Speeding-up Reinforcement Learning through Abstraction and Transfer Learning

Marcelo Li Koga
Universidade de São Paulo
LTI – PCS – EPUSP
Caixa Postal 05508–900
São Paulo, SP. Brazil
mlk@usp.br

Valdinei Freire da Silva
Universidade de São Paulo
EACH – USP Leste
Caixa Postal 03828–000
São Paulo, SP. Brazil
valdinei.freire@usp.br

Fabio Gagliardi Cozman
Universidade de São Paulo
PMR – EPUSP
Caixa Postal 05508–900
São Paulo, SP. Brazil
fgcozman@usp.br

Anna Helena Reali Costa
Universidade de São Paulo
LTI – PCS – EPUSP
Caixa Postal 05508–900
São Paulo, SP. Brazil
anna.reali@usp.br

ABSTRACT

We are interested in the following general question: is it possible to abstract knowledge that is generated while learning the solution of a problem, so that this abstraction can accelerate the learning process? Moreover, is it possible to transfer and reuse the acquired abstract knowledge to accelerate the learning process for future similar tasks? We propose a framework for conducting simultaneously two levels of reinforcement learning, where an abstract policy is learned while learning of a concrete policy for the problem, such that both policies are refined through exploration and interaction of the agent with the environment. We explore abstraction both to accelerate the learning process for an optimal concrete policy for the current problem, and to allow the application of the generated abstract policy in learning solutions for new problems. We report experiments in a robot navigation environment that show our framework to be effective in speeding up policy construction for practical problems and in generating abstractions that can be used to accelerate learning in new similar problems.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Dynamic programming*; I.2.6 [**Artificial Intelligence**]: Learning; I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms

Keywords

Machine learning for robotics, Single agent learning, Evolution and adaptation, Transfer learning

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

1. INTRODUCTION

In reinforcement learning (RL) an agent repeatedly observes the state of its environment and selects actions. Performing an action changes the state of the world, and the agent also obtains an immediate numeric payoff as a result. The agent must learn to select actions so as to maximize a long-term sum or average of the future payoffs it will receive. In the conventional RL framework, the agent does not initially know what effect its actions have on the state of the world, nor what immediate payoffs its actions will produce. In particular, the agent does not know which action is best to select at any given time. Rather, it must try out various actions in various states, and must gradually learn which action is best at each state so as to maximize the long run payoff. Alas, this whole process can be very time-consuming. Hopefully an agent can improve its learning abilities if solutions for similar past problems can be used in the current problem.

Indeed, much research in RL has improved learning speed by exploiting factored state representation. A factored state allows agents to share experiences among similar states [1, 14]. Some algorithms learn an abstract value function which accelerates learning, but may deteriorate policy quality [17]. Seeking to solve this problem, researchers have investigated how to define an abstraction that guarantee a good policy[16, 9, 20, 8]. Factored state can also be used to define temporal hierarchical abstractions, by solving smaller subproblems and obtaining a policy to the original problem [10, 3]. Because under abstraction the Bellman's principle of optimality may not be observed, search directly in the space of abstract policies has played a key role to obtain optimal abstract policies [19, 4, 5, 2]. Abstract policies allow transfer of knowledge among different problems with the same factored representation or transfer of knowledge among different representations of the same problem [11, 21, 13, 7, 6].

In this paper we are interested in the following general question: is it possible to abstract knowledge that is generated while learning the solution of a problem, so that this abstraction can accelerate the learning process? Moreover, is it possible to transfer and reuse the acquired abstract

knowledge to accelerate the learning process in future similar tasks? We propose a framework for simultaneously conducting two levels of reinforcement learning, where an abstract policy is built while learning a concrete policy for a practical problem, and both policies are refined through exploration and interaction of the agent with the environment. The abstract level must generalize knowledge learned in the concrete level; the abstract space must be smaller than the concrete space in ways that allow faster learning in the abstract level. The knowledge learned in the abstract level is then fed back into the concrete level, directing the search for an optimal solution; additionally, the abstract level builds an abstract policy that can be transferred to a number of similar problems. We explore these ideas in the remainder of the paper.

