components.

Definition 2: $\mathcal{X}^k$ is irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, if $k \notin \mathcal{S}$ and for all $U \subseteq \boldsymbol{\mathcal{X}}^{\mathcal{S}} \times \mathbb{R}$ the following holds:

$$Pr((\boldsymbol{X}^{\mathcal{S}}_{t+1}, R_{t+1}) \in U \mid \boldsymbol{X}^{\mathcal{S}}_t, A_t) = Pr((\boldsymbol{X}^{\mathcal{S}}_{t+1}, R_{t+1}) \in U \mid \boldsymbol{X}^{\mathcal{S}}_t, X^k_t, A_t)\,, \tag{7}$$

Informally, an irrelevant state component is a component that affects neither the next value of any component in $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, nor the reward. In Figure 1, $\mathcal{X}^1$ is irrelevant with respect to $\mathcal{X}^3 \times \mathcal{X}^4$ because it affects neither those components nor the reward. Similarly, $\mathcal{X}^3$ is irrelevant with respect to $\mathcal{X}^1 \times \mathcal{X}^2 \times \mathcal{X}^4$ because it affects only itself. The complement class is the class of *relevant* state components:

Definition 3: $\mathcal{X}^k$ is relevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, if it is not irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$.

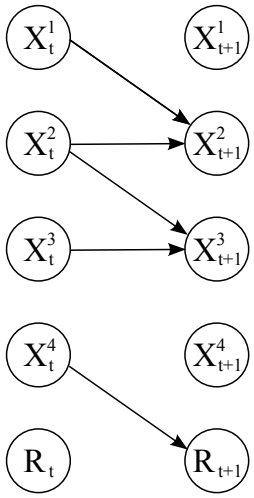

FIGURE 1. A DBN of the transition dynamics of a factored MDP with state space $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \mathcal{X}^2 \times \mathcal{X}^3 \times \mathcal{X}^4$, illustrating various state component types. $X_t^i, X_{t+1}^i \in \mathcal{X}^i$ for $1 \leqslant i \leqslant 4$.

In Figure 1, $\mathcal{X}^1$ is relevant for $\mathcal{X}^2 \times \mathcal{X}^3 \times \mathcal{X}^4$ because it affects $\mathcal{X}^2$. Furthermore, $\mathcal{X}^4$ is relevant for all product sets $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, because it affects reward.

We can divide the irrelevant state component class into three subclasses: *constant*, *empty* and *redundant* components.

Definition 4: $\mathcal{X}^k$ is constant, if $|\mathcal{X}^k| = 1$.

A constant component is a state component that never changes value. It is therefore irrelevant for all $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ with $k \notin \mathcal{S}$.

Definition 5: $\mathcal{X}^k$ is empty, if $|\mathcal{X}^k| > 1$ and $\mathcal{X}^k$ is irrelevant for all product sets $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, with $k \notin \mathcal{S}$.

An empty state component is a component that only affects itself. In Figure 1, $\mathcal{X}^3$ is an example of either a constant or an empty component (depending on whether its value stays constant or not).

Definition 6: $\mathcal{X}^k$ is redundant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, if $\mathcal{X}^k$ is irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, but there exists a product set $\boldsymbol{\mathcal{X}}^{\mathcal{S}'}$ with $\mathcal{S}' \subset \{1, 2, ..., N\}$ and $k \notin \mathcal{S}'$ for which $\mathcal{X}^k$ is relevant.

A redundant component $\mathcal{X}^k$ either affects the value of a component not part of $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ or it affects a component of $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, but $\mathcal{X}^k$ is fully correlated with some other component(s) of $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, hence adding $\mathcal{X}^k$ to $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ does not affect the predictions. In this last case, by removing components from $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$, $\mathcal{X}^k$ could become relevant with respect to this reduced product set.

Apart from the relevant/irrelevant classification, we define another classification: *dependent* and *independent* state components.

Definition 7: $\mathcal{X}^k$ is independent if for all $U \subseteq \mathcal{X}^k$ the following holds:

$$Pr(X_{t+1}^k \in U) = Pr(X_{t+1}^k \in U \mid R_{t+1}, \boldsymbol{X}_t, A_t)\,. \tag{8}$$

Thus, the value of an independent component does not depend on the values of previous state components or the reward just received. Note that an independent component *can* affect the next value of other components or the next reward. In Figure 1, $\mathcal{X}^1$ and $\mathcal{X}^4$ are independent because no variables affect them.

As we prove in the next subsection, an independent state component is unique in the sense that it can contain relevant information, but omitting it still gives a Markov abstraction. Therefore, standard RL methods still converge when using such an abstraction, though the resulting policy is

not optimal for MDP $\mathcal{M}$. However, since we are primarily interested in the best online performance instead of the optimal policy, omitting independent components can play an important role in finding an efficient abstraction.

For completeness, we also define the counterpart of an independent state component:

Definition 8: $\mathcal{X}^k$ is *dependent* if it is not independent.

In Figure 1, $\mathcal{X}^2$and $\mathcal{X}^3$ are dependent because their values at time step $t+1$ depend on the values from time step t: $\mathcal{X}^2$ depends on the values of $\mathcal{X}^1$ and $\mathcal{X}^2$ at the previous time step, and $\mathcal{X}^3$ depends on the values of $\mathcal{X}^2$ and $\mathcal{X}^3$.

The following theorem shows that a Markov abstraction can be constructed by removing certain components from $\boldsymbol{\mathcal{X}}$. We prove this theorem in Appendix A.

Theorem 1: Consider the MDP $\mathcal{M} = (\boldsymbol{\mathcal{X}}, \mathcal{A}, \tau, \rho)$ with $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \mathcal{X}^2 \times ... \times \mathcal{X}^N$. Abstraction $\mu(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}}$ with $\mathcal{S} \subseteq \{1, 2, ..., N\}$ is a Markov abstraction if each component $\mathcal{X}^k$ from $\boldsymbol{\mathcal{X}}$ with $k \notin \mathcal{S}$ is either irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ or an independent component.

Note that using a Markov abstraction to learn a task, guarantees that the performance will converge using basic RL method. However, it does not guarantee that the asymptotic performance is equal to the asymptotic performance of an agent using all state components. If relevant, independent components are ignored, the performance may be lower.

To illustrate this, consider a navigation task where the agent has to reach a goal location. The task is deterministic when two state components are used: one specifying the position and one relevant, independent binary component. This binary state component has a value of zero 90% of the time, but once in a while, 10% of the time, its value changes to one. The effect of this component having value 1 is that all action effects are opposite (i.e., a 'left' action results in a 'right' movement, and so on). An agent that includes this binary component in its state abstraction is able to counter this effect. On the other hand, an agent that does not includes this component perceives the environment as stochastic, causing the selected action to have an opposite effect than expected 10% of the time. Since the agent cannot compensate for this effect, its best policy will have an expected return that is less than the best policy of the agent that uses both state components.

5. ABSTRACTION SELECTION FOR CONTEXTUAL BANDIT PROBLEMS

In this section, we introduce and evaluate abstraction-selection algorithms for a *contextual bandit problem* (Wang *et al.*, 2005; Pandey *et al.*, 2007), which can be viewed as a special case of an MDP that only consists of single-action episodes. Contextual bandit problems are a useful way to model many real-world tasks, e.g., selecting ads to place alongside web pages (Langford and Zhang, 2007; Langford *et al.*, 2008). In this section, we discuss the simplest form of abstraction selection, where a single abstraction is used for the full state space. In Section 7.4 we discuss the case of contextual bandit problems with context-specific structure.

5.1. Contextual Bandit Problems

A *multi-armed bandit problem* (Lai and Robbins, 1985; Auer *et al.*, 2002) is a task in which a choice has to be repeatedly made between the same set of actions. The action produces a reward drawn from an unknown probability distribution corresponding to that action, and the goal is to maximize the total reward over a series of action selections. The term 'multi-armed bandit' is derived from the analogy to a slot machine (traditionally called a 'one-armed bandit') with multiple arms instead of one.

A *contextual multi-armed bandit problem* is an extension of the multi-armed bandit problem for which the reward distributions of the bandit arms are correlated with observed context information. This context information is generated by a fixed probability distribution and changes after each arm pull. This problem maps to an RL problem by interpreting the arms as actions and the context information as states. The task involves learning the average reward of each arm conditioned on the

context information. The optimal policy is simply a policy that selects at each moment the arm with the highest average reward given the current context.

A (contextual) multi-armed bandit problem can be viewed as a special case of an episodic MDP problem: each episode ends after only a single action. For a regular multi-armed bandit problem the initial state is always the same, while for a contextual multi-armed bandit problem, the initial state is drawn from a (fixed) probability distribution over states. Note that the Markov property always holds for a (contextual) multi-armed bandit problem, since there is no history to consider.

5.2. Abstraction Selection

Our approach for abstraction selection is to evaluate the various candidate abstractions by learning with them and measuring the average reward accrued. A key insight behind our approach is that an agent trying out different abstractions in a task faces a similar exploration/exploitation dilemma as an agent choosing between regular actions in such a task. As a result, the choice of which abstraction to use can be modelled as an action internal to the agent. We call such actions *switch actions*. Before we explain our abstraction selection strategy, we would like to motivate why abstraction selection is important with an example.

5.2.1. *Motivating Example.* To see the potential benefits of abstraction selection, consider an ad-placement problem. Because companies that serve ads are typically paid per click, the goal is to select the ads that maximize the chance of being clicked. This task can be modelled as a contextual bandit problem wherein available ads are actions, web pages are states, and rewards are payments for clicked ads.

Typically, the web page is described using a large set of components. These can include the frequency of each term in the web page, a categorization of the page (e.g., news, entertainment, shopping), the number of incoming or outgoing links, the length of the URL, etc. Since the size of the state space depends critically on the number of state components used, selecting a good abstraction is essential. The chosen abstraction must be rich enough to allow the system to determine what ad to place and yet be small enough to make learning feasible.

Suppose that a large set of low-level components is used to construct a set of ten abstract, high-level components, each of which can take on five values. This yields $5^{10} \approx 10^7$ states. One option would be for an agent to try to learn an accurate action-value function for each state-action pair. However, doing so for such a large state space would be immensely challenging. Instead, the system designer could try to select the most useful components. For example, if three components are chosen, the size of the state space is only 125. However, due to the unpredictability of user behaviour, selecting the right components would require enormous domain expertise.

Alternatively, the designer could opt to have the agent learn by itself what the best combination of three components is by using an abstraction selection strategy. There are $\binom{20}{3} = 1140$ candidate abstractions consisting of three components, each consisting of $5^3 = 125$ states. Hence, the total number of states the agent has to consider is $1140 * 125 \approx 1.4 \times 10^5$, which is two orders of magnitude smaller than the number of states of a state space based on all components.

5.2.2. *The Abstraction-Selection Task.* We now formally describe the derived task that models the ability to select different abstractions. We call this derived task the *abstraction-selection task*. We define it in a general way, so it can serve as a stepping stone for the domain of episodic MDPs, discussed in Section 6. At the end of this section, we prove that the abstraction-selection task of a contextual bandit is always Markov. Hence, it can be solved with standard RL techniques. Note that the abstraction-selection task of a contextual bandit task is itself not a contextual bandit task. Instead, it is small episodic task with exactly two actions per episode, the first action being a switch action and the second action being a regular action.

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 $\boldsymbol{\mu} = \{\mu^1, \ldots, \mu^K\}$. 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 that indicates which of the candidate abstractions is 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 $\mathcal{X}^{abs} = \{0, 1, \ldots, K\}$. The values in $\mathcal{X}^{abs}$ refer 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:

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

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:

$$\mathcal{A}^+(\boldsymbol{x}) = \begin{cases} \{a^{sw,1}, \ldots, a^{sw,K}\} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ \mathcal{A} & \text{if } \boldsymbol{x}^{\{abs\}} \neq 0\,, \end{cases} \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+. \tag{9}$$

The effect of taking switch action $a^{sw,i}$ is that the value of component $\mathcal{X}^{abs}$ is set to $x^{abs,i}$ for $1 \leqslant i \leqslant K$. 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, the context-specific abstraction μ^+ has to be applied to $\mathcal{M}^+$, which is defined as:

$$\mu^+(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\{abs\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ (\boldsymbol{x}^{\{abs\}}, \mu^i(\boldsymbol{x}^{\{1,\ldots,N\}})) & \text{if } \boldsymbol{x}^{\{abs\}} = i, 1 \leqslant i \leqslant K \end{cases} \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+. \tag{10}$$

The optimal policy of the abstraction-selection task yields the best switch action, i.e., the best abstraction, as well as the optimal policy for each abstraction. The following theorem proves that the abstraction-selection task is in fact Markov, allowing the use of standard RL methods for solving it.