Our work is based on the fact that many domains can be described in terms of objects and the relations among them. Tasks in different domains can be represented in a similar relational form, leading to direct and elegant way to abstract knowledge through the use of logical variables. In doing so, concrete states are grouped into abstract states by the use of variables, allowing us to define abstract policies to represent knowledge about the solution of the concrete problem. Once an abstract policy is produced, it can be used in other learning problems as starting point. We explore the intuition that generalization from closely related, but solved, problems can produce policies that make good decisions in many states of a new unsolved problem.

Our experiments show that our proposed framework produces excellent results. An abstract policy can offer effective guidance; moreover, learning in the abstract level indeed converges faster than in the concrete level.

The paper is organized as follows. Section 2 reviews basic concepts of relational Markov decision processes (RMDP) and reinforcement learning. RMDPs form the basis for the concepts of abstract states and abstract policies. Section 3 discusses abstract states and how to learn and use abstract policies. Section 4 presents our framework for knowledge transfer from a source concrete problem to a target one, and for learning simultaneously in abstract and concrete levels. Section 5 reports experiments that validate our proposals, and Section 6 summarizes our conclusions.

2. RMDP AND RL

A formalism widely used for modeling sequential decision problems is the Markov Decision Process (MDP) [18]. A Relational MDP (RMDP) [15] is an extension of the MDP that uses a relational vocabulary to describe states and actions.

A *relational vocabulary* $\Sigma = C \cup P_S \cup P_A$ is a set of *constants* C that represent the objects of the environment; *predicates* P_S used to describe properties of and relations among objects; and *action predicates* P_A. We assume a *finite* relational vocabulary, with n_C constants, n_S predicates in P_S and n_A predicates in P_A.

If $\mathtt{t}_1, \ldots, \mathtt{t}_n$ are *terms*, i.e. each one is a constant (depicted with lower-case letter) or a variable (depicted with capital letter), and if $\mathtt{p}/n$ is a predicate symbol with arity $n \geq 0$, then $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ is an *atom*. If an atom does not contain any variable, it is called a *ground atom*. A set of atoms is a *conjunction* and it is a ground conjunction if it contains only ground atoms. The *Herbrand base* HB_Σ is the set of all possible ground atoms that can be formed with the n_S predicates and n_C constants in a relational vocabulary Σ. In our discussion each variable in a conjunction is implicitly assumed to be existentially quantified — this is indeed very important for the semantics of abstract states, as noted later.

An RMDP is defined as a tuple $\langle \Sigma, \mathcal{B}, \mathcal{S}, \mathcal{A}, T, R, \mathcal{G}, b^0 \rangle$, where

- the finite relational vocabulary Σ is used to specify states and actions as indicated below, and also to specify a knowledge base $\mathcal{B}$ of sentences in first-order logic that represents the characteristics and constraints of the problem;
- the set of (ground) states $\mathcal{S}$ is the set of all complete truth assignments for conjunctions of atoms in the Herbrand base $HB_{P_S \cup C}$ satisfying $\mathcal{B}$;
- the set of actions $\mathcal{A}$ is a subset of $HB_{P_A \cup C}$ satisfying $\mathcal{B}$ and $\mathcal{A}^s$ is the set of allowable actions in state $s \in \mathcal{S}$;
- the *transition function* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is such that $T(s, a, s') = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability of reaching state s' at time $t + 1$ when executing action a in state s at time t;
- $R : \mathcal{S} \to \mathbb{R}$ is a *reward function*, such that $r_t = R(s)$ is the reward received when the agent is in state s at time t;
- $\mathcal{G} \subset \mathcal{S}$ is a set of goal states, and when the goal is reached an episode ends and a new episode starts in some initial state chosen according to b^0. There are no transitions from any goal state, i.e. $T(s, a, s') = 0, \forall s \in \mathcal{G}, \forall a \in \mathcal{A}, \forall s' \neq s \in \mathcal{S}$ and $T(s, a, s) = 1$;
- $b^0 : \mathcal{S} \to [0, 1]$ is the initial state probability distribution, such that $b^0(s)$ is the probability of state s being the initial state in an episode.

When $\mathcal{S}$ and $\mathcal{A}$ in an RMDP are ground sets, we call this a *concrete* RMDP, and in this case an RMDP is an MDP where the states and the actions are represented through a relational language. We adopt a closed world assumption: if a ground atom does not appear in a ground sentence, the negated ground atom is assumed, as shown in Example 1:

> **Example 1** If $P_S = \{\mathtt{p}_1/1, \mathtt{p}_2/1, \mathtt{p}_3/1\}$ and $C = \{a, b\}$, then $\mathtt{p}_1(a) \wedge \mathtt{p}_2(b)$ denotes the state $s_1 = \mathtt{p}_1(a) \wedge \mathtt{p}_2(b) \wedge \neg\mathtt{p}_1(b) \wedge \neg\mathtt{p}_2(a) \wedge \neg\mathtt{p}_3(a) \wedge \neg\mathtt{p}_3(b)$.

At this point this closed world assumption is merely a notational simplification to compactly express maximal clauses; later the closed world assumption will be important during abstraction.

Given an (R)MDP, one is interested in finding a *policy*. An *optimal policy* π^* is a policy that maximizes some function R_t of the future rewards $r_t, r_{t+1}, r_{t+2}, \ldots$ A common definition, which we use, is to maximize the sum of *discounted rewards* over an *infinite horizon*: $R = \sum_{t=0}^{\infty} \gamma^t r$, where $0 \leq \gamma < 1$ is the *discount factor*. It is known that in a concrete MDP, the set of deterministic and memoryless policies $\pi : \mathcal{S} \to \mathcal{A}$, contains an optimal policy [12, 18].

In RL the agent does not know the transition probabilities T. Several RL algorithms can be used to find an optimal deterministic memoryless policy $\pi : \mathcal{S} \to \mathcal{A}$. In this paper we apply, when dealing with an RMDP, the conventional Sarsa(λ) algorithm to find π. The idea behind this algorithm

is as follows. The agent uses experience $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1}\rangle$ to learn estimates of optimal Q-value functions that map (s, a) pairs to the optimal return on taking action a in state s. At time step t the experience is used to update the eligibility trace function η_t and the Q-value estimate Q_t, as shown in Algorithm 1, where $0 \leq \lambda \leq 1$ is the decay rate of the eligibility trace, $0 \leq \gamma < 1$ is the discount factor, and $0 \leq \mu \leq 1$ is the learning rate. The eligibility trace function starts identically zero, and in episodic tasks it is reinitialized to zero after every episode. The greedy policy $\pi(s_t) = \arg\max_a Q_t(s_t, a)$ assigns the best action a to the state s_t.

Algorithm 1 Sarsa(λ) algorithm.

Initialize $Q(s,a)$, γ, λ, μ.
for each episode h **do**
  (Re)initialize $\eta_0(s,a)$ with 0.
  Observe s_0.
  Choose a_0 following a ϵ-greedy strategy.
  for each episode step $t \in \{0, 1, 2, \ldots\}$ **do**
    Apply a_t.
    Observe s_{t+1}, receive r_t.
    Choose a_{t+1} following a ϵ-greedy strategy.
    $\eta_{t+1}(s,a) = \begin{cases} 1 & \text{if } (s,a) = (s_t, a_t), \\ \gamma\ \lambda\ \eta_t(s,a) & \text{otherwise.} \end{cases}$
    $Q_{t+1}(s,a) = Q_t(s,a) + \mu\ \delta_t\ \eta_{t+1}(s,a)$, with
      $\delta_t = r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$.

The relational representation used to model the problem as an RMDP allows us to aggregate states and actions by keeping variables instead of substituting them by constants in the predicate terms. We explore this fact to achieve knowledge abstractions.

3. ABSTRACTION

This section defines abstract state and abstract actions, and discusses how to use them. We start by defining the semantics of formulas containing free variables through the ground states they specify. Suppose we have a conjunction F of (non-negated) predicates with free variables (only conjunctions of predicates are allowed in this paper). The free variables are considered implicitly existentially quantified. Also, recall the vocabulary is finite. The grounding of F is a DNF on ground predicates, containing a clause for each possible grounding of the predicates in F, plus negated groundings for all predicates not in F. For instance, in Example 1, the formula $p_1(X)$ is interpreted as

$$(\mathtt{p}_1(a) \wedge \neg\mathtt{p}_2(a) \wedge \neg\mathtt{p}_3(a) \wedge \neg\mathtt{p}_1(b) \wedge \neg\mathtt{p}_2(b) \wedge \neg\mathtt{p}_3(b)) \vee$$

$$(\neg\mathtt{p}_1(a) \wedge \neg\mathtt{p}_2(a) \wedge \neg\mathtt{p}_3(a) \wedge \mathtt{p}_1(b) \wedge \neg\mathtt{p}_2(b) \wedge \neg\mathtt{p}_3(b)).$$

Note the importance of the closed world assumption: predicates not in F appear negated in the grounding of F.