Theorem 2: The abstraction-selection task of a contextual bandit problem obeys the Markov property.

Proof. To prove that the abstraction-selection task is Markov, we have to prove that $\mathcal{M}^+$ is Markov and that μ^+ is a Markov abstraction for $\mathcal{M}^+$. That $\mathcal{M}^+$ is Markov can be checked easily, because it contains all the state components from MDP M and the dynamics of the switch actions are not affected by the history. For μ^+ to be a Markov abstraction, the transition probabilities for the abstract states should be independent of all possible histories. An episode of the abstraction-selection task consists of two actions: a switch action followed by a regular action. The switch action always obeys the Markov property since there is no history yet. On the other hand, because the history of a regular action is always the same for a given state, this action also always obeys the Markov property. Hence, abstraction μ^+ is Markov, and therefore the abstraction-selection task as well. □

5.2.3. *Example.* Consider a simple contextual bandit problem with two actions, a^1 and a^2, and a state space consisting of four states described 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 states have the same probability of occurring. Action a^1 always produces a reward of 0, while the expected value of the reward for action a^2 depends on the state, as 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 $\sum_{x^1 \in \mathcal{X}^1, x^2 \in \mathcal{X}^2} P_0(x^1, x^2) \cdot \max_a \rho((x^1, x^2), a) = 1.5$. Although the state space of this task is small enough to use both components, for illustrative purposes we assume that the agent must choose

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

between using either component $\mathcal{X}^1$ or component $\mathcal{X}^1$. That is, 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 $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$.

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 as follows:

$$\mu^+(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\{abs\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ \boldsymbol{x}^{\{abs,1\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 1 \\ \boldsymbol{x}^{\{abs,2\}} & \text{if } \boldsymbol{x}^{\{abs\}} = 2\,, \end{cases} \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+. \tag{11}$$

The complete MDP resulting from applying μ^+ to $\mathcal{M}^+ = (\boldsymbol{\mathcal{X}}^+, \mathcal{A}^+, \tau^+, \rho^+)$ is visualized in Figure 2. The value shown behind each action identifier is the expected value of the reward when taking that action from the corresponding state. The reward after a switch action is always zero. For actions a^1 and a^2 the expected reward can be derived from Table 1. For example, the expected value of $\rho(\boldsymbol{X}, a^2)$ given $X^1 = \mathit{true}$ is $(0.25 * 4 + 0.25 * 2)/0.5 = +3$. On the other hand, the expected value of $\rho(\boldsymbol{X}, a^2)$ given $X^2 = \mathit{true}$ is only $(0.25 * 4 + 0.25 * (-2))/0.5 = +1$. Overall, abstraction μ^1 is the better choice, because selecting it results in an expected reward of +1.5, which is same as the expected reward when no abstraction is used. By contrast, selecting abstraction μ^2 results in an expected reward of only +0.5.

5.3. Model-Free Updating

In this section, we present an update scheme for the abstraction-selection task of a contextual bandit problem. Since the state space of the abstraction-selection task can be exponentially smaller than that of the original MDP, using this task can greatly reduce the computational, memory, and sample requirements of learning. Furthermore, since the abstraction-selection task obeys the Markov property, it is itself an MDP. Consequently, standard TD methods can be used to solve it.[2]. However, in this section we show how the special properties of the abstraction-selection task can be exploited to speed learning even further. In particular, since the agent can observe all components, even those that are not part of the currently selected abstraction, it can construct a *parallel experience sequence* for each abstraction that is not selected. This parallel sequence can be used, under certain conditions, for efficient *off-policy* updating.

5.3.1. *Updating Switch Actions.* An episode of the abstraction-selection task consists of a switch action followed by a regular action. Since a regular action always results in a terminal state, its Q-value is simply the expected immediate reward, which can be trivially estimated by averaging the corresponding observed sample rewards. However, estimating the Q-values of switch actions is less straightforward.

Before describing how to do this, we illustrate why learning such Q-values is necessary. After all,

[2]In this article, we focus on model-free updates. But model-based methods could also be applied.

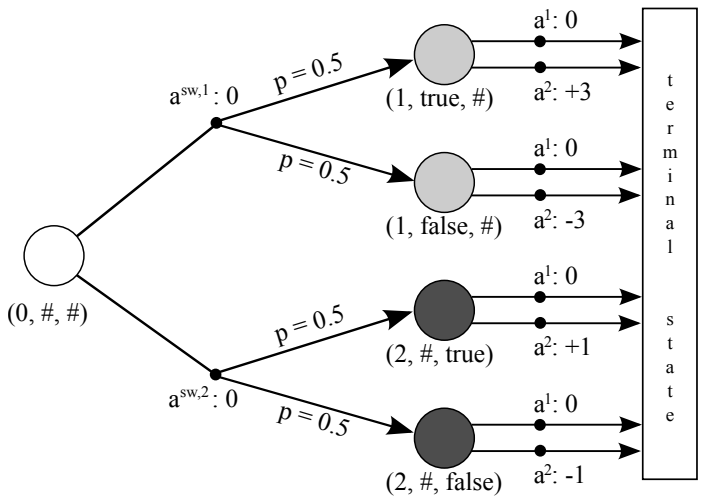

Figure 2. MDP resulting from applying μ^+ to $\mathcal{M}^+$ for the contextual bandit task described in Table 1. Circles indicate states. If states have the same colour, they use the same candidate abstraction. The small black dots indicate actions. Actions with stochastic transitions have multiple arrows. 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.

because switch actions are internal actions, the agent already knows what the next state will be before it takes a switch action. Thus, it may seem that learning Q-values for switch actions is unnecessary since they could simply be inferred from the next state. To see why such a strategy fails, consider the example presented in Section 5.2.3. If the current state for MDP $\mathcal{M}$ is $(X^1 = false, X^2 = true)$, abstraction μ^1 predicts action a^1 is best, yielding a reward of 0. On the other hand, abstraction μ^2 predicts action a^2 is best, yielding an average reward of +1 (see Figure 2). If abstractions selection was done merely on the basis of the state values of each abstraction, then abstraction μ^2 would be chosen, since it predicts a higher reward. However, for state $\mathcal{M}$ is $(X^1 = false, X^2 = true)$ a^1 is actually the best action (see Table 1). The problem is that this strategy implicitly uses both components to select the switch action while the abstraction uses only a single component, resulting in a non-Markov task.

Thus, learning separate Q-values for the switch actions is essential. However, it is not obvious what the best way to do this is. Using a simple Q-learning update is not ideal because it bootstraps the value of the switch action with values of the corresponding candidate abstraction. If this candidate abstraction is large and has optimistically initialized Q-values, then the Q-value of the switch action will have a positive bias for a long time, even if the abstraction itself is poor. To get an estimate of how good a candidate abstraction is more quickly, a Monte Carlo update can be used, which updates a Q-value with the complete observed return and is therefore not biased by values of the candidate abstraction.

The regular Monte Carlo update is an *on-policy update*: the policy the agent follows (the *behaviour policy*) is the same as the policy whose Q-values are being estimated (the *estimation policy*). Alternatively, an *off-policy update* can be performed, in which case the behaviour and estimation policies are different. For example, Sutton and Barto (1998) present an off-policy MC update that evaluates the greedy policy, while following an ϵ-greedy behaviour policy (i.e, one that selects the greedy action with $1 - \epsilon$ probability, while selecting a random action otherwise).

The main disadvantage of off-policy Monte Carlo updates is that learning is inefficient for long episodes because a state-action pair can be updated only when all subsequent actions are greedy, which occurs only rarely given an ϵ-greedy behaviour policy. Fortunately, the derived task of a contextual bandit problem has episodes that are only 2 actions long, reducing the off-policy MC update to a particular simple form. In Appendix B, we show that for the abstraction-selection task of a contextual bandit problem, the off-policy MC update reduces to one based on the immediate reward of the regular action under the condition that this regular action is the greedy action (with

respect to the Q-value estimates):

$$Q(\boldsymbol{x}_0, a_0) = (1-\alpha)\, Q(\boldsymbol{x}_0, a_0) + \alpha\, r\,, \qquad \text{if 'regular action is greedy'}\,, \tag{12}$$

where $\boldsymbol{x}_0$ is the initial state, r is the reward after the regular action, and a_0 is the first action selected in the abstraction-selection task, which is always a switch action. The condition that the regular action should be greedy is required to ensure that the update is unbiased.

5.3.2. *Off-Policy Updating Regular Actions.* The concept of off-policy updating can be used for more than merely evaluating the greedy policy while using an ϵ-greedy behaviour policy. In fact, we can also use off-policy updating to simultaneously update the state-action pairs of candidate abstractions that are not currently selected, which can speed learning considerably.

Since the agent observes all the available components, it can construct a parallel experience sequence for each candidate abstraction that was not selected. For example, consider the abstraction-selection task from Section 5.2.3, and assume the current state in $\mathcal{M}$ is $(X^1 = true, X^2 = false)$, and the action taken by the agent is $a^{sw,1}$. The experience sequence from the abstraction-selection task observed by the agent is:

$$(0, \#, \#) \rightarrow a^{sw,1} \rightarrow (1, true, \#) \rightarrow a \rightarrow r\,, \tag{13}$$

where a is the regular action taken by the agent, and r is the resulting reward. Even though the agent did not take action $a^{sw,2}$, it can deduce what would be the resulting next state, because this state is based on X^2, which it can also observe. The agent can therefore construct a parallel experience sequence corresponding to abstraction μ^2:

$$(0, \#, \#) \rightarrow a^{sw,2} \rightarrow (1, \#, false) \rightarrow a \rightarrow r\,. \tag{14}$$

This parallel sequence can be used to update the regular action a for abstraction μ^2 as well as the switch action $a^{sw,2}$.

That said, the updates performed using such a sequence may be biased, since regular actions for abstraction μ^2 are updated but action selection is based on abstraction μ^1. This bias is a consequence of the fact that state $(2, \#, true)$ is actually an aggregation of two states in the original state space $\mathcal{X}^1 \times \mathcal{X}^2$: $(X^1 = false, X^2 = true)$ and $(X^1 = true, X^2 = true)$. The expected reward for action a^2 in state $(2, \#, true)$ is a weighted average of the expected rewards of these two underlying states. Since abstraction μ^1 aggregates the states from original state space in a different way, these states correspond to two different states in abstraction μ^1. Therefore, if the selection probability of a^2 is different for those states, the rewards are not properly weighted to correctly estimate the expected reward.

To avoid this problem, the agent can perform such off policy-updates only under certain conditions that guarantee no bias will be introduced. The following theorem establishes those conditions. Note that the theorem is stated in terms of an MDP, of which a contextual bandit problem is a special case.

Theorem 3: Consider MDP $\mathcal{M}$ with state space $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \cdots \times \mathcal{X}^N$ and the candidate abstractions $\mu^1(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}_1}$ and $\mu^2(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}_2}$, $\mathcal{S}_1, \mathcal{S}_2 \subseteq \{1, \ldots, N\}$, which are both Markov abstractions for M. Assume the agent selects abstraction μ^1. Then an unbiased, single-step update for abstraction μ^2 can be performed, based an the parallel experience sequence, if either one of the following two conditions hold:

(1) $\mathcal{S}_1 \subseteq \mathcal{S}_2$
(2) If the action was an exploratory action under an exploration scheme that does not depend upon the specific state (e.g., an ϵ-greedy exploration scheme).