3.1 Abstract states and actions

We call *abstract state* σ (and similarly *abstract action* α) a conjunction of a set of non-negated predicates that specify a nonempty set of ground states s (ground actions a). An abstract state (action) is generated from a ground state (action) when constants in the ground state (action) are replaced by variables. Replacing just one or more constants with variables lets one vary the amount of abstraction in a particular scenario. In this paper we focus on a single kind of abstraction, where only non-negated ground atoms are replaced by non-ground non-negated atoms. That is, we only consider abstractions of the following form: given a ground state containing non-negated atom $p_1(a)$, replace this atom by $p_1(X)$ where X is a fresh logical variable (and do it for all non-negated ground atoms of the ground state) — the resulting expression represents an abstract state that abstracts the ground state.

> **Example 2** Consider a state where a robot is in a room a; that is, the state contains $\mathtt{inRoom}(a)$. Another state contains $\mathtt{inRoom}(b)$. An abstract state that abstracts both states contains $\mathtt{inRoom}(X)$, meaning that there exists a room such that the robot is in it.

We denote by P_σ the set of predicates that describes an abstract state σ, and by $\mathcal{S}_\sigma$ the set of ground states that are described using predicates of P_σ and that satisfies the constraints in the knowledge base $\mathcal{B}$. Note that an abstract action is a single atom in our model. If p_α is the action predicate of α, then we denote by $\mathcal{A}_\alpha$ the set of ground actions that are encoded by α; that is, the set of actions produced by grounding α. We also define $\mathcal{S}_{ab}$ and $\mathcal{A}_{ab}$ as the set of all abstract states and abstract actions in an RMDP, respectively. A *substitution* θ is a set $\{X_1/t_1, \ldots, X_n/t_n\}$, binding each variable X_i to a term t_i.

> **Example 3** Consider the abstract state $\sigma = \{p_1(X_1), p_2(X_2, X_3)\}$; then $P_\sigma = \{p_1/1, p_2/2\}$. Ground state $s_1 = \{p_1(t_1), p_2(t_1, t_2)\}$ is generated from abstract state σ by the substitution $\{X_1/t_1, X_2/t_1, X_3/t_2\}$. Likewise, ground state $s_2 = \{p_1(t_3), p_2(t_3, t_4)\}$ is generated from abstract state σ by the substitution $\{X_1/t_3, X_2/t_3, X_3/t_4\}$. In this case σ abstracts both s_1 and s_2; both s_1 and s_2 belong to $\mathcal{S}_\sigma$.

Hence each σ represents an aggregate of ground states, and the abstract state space partitions the original concrete state space $\mathcal{S}$ into a set of k subsets $\mathcal{S}_{\sigma_1}, \ldots, \mathcal{S}_{\sigma_k}$, where $k = \mid \mathcal{S}_{ab} \mid$. That is, $\mathcal{S} = \cup_{i=1}^{k} \mathcal{S}_{\sigma_i}$ and $\mathcal{S}_{\sigma_i} \cap \mathcal{S}_{\sigma_j} = \emptyset$, $i \neq j$.

3.2 Abstract policy

An abstract policy specifies an abstract action for each abstract state: $\pi_{ab} : \mathcal{S}_{ab} \rightarrow \mathcal{A}_{ab}$. The challenge is to apply an abstract policy in a concrete problem: we must provide a way to translate from the concrete to the abstract level, and vice-versa. We propose the following scheme.

Assume an abstract policy is given. For a ground state s we find the corresponding abstract state σ so that $s \in \mathcal{S}_\sigma$. The current abstract policy π_{ab} defines the best abstract action α to be applied in the abstract state σ. Note that an abstract action may be mapped into a set of concrete actions for the underlying concrete decision problem; thus an abstract state is mapped into a set of ground actions. To produce a particular ground action, we select randomly (with uniform probability) a concrete action a from the set of concrete actions $\mathcal{A}_\alpha \cap \mathcal{A}^s$, which combines $\mathcal{A}_\alpha$ with the set $\mathcal{A}^s$ of allowable actions in state $s \in \mathcal{S}$. This whole process yields the grounding of an abstract policy $\pi_{ab}(\sigma)$, which corresponds to using a strategy (greedy or ϵ-greedy, depending on the case) to define the abstract action α corresponding

to the abstract state σ, and then to defining the concrete action a that will be applied in the concrete state s. This is denoted by $a = \text{grounding}(\alpha, s)$. Obviously, other schemes may be used to produce a concrete action a, given s and $\pi_{ab}(\sigma)$.

3.3 Learning an abstract policy

Suppose one uses the Sarsa(λ) algorithm, given in Algorithm 1, to learn an abstract policy. Clearly only ground states are visited by the real system, and only ground actions can be actually applied. Learning must proceed by processing, at time t, the experience $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1}\rangle$. This concrete experience is obviously related to the tuple $\langle \sigma_t, \alpha_t, r_t, \sigma_{t+1}, \alpha_{t+1}\rangle$ that is used in the Sarsa(λ) algorithm, to update the eligibility trace function $\eta^t_{ab}(\sigma, \alpha)$ and the Q_{ab}-value estimate $Q^t_{ab}(\sigma, \alpha)$.

Now one may consider two distinct strategies. First, one can observe the states s_t and s_{t+1}, and the actions a_t and a_{t+1}, and translate them directly into σ_t, σ_{t+1}, α_t, α_{t+1}. This is what we call *active abstraction*, and the resulting reinforcement learning scheme is given by Algorithm 2. The second strategy assumes the existence of a "concrete level" reinforcement learning scheme that is learning a (concrete) policy, say using Sarsa (λ). Now the "abstract level" learner can observe the actions a_t and a_{t+1} taken by the concrete level learner, and translate then into α_t and α_{t+1} (together with σ_t and σ_{t+1}) so as to run Sarsa(λ). This is what we call *passive abstraction*, and the resulting reinforcement learning scheme is given by Algorithm 3.

Algorithm 2 Active Abstraction.

Initialize $Q_{ab}(\sigma, \alpha)$, γ, λ, μ.
for each episode h **do**
  (Re)initialize $\eta^0_{ab}(\sigma, \alpha)$ with 0.
  Observe s_0 and find the corresponding σ_0.
  Choose α_0 following a ϵ-greedy strategy.
  $a_0 = \text{grounding}(\alpha_0, s_0)$.
  for each episode step $t \in \{0, 1, 2, \ldots\}$ **do**
    Apply a_t.
    Observe s_{t+1} and find the corresponding σ_{t+1}.
    Receive r_t.
    Choose α_{t+1} following a ϵ-greedy strategy.
    $a_{t+1} = \text{grounding}(\alpha_{t+1}, s_{t+1})$.
    $\eta^{t+1}_{ab}(\sigma, \alpha) = \begin{cases} 1 & \text{if } (\sigma, \alpha) = (\sigma_t, \alpha_t). \\ \gamma\ \lambda\ \eta^t_{ab}(\sigma, \alpha) & \text{otherwise.} \end{cases}$
    $Q^{t+1}_{ab}(\sigma, \alpha) = Q^t_{ab}(\sigma, \alpha) + \mu\ \delta_t\ \eta^{t+1}_{ab}(\sigma, \alpha)$, with
      $\delta_t = r_t + \gamma Q^t_{ab}(\sigma_{t+1}, \alpha_{t+1}) - Q^t_{ab}(\sigma_t, \alpha_t)$.

Algorithm 3 Passive Abstraction.