Proof. Bias is introduced if a group of states from $\boldsymbol{\mathcal{X}}$ map to a single state in $\boldsymbol{\mathcal{X}}^{\mathcal{S}_2}$, but to multiple states in $\boldsymbol{\mathcal{X}}^{\mathcal{S}_1}$, while the action outcomes of these multiple states are incorrectly weighted. Under the first condition, a single state in $\boldsymbol{\mathcal{X}}^{\mathcal{S}_2}$ always corresponds to a single state of $\boldsymbol{\mathcal{X}}^{\mathcal{S}_1}$, and therefore no incorrect weighting can occur. For the second condition, recall that since $\boldsymbol{\mathcal{X}}^{\mathcal{S}_2}$ is a Markov abstraction, components with an index in $\mathcal{S}_1$ but not in $\mathcal{S}_2$ are either independent components or components that are irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}_2}$. Components that are irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}_2}$ result in states with equal one-step models, therefore the action outcomes can never be incorrectly weighted. States based on different

values of an independent component occur with a probability that scales with the component value probability, since the agent has no control over the value of independent components. Therefore, if the exploration scheme does not depend on the state, a particular action will also be selected with a probability that scales with the component value probability, hence correctly weighting the action outcomes. □

Thus, when one of the conditions of Theorem 3 holds, the reward can be used to update the regular action of the unselected abstraction.

5.3.3. *Off-Policy Updating Switch Actions.* To be able to perform an unbiased Monte Carlo update based on the parallel experience sequence, all transitions involved must be unbiased. In the previous section, we showed that an update of the regular action is unbiased if one of the conditions of Theorem 3 holds. However, the switch action update is always unbiased because the state to which it belongs is the same for all switch actions. Therefore, when one of the conditions of Theorem 3 holds, an unbiased Monte Carlo update can also be performed for the corresponding switch action. In other words, both the switch and regular actions of a candidate abstraction can be updated without selecting that abstraction. As a result, in contextual bandit problems, abstractions can be fully evaluated in an off-policy manner. An immediate consequence is that the agent can use greedy action selection for the switch actions, since exploring the switch actions is unnecessary.

Table 2 summarizes our action selection and update strategy for the abstraction-selection task of the contextual bandit problem. In the next section, we illustrate the performance improvements that this update scheme makes possible.

Table 2. Action selection and update strategy for the abstraction-selection task of a contextual bandit problem. The 'if subset/on explore' condition refers to the two conditions of Theorem 3.

	action selection	update, current abs.	update, other abs.
regular action	ϵ-greedy	average reward	if subset/on explore: average reward
switch action	greedy	MC update	if subset/on explore: MC update

5.4. Experimental Results

In this section, we present two experiments involving abstraction selection for a contextual bandit problem. For both experiments we use the action selection and update strategy described in Table 2. In the first experiment, we consider a contextual bandit problem with n state components, where only one of the components is relevant, while the others are empty. The agent knows this, but does not know which component is relevant. Hence, there are n candidate abstractions, one for each component. We compare switching between these abstractions with using all components for $n = 3$ and $n = 4$. To demonstrate that our switching method can deal with huge state spaces, we also show the performance for switching with $n = 50$. Due to the size of the state space, evaluating the performance of using all components is infeasible for this setting. Finally, as an upper bound, we show the performance for using only the relevant component.

Each of the n components has eight values, which are initialized randomly after each arm pull. The bandit has two arms, with opposite expected rewards: one arm has an expected reward of $+1$, and the other -1. Each reward is drawn from a normal distribution with a standard deviation of 2. Which arm has the positive reward depends on the context. For half of the values of the relevant component, the first arm has the $+1$ expected reward. Therefore, when this component is ignored, the expected reward is zero.

For the switching method, we use a step-size of 0.01 for the switch action, and an ϵ of 0.2 for ϵ-greedy selection of the regular action. To kick-start the switch method, we use $\epsilon = 1.0$ for the first 50 arm pulls. Since all candidate abstractions are updated during this exploration phase, this has a positive effect on the overall performance.

Figure 3 shows the results, averaged over 10,000 independent runs and smoothed by additional averaging over intervals of 10 arm pulls. The performance of the switch method illustrates how

effectively it can exploit prior knowledge about the set of candidate abstractions. While the size of the full abstraction grows exponentially in the number of components (512 states for a set of 3 components and 4096 for a set of 4 components), the total number of states for the switching methods grows only linearly (24 states for a set of 3 components and 32 states for a set of 4 components). Simultaneously updating the abstractions increases the performance even further, making it nearly indistinguishable for $n = 3$ and $n = 4$ from the agent that knows in advance which candidate abstraction is correct. For $n = 50$, the performance is only slightly lower, demonstrating that the switching method can tackle huge problems (note that the state space size of the full abstraction is 8^{50} in this case).

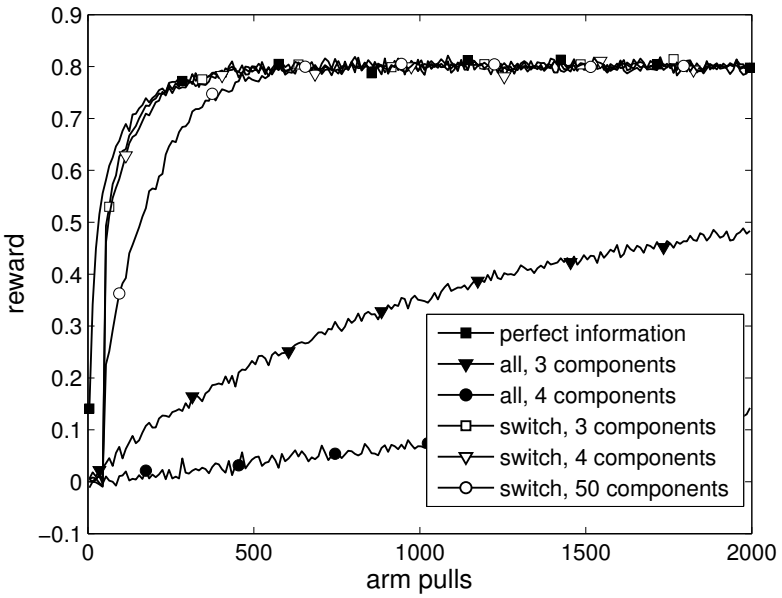

FIGURE 3. The performance of the switch method on a contextual bandit problem with in total 3, 4 or 50 components (for the original MDP), compared to a naïve method that uses all the available components and one that knows in advance which candidate abstraction is correct ('perfect information').

The second experiment is a variation on the first. The state space consists of three components: $\mathcal{X}^1 \times \mathcal{X}^2 \times \mathcal{X}^3$. Each component has eight values and is initialized randomly (with equal probability for each value) after each arm pull. This time, however, there is not a single relevant component; instead, all three components contain some relevant information. From Table 3, which shows the corresponding rewards and probabilities, it follows that the expected reward of the optimal policy for this abstraction is +1. The size of the state space of this abstraction is $8^3 = 512$.

Table 4 is deduced from Table 3 and shows the initial probability distribution for X^1 as well as the expected reward conditioned on X^1. While all three state components are necessary to achieve an expected reward of +1, using only component X^1 results in an expected reward that is only slightly less than using all components, while the state space size is considerably smaller (8 states). We compare the performance of 1) learning with an abstraction containing only component $\mathcal{X}^1$ ('small abstraction'), 2) learning with an abstraction containing all three components ('large abstraction'), and 3) using the switch method given both abstractions as candidates ('switching'). We use a learning rate of 0.001 for the switch action and an ϵ of 0.2 for the ϵ-greedy selection of the regular action for all methods. The switch method uses an ϵ of 1.0 for the first 100 arm pulls.

Figure 4 shows the average reward for the first 20,000 arm pulls, averaged over 10,000 independent runs and smoothed by additional averaging over intervals of 10 arm pulls. For approximately the first 5000 arm pulls, the small abstraction outperforms the large one since it learns more quickly. However, since its abstraction does not contain all relevant information, it plateaus below the final

TABLE 3. Expected rewards and initial state probability P_0 for $\boldsymbol{X} = (X^1, X^2, X^3) \in \mathcal{X}^1 \times \mathcal{X}^2 \times \mathcal{X}^3$. The true/false labels are derived from the real component values: for X^1 and X^2, 4 of the 8 values correspond to a 'true' label, while the other values correspond to a 'false' label. For X^3, 6 of the 8 components correspond to a 'true' label, while the other values correspond to a 'false' label.

X^1	X^2	X^3	$P_0(\boldsymbol{X})$	$\mathbb{E}[\,\rho(\boldsymbol{X}, a^1)\,]$	$\mathbb{E}[\,\rho(\boldsymbol{X}, a^2)\,]$
true	true	true	0.1875	+1	-1
true	true	false	0.0625	-1	+1
true	false	true	0.1875	+1	-1
true	false	false	0.0625	+1	-1
false	true	true	0.1875	-1	+1
false	true	false	0.0625	+1	-1
false	false	true	0.1875	-1	+1
false	false	false	0.0625	-1	+1

TABLE 4. Initial probability P_0 for X^1 and expected reward conditioned on X^1

X^1	$P_0(X^1)$	$\mathbb{E}[\,\rho(\boldsymbol{X}, a^1)\|X^1\,]$	$\mathbb{E}[\,\rho(\boldsymbol{X}, a^2)\|X^1\,]$
true	0.5	0.75	-0.75
false	0.5	-0.75	0.75

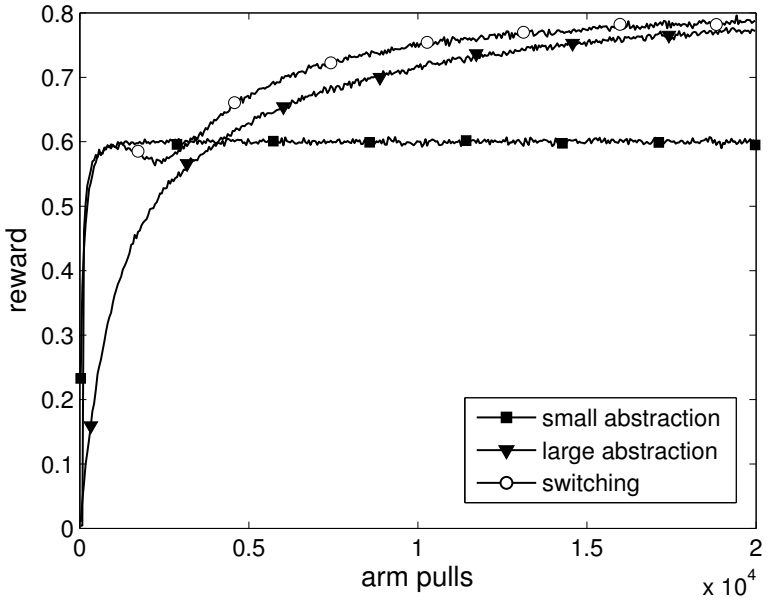

FIGURE 4. Average reward for a contextual bandit when switching between a large and a small abstraction.

performance of the large abstraction. After the initial exploration phase, the switch method quickly catches up with the small abstraction after which the performance shows a small dip before climbing above that of REP-LARGE.

We explain this dip as follows: while the small abstraction is quickly recognized as the better one initially, the agent uses off-policy updating to keep improving the Q-values of both the large abstraction and the switch actions. In some runs, once the Q-value of the switch-action for the large abstraction approaches that of the small abstraction, the estimates will prematurely indicate that the large one is better, causing a small dip in the performance. After the dip, the Q-values estimates

of the large abstraction improve further, exceeding those of the small abstraction and causing the second climb in performance.

Interestingly, the switch method outperforms the large abstraction at each point during learning. The reason is that the exact point where large abstraction outperforms the small one is slightly different for each run. Since the switch method uses an up-to-date estimate of the expected reward for each abstraction, it simply makes longer use of the small abstraction for those runs where the Q-values of the large abstraction improve more slowly than average. Therefore, once the Q-values of the small abstraction have been properly learned, the performance of the small abstraction forms a lower bound for the remaining arm pulls. This lower bound is not present for the large abstraction, whose performance is bounded only by zero, leading to lower average performance.

The results of this experiment underscore an important advantage of the switching method. Because it evaluates candidate abstractions on-line, it can automatically identify, not only the best abstraction to use in the long run, but also the best one to use *during learning*. For example, the small abstraction, although in the long run inferior to the large one, is preferable early in learning when insufficient data is available for the large abstraction to be effective. The switch method gets the best of both worlds, mimicking the performance of the small abstraction early in learning and that of the large abstraction later on.

6. ABSTRACTION SELECTION FOR EPISODIC MDPS

In the previous section, we showed that a contextual bandit problem with abstraction selection can be modelled as an MDP with a context-specific state space. The solution of this derived MDP not only yields the best candidate abstraction for the contextual bandit problem, but also the optimal policy of that abstraction. In this section, we discuss abstraction selection for episodic MDPs. As with a contextual bandit problem, abstraction selection for an episodic MDP can be modelled by introducing switch actions. In this case, after taking a switch action to select an abstraction, the agent takes not one but a series of regular actions until a terminal state is reached. The regular actions are chosen based on the Q-values of the selected abstraction.

The definition of the abstraction-selection task of an episodic MDP is similar to that of a contextual bandit problem. However, we show that the conditions under which this task is Markov are more restrictive. In Section 7 we discuss abstraction selection for general MDPs with context-specific structure.

6.1. Abstraction-Selection Task

The abstraction-selection task of an episodic MDP is constructed in exactly the same way as for a contextual-bandit problem (see Section 5.2.2), because we formalized a contextual-bandit task as an episodic MDP with only a single action.

Although the construction process is the same, the resulting abstraction-selection task is of course not the same. Specifically, the abstraction-selection task of a contextual-bandit problem consists a selecting a switch action, and then selecting a single regular action, which terminates the episode. In case of an episodic MDP the switch action can be followed by a long series of regular actions before the episode terminates. Note that the abstraction is always selected at the start of an episode and never changes within an episode.

Because the abstraction-selection task of an episodic MDP has longer episodes, an additional condition is required to ensure that the abstraction-selection task is Markov.

Theorem 4: The abstraction-selection task of an episodic MDP $\mathcal{M}$ obeys the Markov property if all candidate abstractions are Markov abstractions with respect to $\mathcal{M}$.

Proof. An episode of the abstraction-selection task of an episodic MDP starts with a switch action, selecting a candidate abstraction, and then a series of regular actions until a terminal state is reached. The switch action always obeys the Markov property since there is no history yet. The history of the first regular action is always the same for a given state. Hence, it always obeys the Markov property

as well. For the regular actions after that the Markov property follows from the condition that all candidate abstractions are Markov. □

The reason that the condition of Markov abstractions is not required for the abstraction-selection task of a contextual-bandit problem is that for a contextual-bandit problem every abstraction is always Markov, because there is no history to consider. Hence, this condition can be ignored.

6.2. Model-Free Updating

In this section, we discuss two different update strategies. As before, for simplicity, we restrict ourselves to model-free learning. The first strategy adapts the update scheme used for contextual bandit problems to the MDP setting. For this strategy, the regular actions are updated with Q-learning updates. As with contextual bandit problems, a parallel experience sequence is created for each candidate abstraction. In the MDP setting, this parallel sequence is used to perform off-policy Q-learning updates. These updates are applied to the regular actions of all unselected candidate abstractions for which at least one of the conditions of Theorem 3 holds, guaranteeing that the updates are unbiased.

As before, a Monte Carlo (MC) update is used for the switch action. However, the off-policy MC update described in Section 5.3.1 is inefficient for the MDP case, since it can only be performed when all actions following the switch action are greedy. Therefore, we use an on-policy MC update instead. Updating the switch actions of other abstractions with an MC update requires that all actions are unbiased, which is not the case in general. Therefore, such updates are not performed. A consequence is that each switch action needs to be explored in order to obtain accurate Q-value estimates for that action. We call the resulting update scheme, summarized in Table 5, the *Monte Carlo update scheme* since a Monte Carlo update is used for the switch action.

Table 5. Monte Carlo update scheme. The "if subset/on explore" condition refers to the two conditions of Theorem 3.

	action selection	update, current abs.	update, other abs.
regular action	ϵ-greedy	Q-learning	if subset/on explore: Q-learning
switch action	ϵ^{sw}-greedy	MC-update	-

The second strategy uses a Q-learning update for the regular actions as well as for the switch actions. The update of a switch action is performed for *all* candidate abstractions *before* selecting the switch action, since the agent can already observe the result of a switch action before taking it. Performing the update before selecting the action is preferable, since the selection is then based on more accurate Q-values. We call this technique *just-in-time* (JIT) updating, because the updates are performed right before the Q-values are needed (for action selection). Since the regular actions as well as the switch action of a candidate abstraction can be updated without selecting it, exploring the switch actions is unnecessary and thus the agent can always choose the greedy switch action.

We call the resulting update scheme, summarized in Table 6, the *Q-learning update scheme* since a Q-learning update is used for the switch action.

Table 6. Q-learning update scheme. The "if subset/on explore" condition refers to the two conditions of Theorem 3.

	action selection	update, current rep	update, other reps
regular action	ϵ-greedy	Q-learning	if subset/on explore: Q-learning
switch action	greedy	JIT Q-learning	JIT Q-learning

Each scheme has its advantages and disadvantages. On the one hand, the MC update scheme has switch actions that are not biased by the values of the candidate abstractions; therefore, it is expected to more quickly learn an accurate estimate of an abstraction's value. On the other hand,

it does not update the switch actions off-policy; therefore, more exploration is required. The Q-learning update scheme can perform off-policy updates of the regular actions as well as of the switch actions, allowing greedy selection of the switch action. However, the values of the switch actions are bootstrapped from the abstractions. Thus, each candidate abstraction needs to be sufficiently explored before its switch action has an accurate value. In the next section, we empirically compare both update schemes.

6.3. Experimental Results

In this section, we present experimental results evaluating both the Monte Carlo and Q-learning update schemes described above. These experiments are conducted on an MDP we call the *Mars rover task*. Suppose a rover on a Mars mission must frequently navigate between its home base and a research site. The area the rover must cross can be described by a 15×15 square grid, with the home base and research site in opposite corners. The rover observes its current position and has four movement actions: *north*, *south*, *east* and *west*, which, on a regular surface, cause a movement of one square in the corresponding direction. However, on the sandy soil of Mars, the rover's action outcomes are heavily distorted. The effect of the distortion can be modelled as an additional (clockwise) rotation of either 0, 90, 180 or 270 degrees applied before the directional movement. A *north* action can for example lead to an *east* movement if the distortion is 90 degrees.

The distortion is affected by the local sand structure of the rover's current location, which consists of a number of ditches of different sizes. These ditches are described by a set of structural state components that the agent observes along with its position component (see Figure 5). Each structural component corresponds to a different ditch size, while its value indicates the number of ditches of that size on the local square. The structure is not static, but changes each time the rover enters a square, according to some probability distribution, due to the interaction between the rover and the sand.

By learning the relationship between the structural properties and the distortion for a certain square, the rover can compensate for the distortion. However, learning with all structural components is prohibitively slow. Fortunately, experiments on Earth showed that only ditches of one size cause the distortion, i.e., only one structural component is relevant. The grain size of the sand, which cannot be observed, determines which of the structural components is relevant. The rover uses abstraction selection to determine what the relevant component is.

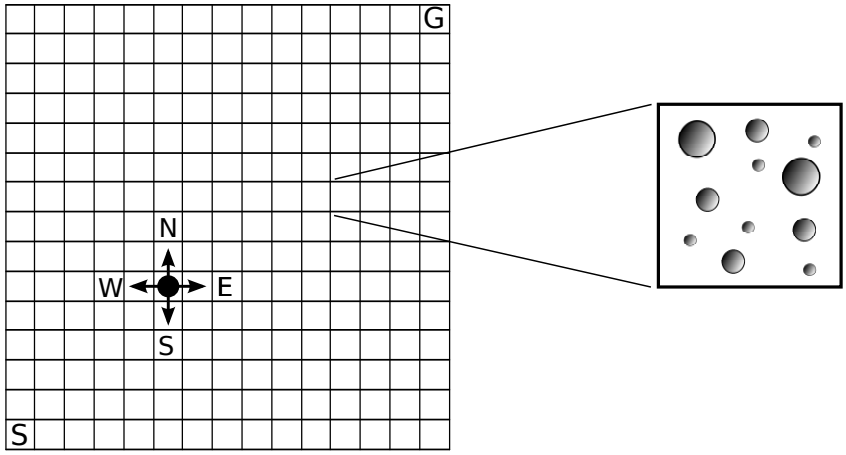

FIGURE 5. Mars rover task: the agent must move from S to G using four directional actions: (N)orth, (S)outh, (E)ast and (W)est. The local sand structure, consisting of ditches of different sizes, causes a distortion of the regular action outcome. If the agent learns the relationship between the local sand structure and the distortion, it can compensate for it and thus reach the goal location more quickly.

We perform experiments where the local sand structure is described by 5 components and by

30 components. In each case, we assume the components are independent and consist of 4 values each, with equal probability. Together with the position component, these (full) state space consists of $2 \cdot 10^5$ states and $3 \cdot 10^{20}$ states, respectively. Each of the 4 values of the relevant component corresponds with a different distortion value (0, 90, 180 or 270 degrees). In addition to the distortion caused by the local sand structure, there is a 10% chance on an additional distortion, modelled as another rotation of 0, 90, 180 or 270 degrees (each value has equal probability).

Under these settings, the transition probabilities conditioned on only the position component (i.e., ignoring the relevant structural component) are the same for all actions. Consequently, with only the position component, the agent cannot learn an effective policy, since all actions have the same effect and the agent moves randomly through the grid, until it lands on the goal state. When the agent does consider the relevant component, it can learn to compensate for the distortion caused by the sand structure, causing near-deterministic action outcomes (the additional 10% distortion cannot be compensated for).

Our experiments compare the performance of the Monte Carlo and Q-learning abstraction selection algorithms described in Section 6.2 with both 5 and 30 structural components. We also compare against an algorithm that uses perfect information, i.e., that uses only the position component and the relevant structural component. As in the experiments presented in Section 5.4, this algorithm provides an upper bound on performance. In addition, to get lower bounds, we compare against an algorithm that uses only on the position component and an algorithm that uses all components. The latter algorithm is evaluated only in the setting with 5 structural components since doing so with 30 components is infeasible.

For each algorithm, we measure the average return over the first 1000 episodes. All algorithms use ϵ-greedy selection for the regular action with $\epsilon = 0.1$, and a step-size with an initial value α_0 of 1.0 that is decayed according to:[3]

$$\alpha(\boldsymbol{x}, a) = \alpha_0 \, d^{\, n(\boldsymbol{x}, a)} \,, \tag{15}$$

where $n(\boldsymbol{x}, a)$ is the number of times action a was previously selected in state $\mathbf{x}$. We optimize the decay rate d for each method. For the switch methods, the extra parameters are also optimized. The range for which parameters are optimized is determined by performing some initial experiments to find roughly the settings with the best performance. For the Q-learning update scheme, we use full exploration for the first z episodes (the value of z is optimized). We do not do this for the Monte Carlo update scheme since the initial experiments showed that this extra parameter causes negligible performance improvement. All results are averaged over 200 independent runs and smoothed.

Table 7 shows the average performance of the different methods over the first 1000 episodes together with the optimal parameters, while Figure 6 plots the average return (from the start of an episode) as function of the number of episodes for these optimal parameters. As predicted, when the structural components are ignored, no learning occurs since all actions have the same expected outcome. The agent moves randomly through the environment, generating large negative reward. By contrast, learning does occur when all 5 structural components are used. However, because of the size of the resulting state space, learning is slow and the performance improvement is marginal.

In the setting with 5 structural components, both of the switching methods perform much better than when using all components, generating up to 9 times less negative reward. Comparing the two switching methods with each other reveals that the Q-learning update scheme outperforms the Monte Carlo update scheme, generating 1.3 times less negative reward. Apparently, the advantage of the Q-learning update scheme (more off-policy updates resulting in less exploration) outweighs its disadvantage (bootstrapping from the abstractions values).