Initialize $Q_{ab}(\sigma, \alpha)$, γ, λ, μ.
for each episode h **do**
  (Re)initialize $\eta^0_{ab}(\sigma, \alpha)$ with 0.
  for each episode step $t \in \{0, 1, 2, \ldots\}$ **do**
    Observe $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1}\rangle$.
    Find corresponding $\langle \sigma_t, \alpha_t, r_t, \sigma_{t+1}, \alpha_{t+1}\rangle$..
    $\eta^{t+1}_{ab}(\sigma, \alpha) = \begin{cases} 1 & \text{if } (\sigma, \alpha) = (\sigma_t, \alpha_t) \\ \gamma\ \lambda\ \eta^t_{ab}(\sigma, \alpha) & \text{otherwise.} \end{cases}$
    $Q^{t+1}_{ab}(\sigma, \alpha) = Q^t_{ab}(\sigma, \alpha) + \mu\ \delta_t\ \eta^{t+1}_{ab}(\sigma, \alpha)$, with
      $\delta_t = r_t + \gamma Q^t_{ab}(\sigma_{t+1}, \alpha_{t+1}) - Q^t_{ab}(\sigma_t, \alpha_t)$.

To summarize, active abstraction runs reinforcement learning and directly specifies the ground actions by translating the abstract actions, while passive abstraction simply observes the experiences of interactions experienced by a concrete level learner. Passive abstraction does not actually apply the abstract policy being learned in the task, but only receives the experiences from the interactions conducted at the concrete level of learning and uses these experiences to update the Q_{ab}-values.

Recall that our purpose here is to use the abstracted (smaller) state space to quickly guide the search for a policy for the concrete problem. We now argue that passive abstraction is better than active abstraction for such purpose, since the former is an abstraction of the learning process conducted at the concrete level, while the latter seeks to directly extract the structure of the task as built by a concrete learner, which is much more complex given the difficulties derived from aggregations of state information made in the abstract level.

An experimental comparison of the two strategies for learning abstract policies is shown in Figure 1, and in this figure we can see that the results support our arguments. These experiments were performed in the robotic navigation domain described in Section 5.1. In this experiment, the agent runs 1000 episodes (with a maximum of 500 steps in each) to complete the task to reach some goal location (5 different goal locations are used) from random initial states. We draw three curves: *Concrete*, *Abs-active* and *Abs-passive*. The curve *Concrete* is the result of learning at the concrete level using the standard Sarsa(λ) algorithm and an ϵ-greedy strategy. The curves *Abs-active* and *Abs-passive* show the result of applying Algorithms 2 and 3, respectively. Each point in the curve represents the actual value of the policy learned up to that episode. The value V^π of a policy π is the expected value of a policy, i.e.,

$$V^\pi = \sum_{s \in \mathcal{S}} b_0(s) \left\{ \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi, s_0 = s \right] \right\}. \qquad (1)$$

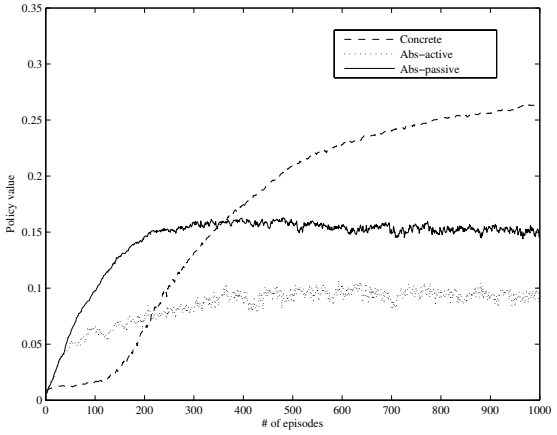

Figure 1: Comparison between learning in the concrete level with the plain Sarsa(λ) algorithm, and active and passive abstractions.

Note that active abstraction is significantly worse than passive abstraction, both failing to reach the optimal concrete policy. Also, note that both abstract schemes have faster convergence than the concrete level learner. Besides, note that both, passive and active abstractions, have a better performance in the initial episodes than the concrete level learner.

The challenge here is to conceive a scheme that can both converge quickly and converge to an optimal performance level. We now explain how learning in the abstract level can be combined with learning in the concrete level to produce such a result.

4. PROPOSED RL FRAMEWORK

As we have noted already, we are primarily interested in the abstraction of knowledge generated during the learning process, so that this abstraction can be used to speed up the process. Furthermore, our interest is also the transfer of learning results between tasks that have the same $\mathcal{S}_{ab}$ and $\mathcal{A}_{ab}$.