In the setting with 30 structural components, both switching methods perform worse than in the setting with 5 structural components, which can be expected, since the set of candidate abstractions is larger (30 instead of 5). The relative performance advantage of the Q-learning update scheme compared to the Monte Carlo update scheme is much larger than in the setting with 5 structural components. This can be explained by the fact that more off-policy learning occurs under the Q-

[3]This type of decay does not meet the requirements for convergence in the limit of many TD algorithms (Jaakkola *et al.*, 1994; Singh *et al.*, 2000). However it gives good results in practice and is very easy to implement.

learning update scheme. With more candidate abstractions, the effect of off-policy learning increases, resulting in a higher performance advantage for Q-learning.

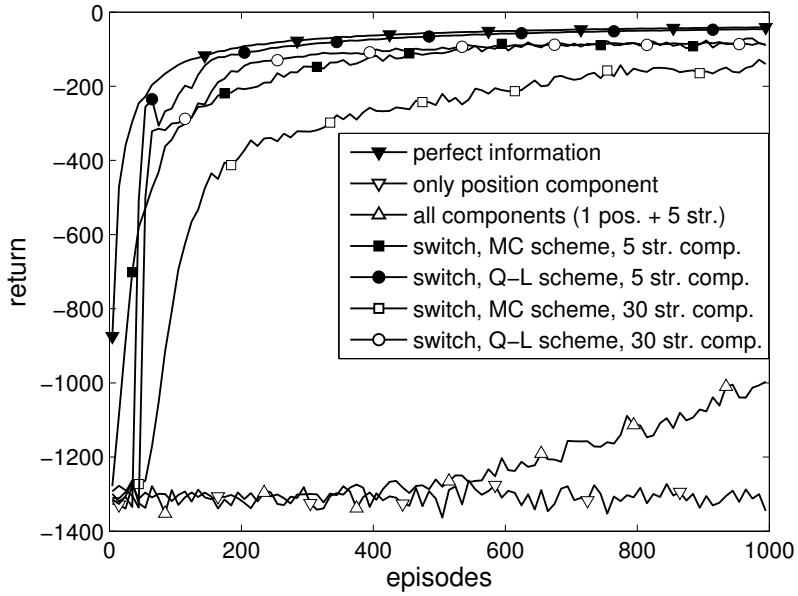

FIGURE 6. Performance as a function of number of episodes on the Mars rover task.

TABLE 7. Average performance over the first 1000 episodes and optimal parameters on the Mars rover task.

	optimal parameters	average return	standard error
perfect information	d = 1%	-87.75	0.05
only position component	(no learning occurs)	-1308	2
all components (1 pos. + 5 str.)	d = 1%	-1226	1
switch, MC scheme, 5 str. comp.	$\epsilon^{sw} = 0.04$, $d^{sw} = 1\%$, $d^{reg} = 2\%$	-180	7
switch, Q-L scheme, 5 str. comp.	$z = 40$, $d^{sw} = 7\%$, $d^{reg} = 2\%$	-138	1
switch, MC scheme, 30 str. comp.	$\epsilon^{sw} = 0.06$, $d^{sw} = 3\%$, $d^{reg} = 1\%$	-345	17
switch, Q-L scheme, 30 str. comp.	$z = 50$, $d^{sw} = 6\%$, $d^{reg} = 3\%$	-184	14

These results show that tasks with a limited number of candidate abstractions can be efficiently solved using our abstraction-selection task. In the next section, we analyze what happens when the candidate abstractions themselves are context-specific.

7. ABSTRACTION SELECTION FOR GENERAL MDPS WITH CONTEXT-SPECIFIC STRUCTURE

In the previous two sections, we considered tasks for which the best abstraction had globally the same components. However, many real-world problems have a context-specific structure. Consider for example an extension of the Mars rover task from Section 6.3, where the rover encounters different types of sand along its journey. The relevant component can be different for each of these types, resulting in a context-specific abstraction. To find the best abstraction for this task, the candidate abstractions must also be context-specific.

The solution strategy developed in Section 6 can in principle also be applied when the candidate abstractions are context-specific. A problem however is that the number of candidate abstractions can increase exponentially when context-specific structure is introduced. Consider the extended Mars rover task described above with 6 types of sand and choice between 5 different components. If the rover can distinguish between the different sand types, then there are $6^5 = 7776$ possible assignments of relevant components to sand types possible, resulting in 7776 candidate abstractions. When the agent cannot distinguish between the different sand types, things get even worse since the agent must learn the relevant component for each position. With 225 positions (the number of positions from the original Mars rover task), there are $225^5 \approx 6 \cdot 10^{11}$ candidate abstractions. Thus, the strategy from Section 6, which results in an abstraction-selection task whose state space size is linear in the number of candidate abstractions, is no longer feasible.

In this section, we introduce a strategy for context-specific candidate abstractions that results in an abstraction-selection task with an exponentially smaller state space. With this strategy, the $6 \cdot 10^{11}$ context-specific candidate abstractions from the extended Mars rover task can be evaluated without explicit enumeration of each context-specific candidate abstraction. In short, our strategy allows the agent to switch between abstractions in the middle of an episode, instead of just selecting one at the start of an episode. By doing so, a small number of regular abstractions (5 in the case of the extended Mars rover task) can be used to represent and evaluate a large number of context-specific abstractions. In addition, this strategy can deal with MDPs that do not have terminal states, since the agent is not required to reach a terminal state, before it can switch to a new abstraction.

7.1. Modelling Partial Knowledge of the Context-Specific Structure

Before we explain how the abstraction-selection task is constructed, we discuss the problem description. The domain that we are interested in is an MDP with context-specific structure. In other words, an MDP where, in different parts of the state space, different components are relevant. When full knowledge of the context-specific structure is available, this knowledge can be exploited by applying a context-specific abstraction that encodes this knowledge, as explained in Section 3.3. When only partial knowledge about the structure is available, this approach cannot be used. An example of partial knowledge is knowledge that two states share the same relevant state components, without knowing *which* components are relevant. While theoretically each state can have a different set of relevant state components, in practise, many states share the same relevant components. By aggregating states that share the same relevant state components, the state space can be divided into regions, which we call *contexts*. All states that are part of the same context, share the same relevant state components.

In this section, we consider the problem of an MDP with context-specific structure, where the agent has knowledge about the contexts, that is, knowledge about which states share the same relevant state components. However, the agent does not know *which* components are relevant for a specific context.

Consider an MDP with a state space $\boldsymbol{\mathcal{X}}$ that can be divided into M different contexts. Knowledge about these contexts can be modelled with an extra, artificial, state component $\mathcal{X}^c = \{1, 2, \ldots, M\}$ that specifies to which context a state belongs. States with the same value for $\mathcal{X}^c$ share the same relevant state components. Given a set of candidate abstractions $\boldsymbol{\mu} = \{\mu^1, \ldots, \mu^K\}$, we want the agent to learn the best abstraction for each of the M contexts.

In a more general setting, we might want to specify a different subset of candidate abstractions for each of the contexts. To enable this, we introduce the function β that maps each value in $\mathcal{X}^c$ to a subset $\mathcal{B} \subseteq \{1, \ldots, K\}$ of abstraction indices. β defines which candidate abstractions the agent can choose between for each context.

The state component $\mathcal{X}^c$ and the function β enable encoding of two different types of partial knowledge about the structure of an MDP. Knowledge about states sharing the same relevant state components is modelled by $\mathcal{X}^c$ (or more specifically, by the correlation matrix between $\mathcal{X}^c$ and $\boldsymbol{\mathcal{X}}$). On the other hand, knowledge about potentially useful abstractions for a specific context is modelled by β. Note that if no knowledge is available about the structure, $\mathcal{X}^c$ can still be defined. In this case, it assigns a different value to each state. The function β can also still be defined: it assigns the set of all abstraction indices to each context.

7.2. Abstraction-Selection Task

Consider the MDP $\mathcal{M} = (\boldsymbol{\mathcal{X}}, \mathcal{A}, \tau, \rho)$, the state component $\mathcal{X}^c = \{1, 2, \ldots, M\}$ that models the contexts, a set of K candidate abstractions $\boldsymbol{\mu} = \{\mu^1, \ldots, \mu^K\}$, and the function β, which maps each value in $\mathcal{X}^c$ to a subset $\mathcal{B} \subseteq \{1, \ldots, K\}$ of abstraction indices. In addition, consider the *switch abstraction* μ^0, which specifies the state components that are used when no candidate abstraction is selected (its purpose will become clear shortly).

As before, the abstraction-selection task for $\mathcal{M}$ is a task resulting from applying a context-specific abstraction, μ^+, to an extended version of $\mathcal{M}$, $\mathcal{M}^+ = (\boldsymbol{\mathcal{X}}^+, \mathcal{A}^+, \tau^+, \rho^+)$. The state space $\boldsymbol{\mathcal{X}}^+$ extends $\boldsymbol{\mathcal{X}}$ as follows:

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

with $\mathcal{X}^{abs} = \{0, 1, \ldots, K\}$. The values in $\mathcal{X}^{abs}$ refer the indices of the candidate abstractions. The value 0 means that there is currently no candidate abstraction selected (in which case abstraction μ^0 applies). The initial value of $\mathcal{X}^{abs}$ is always 0.

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

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

The switch actions are only available in state $\boldsymbol{x}$, if $\boldsymbol{x}^{\{abs\}} = 0$. Which switch actions are available is determined by the value of component $\mathcal{X}^c$, $\boldsymbol{x}^{\{c\}}$, in combination with the function β:

$$\mathcal{A}^+(\boldsymbol{x}) = \begin{cases} \{a^{sw,i} : i \in \beta\big(\boldsymbol{x}^{\{c\}}\big)\} & \text{if } \boldsymbol{x}^{\{abs\}} = 0 \\ \mathcal{A} & \text{if } \boldsymbol{x}^{\{abs\}} \neq 0\,, \end{cases} \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+. \tag{16}$$

The effect of switch action $a^{sw,i}$ is that the value of $\mathcal{X}^{abs}$ is set to $x^{abs,i}$ for $1 \leqslant i \leqslant K$. The agent receives no reward for taking a switch action, and no other component values are affected. The effect of a regular action on the state components in $\boldsymbol{\mathcal{X}}$ is defined by $\mathcal{M}$. By definition, $\mathcal{X}^c$ fully correlated with $\boldsymbol{\mathcal{X}}$. Hence, the value of $\mathcal{X}^c$ is determined by the values of the state components in $\boldsymbol{\mathcal{X}}$ (from the same time step).

Our strategy for MDPs with context-specific structure is to keep the abstraction the same as long as the agent stays in the same context. Whenever the environment changes context, the agent chooses between the available abstractions for the new context. To enable this, we specify that the value of $\mathcal{X}^{abs}$ is set back to 0, when the agent crosses the border between two contexts (i.e., when it takes a regular action that results in a value change for component $\mathcal{X}^c$). Setting $\mathcal{X}^{abs}$ back to 0, gives the agent the opportunity to select a candidate abstraction for the new context.

So far, no abstraction has been applied yet. To get the abstraction-selection task, the context-specific abstraction μ^+ has to be applied to $\mathcal{M}^+$, which is defined as follows:

$$\mu^+(\boldsymbol{x}) = (\, i,\, \mu^i(\boldsymbol{x}^{\{1,\ldots,N\}})), \text{ with } i = \boldsymbol{x}^{\{abs\}} \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{\mathcal{X}}^+. \tag{17}$$

For states with $\boldsymbol{x}^{\{abs\}} = 0$, the switch abstraction μ^0 is required to ensure convergence of standard RL methods.

The abstraction-selection task defined above is Markov only under strict conditions. We can relax these conditions by requiring a slightly weaker form of the Markov property, which we call *Markov with respect to relevant components*. This property holds when the values of the relevant components and the reward depend only on the current state and action.

Definition 9: A task with state space $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times ... \times \mathcal{X}^N$ is 'Markov with respect to relevant components' if for the sequence of random variables it defines the following holds, for all $U \subseteq \boldsymbol{\mathcal{X}}^{\mathcal{S}} \times \mathbb{R}$:

$$Pr((\boldsymbol{X}^{\mathcal{S}}_{t+1}, R_{t+1}) \in U | \boldsymbol{X}_t, A_t) = Pr((\boldsymbol{X}^{\mathcal{S}}_{t+1}, R_{t+1}) \in U | \boldsymbol{X}_t, A_t, R_t, ..., R_1, \boldsymbol{X}_0, A_0)\,, \tag{18}$$

where $\mathcal{S} \subset \{1, \ldots, N\}$ is a subset of indices, such that for all $k \in \{1, \ldots, N\}$: if $k \notin \mathcal{S}$, then $\mathcal{X}^k$ is irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$.

Because irrelevant state components neither affect the values of other state components, nor the reward, states that differ from each other only in their irrelevant components have the same value. Therefore, the standard RL algorithms converge to the optimal policy of a task, when a task is Markov with respect to the relevant state components. Note that the definition only requires that

there *exists* a Markov abstraction. To obtain convergence, it is not necessary to find and use this abstraction.

The following theorem specifies a set of conditions that guarantees the abstraction-selection task of an MDP with context-specific structure is Markov with respect to the relevant components:

Theorem 5: Consider the task $\mathcal{M}$ with context-specific structure. The state space $\boldsymbol{\mathcal{X}}$ of $\mathcal{M}$ consists of N state components. In addition, consider the candidate abstractions $\{\mu^1, \ldots, \mu^K\}$ and switch abstraction μ^0. The abstractions are defined as follows: $\mu^i(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}^i}$ with $\mathcal{S}^i \subseteq \{1, \ldots, N\}$, for all $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ and $0 \leqslant i \leqslant K$. The abstraction-selection task of $\mathcal{M}$ is Markov with respect to relevant components if the following three conditions hold:

(1) All candidate abstractions are Markov with respect to $\mathcal{M}$.
(2) The switch abstraction μ^0 is Markov with respect to $\mathcal{M}$.
(3) $\mathcal{S}^0 \subseteq \mathcal{S}^i$ for $1 \leqslant i \leqslant K$.

Proof. Let $\mu_t(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}_t}$ be the abstraction at time step t and $\mu_{t+1}(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}_{t+1}}$ be the abstraction at time step $t+1$. Because $\mathcal{S}^0 \subseteq \mathcal{S}_{t+1}$ and because μ^0 is a Markov abstraction (i.e., all components with an index not in $\mathcal{S}^0$ are either independent or irrelevant), $\mathcal{S}_{t+1}$ can be decomposed as $\mathcal{S}_{t+1} = \mathcal{S}^0 \cup \mathcal{S}^{ind} \cup \mathcal{S}^{irr}$, where $\mathcal{S}^{ind}$ are the indices of the independent components, and $\mathcal{S}^{irr}$ are the indices of the components that are irrelevant with respect to $\boldsymbol{\mathcal{X}}^{\mathcal{S}^0}$.

To prove the theorem, we need to prove that Equation (18) holds, for $\mathcal{S} = \mathcal{S}^0 \cup \mathcal{S}^{ind}$. In other words, we need to prove that the components with an index in $\mathcal{S}^0 \cup \mathcal{S}^{ind}$ are only affected by $\boldsymbol{X}^{\mathcal{S}_t}$, and not by the history. The independent components are not affected by the history by definition. On the other hand, the components with an index in $\mathcal{S}^0$ do not depend upon the history, because μ^0 is a Markov abstraction. □

7.3. Model-Free Updating

The abstraction-selection task described in the previous subsection can be solved using update schemes similar to those for the abstraction-selection task of an episodic MDP with regular candidate abstractions (see Section 6.2). The only difference arises from the fact that now a state with switch actions can be revisited multiple times before a terminal state is reached. Therefore, using a MC update for the switch actions is less suitable, since it is an off-line update that misses out on the opportunity to update the Q-values during the episode. Hence, the MC update is replaced with an *n-step update*, which accumulates the reward received after taking the switch action until the agent re-enters a state with switch actions, i.e., until the context changes. This n-step update still has the advantage that it does not bootstrap from the Q-values of an abstraction. However, it is also an on-line update, i.e., it is performed *during* an episode. We will refer to this update scheme as the *n-step update scheme*. Besides this update scheme, the *Q-learning update scheme* from Section 6.2 can be applied. This update scheme can be applied without making any modifications.

We summarize the update schemes for updating the Q-values of the abstraction-selection task of an MDP with context-specific structure in Table 8 and Table 9.

Table 8. n-step update scheme.

	action selection	update, current abs.	update, other abs.
regular action	ϵ-greedy	Q-learning	if subset/on explore: Q-learning
switch action	ϵ^{sw}-greedy	n-step update	-

In Section 7.5, we compare these update scheme experimentally on an extension of the Mars rover task.

TABLE 9. Q-learning update scheme.

	action selection	update, current abs.	update, other abs.
regular action	ϵ-greedy	Q-learning	if subset/on explore: Q-learning
switch action	greedy	JIT Q-learning	JIT Q-learning

7.4. Contextual Bandit Problems with Context Specific Structure

Because a contextual bandit problem is a special case of an MDP, the definition of the abstraction-selection task in Section 7.2 is also applicable to contextual bandit problems with context-specific structure. In addition, due to the simplicity of this domain, this task is Markov under minimal conditions, as shown by the following theorem.

Theorem 6: Consider the contextual-bandit problem $\mathcal{M}$ with context-specific structure. The state space $\boldsymbol{\mathcal{X}}$ consists of N state components. In addition, consider the candidate abstractions $\{\mu^1, \ldots, \mu^K\}$ and switch abstraction μ^0. The abstractions are defined as follows: $\mu^i(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}^i}$ with $\mathcal{S}^i \subseteq \{1, \ldots, N\}$, for all $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ and $0 \leqslant i \leqslant K$. The abstraction-selection task of $\mathcal{M}$ is Markov if $\mathcal{S}^0 \subseteq \mathcal{S}^i$ for $1 \leqslant i \leqslant K$.

Proof. The proof closely follows the arguments as in the case of contextual bandit problems without context-specific structure (Theorem 2). The Markov property states that the transition probabilities and reward for a state-action pair should be independent of all possible histories. An episode of the abstraction-selection task consists of two actions: a switch action followed by a regular action. The switch action always obeys the Markov property since there is no history yet. In addition, because, $\mathcal{S}^0 \subseteq \mathcal{S}^i$ for $1 \leqslant i \leqslant K$, the history of a regular action is always the same, for any state $\boldsymbol{x}$ in $\boldsymbol{\mathcal{X}}^+$. Therefore, also this action always obeys the Markov property. □

The update schemes for this abstraction-selection task are the same as the update scheme for the abstraction-selection task of a regular contextual bandit problem (Table 2).

7.5. Experimental Results

In this subsection, we consider an extension of the Mars rover task of Section 6.3. In this extension, the area the rover has to cross to get from the start state to the goal state is divided into 4 types of sand (see Figure 7). Positions with the same sand type have the same relevant structural component, but the relevant component can differ for each sand type.

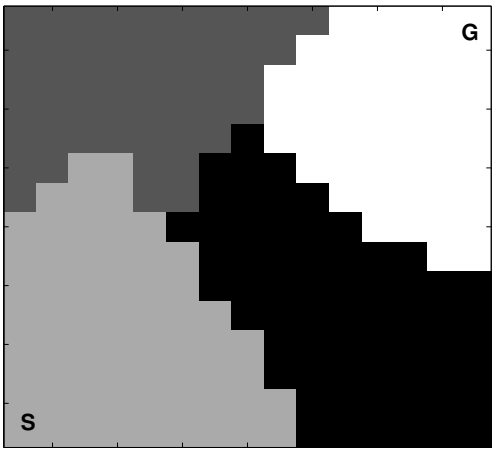

FIGURE 7. The extended Mars rover task: the rover must move from S to G while crossing different types of sand, each of which can have a different relevant structural component.

We consider two scenarios: in the first, the agent observes the different sand types (e.g., by

observing the sand color) while in the second scenario it does not. When the agent observes the sand types, it selects a new abstraction only when it crosses the border between sand types. In the second scenario, the agent selects a new abstraction after each regular action. We compare the two switching strategies from Section 7.3 on both these scenarios.

All switch methods use five candidate abstractions. Each candidate abstraction is based on two components: the position component and one of the the five structural components. The switch abstraction only uses the position component. We compare the switch methods with a method having perfect information, a method that uses all components, and a method that uses only the position component.This results in the following lists of methods:

- Perfect information - Single, context-specific, abstraction consisting of only the position component and the locally relevant structural component.
- All structural components - Single abstraction taking into account the position component and all structural components.
- Only position component - Single abstraction taking into account only the position component.
- Switch, n-step scheme, border - Abstraction selection with n-step update scheme and switching only when the border between sand types is crossed.
- Switch, n-step scheme, always - Abstraction selection with n-step update scheme and switching after each regular action.
- Switch, Q-L scheme, border - Abstraction selection with Q-learning update scheme and switching only when the border between sand types is crossed.
- Switch, Q-L scheme, always - Abstraction selection with Q-learning update scheme and switching after each regular action.

We measure the average return over the first 1000 episodes for these seven methods. All methods use ϵ-greedy action selection for the regular action with $\epsilon = 0.1$ and decaying learning rates (according to Equation 15) with initial values of 1 and an optimized decay rate, d^{reg}. For the switch algorithms, the extra parameters are also optimized. All results are averaged over 200 independent runs and smoothed. At the start of each run, a random assignment of relevant components to sand types is made.

Table 10 shows the average performance of the different methods over the first 1000 episodes together with the optimal parameters, while Figure 8 plots the return as a function of the number of episodes for these optimal parameters. Note that the perfect information method, the method that uses all structural components, and the method using only the position component have the same performance as in the regular Mars rover task (see Figure 6). The reason for this is different for each method. For the perfect information method, it does not matter which structural component is relevant, since it uses the right one by definition. For the method using all structural components, it does not matter since the relevant is always observed and the total number of candidate abstractions is still five. For the method using only the position component, the performance is the same since it never uses the relevant component.

All switch methods show a large performance improvement compared to using all components. Surprisingly, the n-step method with switching after each regular action performs remarkably well. It yields an average reward of -123.6, while the other switch methods yield rewards between -240 to -280.[4] This result is remarkable because this method does not require the agent to distinguish the different sand types. The methods that do require this extra knowledge (the switch methods that only let the agent select a new abstraction when the border between sand types is crossed) perform worse.

Apparently, the combination of the n-step update scheme with a low value of n (in our case, $n = 2$) is quite powerful. This can be explained as follows. The n-step update scheme has the same advantage as the MC update scheme from Section 6.2: the update targets for the switch actions are not biased by the Q-values of the candidate abstractions. A disadvantage of an MC update or an n-step update with high values of n is that the variance of the update target is very high, since many

[4] Around episode 300, the n-step method even yields a higher return than the perfect information graph. However, note that the overall performance is worse at each point in time, since the return during the first 200 episodes is much lower.

different state-action sequences can cause the update. When n is small, the variance of the update target is smaller, yielding more accurate updates. Thus, the property of not bootstrapping from the abstractions Q-value seems powerful, an effect that was obscured in our previous experiments by the high variance of the MC update targets.

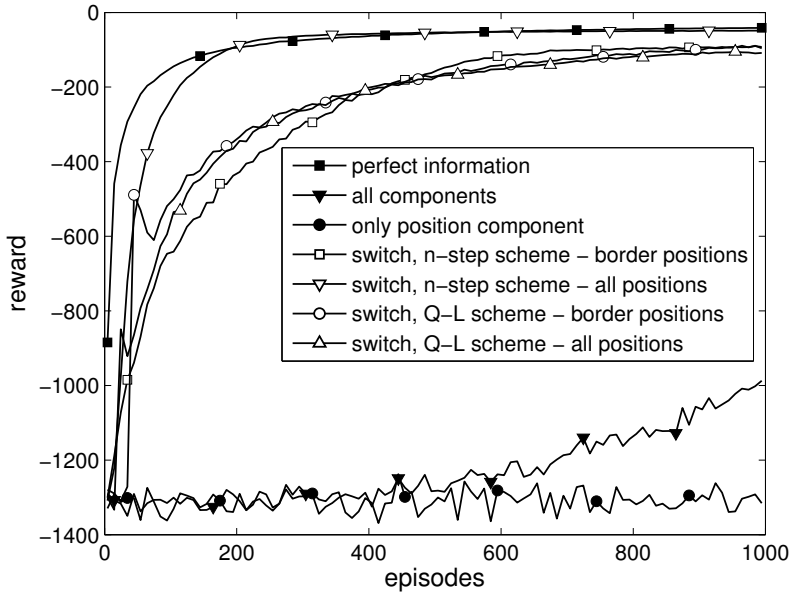

FIGURE 8. Performance as a function of number of episodes on the extended Mars rover task.