Recall that we have a given concrete decision problem for which we should learn a policy. As will be clear, just applying reinforcement learning to the concrete problem is *not* the best possible use of the available exploration data. The idea here is to use reinforcement learning to learn simultaneously a concrete level and an abstract level policy. Because the abstract policy is simpler (less states, less actions) than the concrete policy, learning is faster in the abstract level than in the concrete level, as we show in Figure 1. We thus have a framework where the abstract level gives an initial policy to the concrete level; then the abstract level is quickly refined, and continues to provide guidance to the concrete level; and then the concrete level is finely tuned to the target problem, finally leaving the guidance of the abstract level. The challenge is to create a framework where this process is guaranteed to reach optimality in the concrete target problem.

The agent's interaction with the environment clearly occurs in the concrete level, however the experience $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1}\rangle$ is used in both, abstract and concrete, so that values can be updated in both levels.

We use Sarsa(λ) (given in Algorithm 1) in both, concrete and abstract levels. In the abstract level, moreover, we use the passive learning strategy (given in Algorithm 3). However, the two learning processes should be integrated, so that the abstract policy π_{ab} that is being learned may suggest the exploration on the concrete level, guiding the agent to more promising spaces. We then balance the exploration of the environment and the exploitation of the concrete policy being learned. In order to achieve this balance, we use an adaptation of the PRQ-learning algorithm proposed elsewhere [6, 7].

In our framework, learning proceeds in multiple episodes. Each episode positions the agent in an initial state according to b_0 and terminates when either the goal state s_g is reached or a maximum number of steps is achieved. Here we consider that each task is defined by only one goal state, i.e. in each task $\mathcal{G} = \{s_g\}$.

At the beginning of each episode, the agent selects among possible policies to apply: the π_c policy in the concrete level, the π_{ab} policy in the abstract level, and the (if existing) π_{past} policies learned previously in similar tasks. This policy selection is guided by a probabilistic choice, according to values W^{π_c}, $W^{\pi_{ab}}$, and np values of $W^{\pi_{past}}$, and a softmax selection, where np is the number of past abstract polices considered.

The values W_k^π at any episode k are the average reinforcement received per episode after executing that policy, i.e.,

$$W_k^\pi = \frac{1}{\sum_{i=1}^{k-1} \mathbf{1}(\pi_i = \pi)} \sum_{i=1}^{k-1} \sum_{h=1}^{H} \mathbf{1}(\pi_i = \pi)\gamma^h r_{k,h},$$

where the function $\mathbf{1}(\pi_i = \pi)$ indicates whether the policy π was chosen in the episode i, and H is the maximum number of steps in each episode. Whatever the policy π_k selected, it will be applied for an entire episode k following an ϵ-greedy strategy, which at any time step t applies a random policy π_{random} in the concrete level with a probability ϵ, $0 \leq \epsilon \leq 1$ and a fully-greedy strategy otherwise.

If any abstract policy, whether π_{abs} or π_{past}, was followed in the episode, then only the values of $Q(s, a)$ are updated, since we use the passive approach for abstraction. Otherwise, the values of $Q(s, a)$ and $Q_{ab}(\sigma, \alpha)$ are updated after the episode. Also, after each episode we update the value W corresponding to the policy followed in the episode, i.e., we update W^{π_c}, $W^{\pi_{ab}}$, or one of the past polices $W^{\pi_{past}}$.

This procedure, called S2L-RL, a Simultaneous Two-layer Reinforcement Learning algorithm is described in Algorithm 4, where:

- ϵ, $0 \leq \epsilon \leq 1$, is the exploration probability for the ϵ-greedy strategy;
- λ, $0 \leq \lambda \leq 1$, is the decay rate of the eligibility trace;
- γ, $0 \leq \gamma < 1$, is the discount factor;
- μ, $0 \leq \mu \leq 1$, is the learning rate;
- τ is the temperature parameter for the Boltzmann strategy;
- Δ_τ is an incremental size in τ;
- W_0 is the estimated value of the policy in the concrete level, W^{π_c};
- W_1 is the estimated value of the policy in the abstract level, $W^{\pi_{ab}}$;
- W_i, $i = 2 \ldots (np + 1)$, is the estimated value of each past policy $W^{\pi_{past}}$.