TABLE 10. Average performance over the first 1000 episodes and optimal parameters on the extended Mars rover task.

	optimal parameters	average return	standard error
perfect information	$d^{reg} = 1.0\%$	-87.83	0.06
switch, n-step scheme, border	$\epsilon^{sw} = 0.5$, $d^{sw} = 0.4\%$, $d^{reg} = 0.8\%$	-278.9	0.6
switch, n-step scheme, always	$\epsilon^{sw} = 0.2$, $d^{sw} = 0.4\%$, $d^{reg} = 0.8\%$	-123.6	0.2
switch, Q-L scheme, border	$z = 40$, $d^{sw} = 0.6\%$, $d^{reg} = 0.4\%$	-242.3	0.2
switch, Q-L scheme, always	$z = 20$, $d^{sw} = 0.6\%$, $d^{reg} = 0.2\%$	-270.0	0.2
all components	$d^{reg} = 1.0\%$	-1227	1
only position component	(no learning occurs)	-1309	2

8. DISCUSSION AND FUTURE WORK

In this section, we summarize the empirical results of the different sections, discuss the consequences of non-Markov candidate abstractions and provide avenues for future work.

8.1. General

Taken together, the empirical results presented in Sections 5, 6, and 7 provide substantial evidence of the importance of prior knowledge that can be expressed as a set of candidate abstractions. Furthermore, they consistently validate the benefit of using abstraction-selection tasks to exploit such knowledge. In contextual bandit problems, episodic MDPs, and general MDPs with context-specific structure, the methods we propose perform much better than alternatives that do not exploit

such prior knowledge. In addition, they often perform nearly as well as methods that are given the optimal abstraction in advance.

For MDPs with context-specific structure, our methods can also exploit a second form of prior knowledge. In addition to knowledge about the set of candidate abstractions, they also can exploit knowledge about states sharing the same relevant components. This knowledge is represented by the state component $\mathcal{X}^c$, which labels each state with a context identifier. Surprisingly, the extended Mars rover experiment presented in Section 7.5 shows that, when the different contexts use the same candidate abstractions, this additional prior knowledge does not further improve performance. On the contrary, Figure 8 shows that the best performance is obtained by switching at all positions, i.e., ignoring the context information. It is an open question at this point if for certain domains, like the one considered, knowledge about the context is never useful, or if alternative methods exist that can exploit the context in a useful way. Nevertheless, it is not hard to imagine domains where context knowledge would clearly improve performance. For example, in cases where the set of candidate abstractions is different for each context. To illustrate this, suppose each sand type in the extended Mars rover task had a different set of 5 candidate abstractions. In this case, an agent that is ignorant of the sand type would have to explore $5 \times 4 = 20$ candidate abstractions at each position, instead of 5.

Because the best performance in the extended Mars rover task is obtained when switching occurs at all positions, these results also demonstrate that it can be beneficial to switch between abstractions even within the same context. While these experiments evaluate only MDPs with context-specific structure, the conclusion can be applied to all MDPs, since those without context-specific structure are just a special case. Consequently, we hypothesize that the performance on the regular Mars rover task (see Section 6.3) can be further improved, by using the abstraction-selection task presented in Section 7.2 in combination with the n-step update scheme (Table 8). In fact, performance on this task should be the same as on the extended Mars rover task, since the two tasks are actually identical from an algorithmic point of view. In both cases, the agent must learn which component out of a set of 5 components is relevant for each position. We intend to investigate such variations in future work.

8.2. Non-Markov Candidate Abstractions

In this article we focus on abstraction-selection tasks that are Markov, since this results in convergence guarantees of basic RL methods. For the contextual bandit problem, every set of candidate abstractions results in a Markov task (see Theorem 2). For contextual bandit problem where the candidate abstractions share the same context-specific structure, the only condition to achieve a Markov abstraction-selection task is that the components used for the switch abstraction are also part of the component sets of the candidate abstractions (Theorem **??**). This condition can be easily met by adding the components from the switch abstraction to components sets used by the candidate abstractions. Obtaining a Markov abstraction-selection task for a general MDP is less straightforward. In the case of an episodic MDP, all candidate abstractions have to be Markov (Theorem 4). For MDPs with context-specific structure the switch abstraction must also be Markov (Theorem 5). This can be a serious constraint. It is likely, however, that good empirical results can be obtained even when this condition is not fully satisfied, i.e., when the set of candidate abstractions contains non-Markov abstractions.

When a candidate abstraction is not Markov, the expected return depends on the history. This results in incorrect Q-values for this candidate abstraction. When the Q-values of the switch actions are bootstrapped from those of the candidate abstractions, as occurs in the Q-learning update scheme (Table 6), such incorrect values can cause incorrect values for the switch actions, resulting in incorrect abstraction selection. On the other hand, when a Monte Carlo update scheme (Table 5) is used, switch actions are updated by the complete return produced by an abstraction and hence incorrect values of a candidate abstraction have no direct effect. A candidate abstraction that is not Markov will likely produce a bad policy, and consequently a bad return. Hence the corresponding switch action gets updated with low values and the agent will learn to avoid these abstractions.

In the case of abstraction selection for MDPs with context-specific structure, an analysis of the consequences of non-Markov candidate abstractions is less straightforward. However, it is again likely

that the Q-learning update scheme will result in incorrect abstraction selection, for the same reason as mentioned above. On the other hand, with the the n-step update scheme (Table 8), the Q-values of switch actions are not bootstrapped from Q-values from a candidate abstraction. Therefore, we expect that non-Markov candidate abstractions can be dealt with effectively in this setting also. However, we do expect that the switch abstraction has to be Markov in order to achieve effective abstraction selection. Confirming these hypotheses experimentally is a topic for future research.

8.3. Additional Future Work

In addition to the possibilities mentioned above, there are two additional avenues for future work. In this article, we focussed on model-free methods, to get maximum efficiency in terms of space and computation time. In future work, the abstraction-selection task could be combined with model-based learning. This will increase the space requirements (storing the transition table of a candidate abstraction scales quadratically with its size), but allows the required computation time to be traded for improved performance, for example by controlling the number of updates per time step in *prioritized sweeping* (Moore and Atkeson, 1993), which focuses its computation on the updates expected to have the largest effect. Alternatively, by combining the framework with *best-match learning* (van Seijen *et al.*, 2011), the space requirements can also be tuned to optimally exploit the available resources for improving performance.

Another direction of future work is to combine the abstraction-selection task with function approximation. This would further improve the sample efficiency through generalization and make the approach applicable to domains with continuous state spaces. We expect that the conditions for off-policy learning (Theorem 3) will still be Markov when function approximation is applied.

9. CONCLUSIONS

This article presented a new strategy for on-line abstraction selection for factored MDPs. The proposed approach addresses a special case of the structure learning problem in which prior knowledge can be used to restrict the set of candidate abstractions that must be considered. The problem of abstraction selection was formalized by defining an abstraction-selection task that extends the action set with internal switch actions that select the abstraction to be used for regular (external) action selection. We proved that this abstraction-selection task is Markov (or 'Markov with respect to relevant components') under various conditions related to the type of components that are ignored by the candidate abstractions. This result enables the use of resource-efficient model-free learning methods, while still guaranteeing convergence. In addition, we demonstrated that the learning speed can be further improved by constructing parallel experience sequences corresponding to candidate abstractions that are not selected. These parallel sequences can be used for off-policy updating of the Q-values of these abstractions.

We demonstrated the validity of the approach via experiments on a contextual bandit problem, an episodic MDP, and a general MDP with context-specific structure. In all three domains, a large performance improvement was achieved by automatically discovering the best candidate abstractions. Furthermore, we demonstrated that switching between a large abstraction, and a smaller abstraction that is based on a subset of components from this large abstraction can outperform both individual abstractions. The reason is that by switching between them the learning speed of the small abstraction can be combined with the high asymptotic performance of the large abstraction.

This article focussed on Markov abstractions, since this results in convergence guarantees of basic RL methods. In future work, we want to (empirically) evaluate the effect of non-Markov abstractions, especially for MDPs with context-specific structure. In addition, we would like to evaluate our abstraction selection strategy with model-based methods.

ACKNOWLEDGEMENTS

The authors would like to thank Csaba Szepesvari for helping out with some notational issues. The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024.

REFERENCES

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, **47**(2), 235–256.

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

Bellman, R. E. (1957b). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

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

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations* 1. *Artificial Intelligence*, **121**(1-2), 49–107.

Brafman, R. I. and Tennenholtz, M. (2002). R-max: a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, **3**, 213–231.

Chakraborty, D. and Stone, P. (2011). Structure learning in ergodic factored mdps without knowledge of the transition functions in-degree. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning (ICML)*.

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

Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, **5**(2), 142–150.

Dean, T., Givan, R., and Leach, S. (1997). Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 124–131. Citeseer.

Dietterich, T. G. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, **13**, 227–303.

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*.

Ferns, N., Panangaden, P., and Precup, D. (2004). Metrics for finite Markov decision processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 162–169. AUAI Press.

Givan, R., Dean, T., and Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, **147**(1-2), 163–223.

Jaakkola, T., Jordan, M. I., and Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, **6**, 1185–1201.

Jong, N. K. and Stone, P. (2005). State Abstraction Discovery from Irrelevant State Variables. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 752–757.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, **4**, 237–285.

Kearns, M. and Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *International Joint Conference on Artificial Intelligence*, pages 740–747.

Kearns, M. and Singh, S. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, **49**(2), 209–232.

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

Kroon, M. and Whiteson, S. (2009). Automatic Feature Selection for Model-Based Reinforcement Learning in Factored MDPs. In *ICMLA 2009: Proceedings of the Eighth International Conference on Machine Learning and Applications*, pages 324–330.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, **6**(1), 4–22.

Langford, J. and Zhang, T. (2007). The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in Neural Information Processing Systems*.

Langford, J., Strehl, A., and Wortman, J. (2008). Exploration scavenging. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 528–535. ACM.

Leffler, B., Littman, M., and Edmunds, T. (2007). Efficient reinforcement learning with relocat-

able action models. In *Proceedings of the Twenty Second National Conference on Artificial Intelligence*, page 572.

Li, L., Littman, M. L., and Walsh, T. J. (2008). Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th international conference on Machine learning*, pages 568–575. ACM New York, NY, USA.

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

Moore, A. and Atkeson, C. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, **13**, 103–130.

Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. thesis, University of California.

Pandey, S., Agarwal, D., Chakrabarti, D., and Josifovski, V. (2007). Bandits for taxonomies: A model-based approach. In *SIAM Data Mining Conference*. Citeseer.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, **24**, 1127–1137.

Ravindran, B. and Barto, A. G. (2003). SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *International Joint Conference on Artificial Intelligence*, pages 1011–1018. Citeseer.

Schwarz, G. (1978). Estimating the dimension of a model. *The annals of statistics*, **6**(2), 461–464.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvari, C. (2000). Convergence results for single-step on-policy reinforecement-learning algorithms. *Machine Learning*, **38**, 287–308.

Strehl, A. L., Diuk, C., and Littman, M. L. (2007). Efficient structure learning in factored-state MDPs. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, page 645. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, **3**(1), 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224.

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

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112**(1), 181–211.

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

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, **10**, 1633–1685.

van Seijen, H., Whiteson, S., van Hasselt, H., and Wiering, M. (2011). Exploiting Best-Match Equations for Efficient Reinforcement Learning. *The Journal of Machine Learning Research*, **12**, 2045–2094.

Wang, C. C., Kulkarni, S. R., and Poor, H. V. (2005). Bandit problems with side observations. *IEEE Transactions on Automatic Control*, **50**, 338–355.

Watkins, C. and Dayan, P. (1992). Q-Learning. *Machine Learning*, **8(3-4)**, 9–44.

APPENDIX A. Proof of Theorem 1

Theorem 1 Consider the MDP $\mathcal{M} = (\boldsymbol{\mathcal{X}}, \mathcal{A}, \tau, \rho)$ with $\boldsymbol{\mathcal{X}} = \mathcal{X}^1 \times \mathcal{X}^2 \times ... \times \mathcal{X}^N$. Abstraction $\mu(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}}$ with $\mathcal{S} \subseteq \{1, 2, ..., N\}$ is a Markov abstraction if each component $\mathcal{X}^k$ from $\boldsymbol{\mathcal{X}}$ with $k \notin \mathcal{S}$ is either irrelevant for $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ or an independent component.

Proof. For the proof we will make use of three rules that can be easily deduced from the Bayesian statistics rules. Consider the random variables $A \in \mathcal{A}$, $B \in \mathcal{B}$ and $C \in \mathcal{C}$. The following rules hold:

$$Pr(A \in U|B, C) \cdot Pr(B \in V|C) = Pr((A, B) \in U \times V|C), \quad \text{for all } U \in \mathcal{A}, V \in \mathcal{B} \tag{19}$$

$$Pr(A \in U|B) = Pr(A \in U|B,C,D), \quad \text{for all } U \in \mathcal{A} \quad \Rightarrow$$
$$Pr(A \in U|B) = Pr(A \in U|B,C), \quad \text{for all } U \in \mathcal{A} \tag{20}$$

$$Pr((A,B) \in U \times V|C) = Pr((A,B) \in U \times V|C,D), \quad \text{for all } U \times V \in \mathcal{A} \times \mathcal{B} \quad \Rightarrow$$
$$Pr(A \in U|C) = Pr(A \in U|C,D), \quad \text{for all } U \in \mathcal{A} \tag{21}$$

Let $\mathcal{S}^{ind} \subset \{1,2,...,N\}$ be the set of indices of the independent components that are not in $\mathcal{S}$, and $\mathcal{S}^{irr} \subset \{1,2,...,N\}$ be the set of indices of irrelevant components with respect to $\boldsymbol{\mathcal{X}}^{\mathcal{S}}$ (these component indices are by definition not in $\mathcal{S}$).

We start by proving that the abstraction based on all components except those with an index in $\mathcal{S}^{ind}$ is a Markov abstraction. That is, $\mu(\boldsymbol{x}) = \boldsymbol{x}^{\mathcal{S}^+}$ is Markov, with $\mathcal{S}^+ = \{1,\ldots,N\} \setminus \mathcal{S}^{ind}$. After that, we prove that the abstraction based on $\mathcal{S} = \mathcal{S}^+ \setminus \mathcal{S}^{irr}$ is also Markov.

Let $\mathcal{X}^k, k \in \mathcal{S}^{ind}$ be an independent component and let $\boldsymbol{\mathcal{X}}^{-k}$ be the product set spanned by all components, except for $\mathcal{X}^k$. We first prove that an abstraction based on $\boldsymbol{\mathcal{X}}^{-k}$ is Markov. The Markov property for $\boldsymbol{\mathcal{X}}$ is:[5]

$$Pr(\boldsymbol{X}_{t+1}, R_{t+1}|\boldsymbol{X}_t, A_t) = Pr(\boldsymbol{X}_{t+1}, R_{t+1}|\boldsymbol{X}_t, A_t, R_t, ..., R_1, \boldsymbol{X}_0, A_0).$$

Using (21), we can deduce the following equation:

$$Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t, A_t) = Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t, A_t, R_t, ..., R_1, \boldsymbol{X}_0, A_0). \tag{22}$$

We now multiply both sides of (22) with $Pr(X_t^k|\mathbf{X}_t^{-k}, A_t)$, $X_t^k \in \mathcal{X}^k$. The left part is then rewritten as:

$$Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t, A_t) \cdot Pr(X_t^k|\mathbf{X}_t^{-k}, A_t) = Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}, X_t^k|\boldsymbol{X}_t^{-k}, A_t)\,.$$

For the right part, we first rewrite $Pr(X_t^k|\mathbf{X}_t^{-k}, A_t)$, using the definition of an irrelevant component, as:

$$Pr(X_t^k|\mathbf{X}_t^{-k}, A_t) = Pr(X_t^k|\boldsymbol{X}_t^{-k}, A_t, R_t, ..., R_1, \boldsymbol{X}_0, A_0).$$

Multiplying this with the right part of (22) and rewriting it, using (19), gives:

$$Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}, X_t^k|\boldsymbol{X}_t^{-k}, A_t, R_t, ..., R_1, \boldsymbol{X}_0, A_0).$$

Combining the rewritten left and right part of (22) gives:

$$Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}, X_t^k|\boldsymbol{X}_t^{-k}, A_t) = Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}, X_t^k|\boldsymbol{X}_t^{-k}, A_t, R_t, \boldsymbol{X}_{t-1}, \ldots, R_1, \boldsymbol{X}_0, A_0).$$

Using (21), this can be rewritten as:

$$Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t^{-k}, A_t) = Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t^{-k}, A_t, R_t, \boldsymbol{X}_{t-1}, \ldots, R_1, \boldsymbol{X}_0, A_0)\,,$$

which in turn can be rewritten, using (20), as:

$$Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t^{-k}, A_t) = Pr(\boldsymbol{X}_{t+1}^{-k}, R_{t+1}|\boldsymbol{X}_t^{-k}, A_t, R_t, \boldsymbol{X}_{t-1}^{-k}, \ldots, R_1, \boldsymbol{X}_0^{-k}, A_0)\,,$$

By repeating this process for each independent component in $\mathcal{S}^{ind}$, the following equation is obtained:

$$Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}^+}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}^+}, A_t) = Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}^+}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}^+}, A_t, R_t, \boldsymbol{X}_{t-1}^{\mathcal{S}^+}, \ldots, R_1, \boldsymbol{X}_0^{\mathcal{S}^+}, A_0)\,.$$

Starting from this equation we now prove that the abstraction based on $\mathcal{S}$ is also Markov. First, we rewrite the equation, using (21) as:

$$Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}^+}, A_t) = Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}^+}, A_t, R_t, \boldsymbol{X}_{t-1}^{\mathcal{S}^+}, \ldots, R_1, \boldsymbol{X}_0^{\mathcal{S}^+}, A_0)\,.$$

Using the definition of irrelevant components, this can be rewritten as:

$$Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}}, A_t) = Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}^+}, A_t, R_t, \boldsymbol{X}_{t-1}^{\mathcal{S}^+}, \ldots, R_1, \boldsymbol{X}_0^{\mathcal{S}^+}, A_0)\,.$$

which, using (20), can be rewritten as:

$$Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}}, A_t) = Pr(\boldsymbol{X}_{t+1}^{\mathcal{S}}, R_{t+1}|\boldsymbol{X}_t^{\mathcal{S}}, A_t, R_t, \boldsymbol{X}_{t-1}^{\mathcal{S}}, \ldots, R_1, \boldsymbol{X}_0^{\mathcal{S}}, A_0)\,.$$

This last equation proves the theorem.

[5]For notational simplicity, we leave out that $(\boldsymbol{X}_{t+1}, R_{t+1}) \in U$ and that the equation holds for all $U \in \boldsymbol{\mathcal{X}} \times \mathbb{R}$. This same notational simplification is used for all the equations in this proof.

APPENDIX B. Off-Policy Monte Carlo Update

In this appendix, we deduce the equation for the off-policy Monte Carlo update of the switch actions for the abstraction-selection task of a contextual bandit problem. While in general, off-policy Monte Carlo updates are very inefficient, in this specific case a particular simple and efficient equation is obtained. In this appendix, we use $\pi(x)$ to refer to the action given by a deterministic policy, while we use $\pi(x, a)$ to refer to the action selection probability of a stochastic policy.

The experience sequence of the abstraction-selection task of a contextual bandit problem consists of only two actions: first a switch actions, a_0^{sw}, and then a regular action, a_1:

$$\boldsymbol{x}_0 \rightarrow a_0^{sw} \rightarrow \boldsymbol{x}_1 \rightarrow a_1 \rightarrow r_2\,. \tag{23}$$

A Monte Carlo update is an update with the complete return, i.e., the (discounted) cumulative reward. To understand the difference with regular (on-policy) Monte Carlo updates consider that we determine the Q-value of a state-action pair (x, a) by simply taking the average of all returns seen so far:

$$Q(\boldsymbol{x}, a) = \frac{\sum_{i=1}^{N} G_i(\boldsymbol{x}, a)}{N}\,, \tag{24}$$

where $G^i(\boldsymbol{x}, a)$ is the return followed by the i-th visit of state-action pair $(\boldsymbol{x}, a)$ and N is the total number of returns observed for $(\boldsymbol{x}, a)$. A similar off-policy version can then be made by taking the weighted average:

$$Q(\boldsymbol{x}, a) = \frac{\sum_{i=1}^{N} w^i(\boldsymbol{x}, a) \cdot G^i(\boldsymbol{x}, a)}{\sum_{i=1}^{N} w^i(\boldsymbol{x}, a)}\,, \tag{25}$$

where $w^i(\boldsymbol{x}, a)$ is the weight assigned to the i-th return for $(\boldsymbol{x}, a)$. The value of $w^i(\boldsymbol{x}, a)$ is computed as follows. Let $p(\boldsymbol{x}, a)$ be the probability of the experience sequence following (x, a) occurring under the estimation policy π and $p'(\boldsymbol{x}, a)$ be the probability of it occurring under the behaviour policy π'. Than the weight $w^i(\boldsymbol{x}, a)$ is equal to the relative probability of the observed experience sequence of occurring under π and π', i.e. by $p(\boldsymbol{x}, a)/p'(\boldsymbol{x}, a)$. These probabilities can be expressed in their policy probabilities as follows:

$$w(\boldsymbol{x}_t, a_t) = \frac{p(\boldsymbol{x}_t, a_t)}{p'(\boldsymbol{x}_t, a_t)} = \prod_{k=t+1}^{T-1} \frac{\pi(\boldsymbol{x}_k, a_k)}{\pi'(\boldsymbol{x}_k, a_k)}\,, \tag{26}$$

where T is the time step the terminal state is reached.

For a deterministic evaluation policy π the weight w is non-zero only when all actions taken under π' match the action that would have been taken under π. If this is the case, the above equation simplifies to:

$$w(\boldsymbol{x}_t, a_t) = \prod_{k=t+1}^{T-1} \frac{1}{\pi'(\boldsymbol{x}_k, a_k)}\,, \qquad \text{if } \pi(\boldsymbol{x}_k) = a_k \text{ for all } k \geqslant t+1\,, \tag{27}$$

where $\pi(x_k)$ refers to the action the agent would take at time step k (with probability 1) when following this deterministic policy.

Because for the sequence in (23), the state-action pair that requires the off-policy Monte Carlo update, $(\boldsymbol{x}_0, a_0^{sw})$, is followed by only a single action (a_1), the weight expression is reduced even further to:

$$w = \frac{1}{\pi'(\boldsymbol{x}_0, a_0^{sw})}\,, \qquad \text{if } \pi(\boldsymbol{x}_1) = a_1\,. \tag{28}$$

Given that we use an ϵ-greedy behaviour policy with fixed ϵ, and the condition that $\pi(\boldsymbol{x}_1) = a_1$, the weight w is a fixed value and can therefore be scaled to 1. Now, the off-policy Monte Carlo update of the switch action is reduced to:

$$Q(\boldsymbol{x}_0, a_0^{sw}) = (1-\alpha)\, Q(\boldsymbol{x}_0, a_0^{sw}) + \alpha \cdot r_2\,, \qquad \text{if } a_1 \text{ is greedy}\,. \tag{29}$$

□