That is: learn by using policy π_{past} or π_{ab} to guide exploration and gradually replace it with the greedy policy in the concrete level π_c, while using π_{random} less often, just to guarantee exploration and convergence to optimality. The choice is driven by the estimated value of each policy to the learning task.

When the agent starts learning, π_c is equal to π_{random}, since the Q function is usually initialized with the same value to all state-action pairs. If we assume that π_{past} was learned considering a similar task, the actions the agent will consider have the property that they used to be good to solve a similar problem before. Therefore the agent should obtain a better initial performance, i.e. a higher value of R_t in the first episodes.

Algorithm 4 S2L-RL: Simultaneous Two-layer RL.

Given ϵ, λ, γ, μ, τ, Δ_τ.
Given np past policies ($np = 0$ means no past policy).
Initialize $Q(s,a)$ and $Q_{ab}(\sigma,\alpha)$ with 0.
Initialize W_i with 0, $i = 0 \ldots np+1$.
for each episode $k = 1$ to K **do**
  (Re)initialize $\eta_k(s,a)$ and $\eta^k_{ab}(\sigma,\alpha)$ with 0.
  Select a policy π_k, given that each policy π_j, $j = 0, \ldots, np+1$, is assigned the following probability:

$$\frac{e^{\tau W_j}}{\sum_{p=0}^{np+1} e^{\tau W_p}}.$$

  Execute the learning episode k using Sarsa(λ) in the concrete and abstract levels.
  if $\pi_k = \pi_c$ **then**
    Update both $Q(s,a)$ and $Q_{ab}(\sigma,\alpha)$.
  else
    Update only the values $Q(s,a)$ in the concrete level.
  $\tau = \tau + \Delta_\tau$.

5. EXPERIMENTS

To show that our framework works in practice, we run a series of experiments in a robot navigation environment. We start by describing the navigation domain and then the experimental results are presented.

5.1 Robot navigation environment

We modeled the robot navigation environment by discretized regions that are described by a set of predicates and a set of objects. Regions are divided into rooms and corridors, and the predicates `inRoom(ri)` and `inCorridor(ci)` indicate whether the agent is in the first or in the second. With a range vision of two cells, the agent can also see: adjacent rooms, adjacent corridors, doors, fiducial markers (far, near, or next to) and empty space; and the predicates `seeDoor(di)`, `seeAdjRoom(ri)`, `seeAdjCorridor(ci)`, `seeMarkerFar(mi)`, `seeMakerNear(mi)`, `seeMarkerNext(mi)` and `seeEmptySpace` indicates respectively what the agent may see in a region. The agent observes adjacent rooms or corridors only if the agent is next to the door. Finally, the agent sense the direction and distance to the goal with three predicates: `nearGoal`, `appGoal(oi)` and `awayGoal(oi)`; the first indicates the agent is a Manhattan distance of at most 5 from the goal, the other ones indicates when the agent sees an object `oi`, if the object is closer to the goal (*predappGoal*(*oi*)) or farther from the goal (`awayGoal(oi)`) relatively to the agent. Objects can be: rooms (`ri`), corridors (`ci`), doors (`di`) and fiducial markers (`mi`).

All the experiments use a map with 184 concrete states and, depending on the goal position, tasks can be set with the number of abstract states ranging from 23 to 32. Figure 2 shows the map used in all the experiments, where a goal state s_g can be chosen among any state in a room. For example, the Figure 2 shows the task when the goal state is set $s_g = 4$; the abstract state $\sigma_1 =$ `inRoom(X)` $\wedge$ `seeDoor(Y)` $\wedge$ `appGoal(Y)` $\wedge$ `nearGoal` includes the set $\mathcal{S}_{\sigma_1} = \{7\}$, whereas the abstract state $\sigma_2 =$ `inRoom(X)` $\wedge$ `seeDoor(Y)` $\wedge$ `appGoal(Y)` includes the set $\mathcal{S}_{\sigma_2} = \{21, 27, 37, 77, 101, 105, 137, 143, 165, 166, 173, 177, 184\}$. Depending on the goal position, the set of predicates describes differently each enumerated state, giving rise to different abstractions. We ran experiments in the following goal positions: 4, 12, 81, 95 and 181.

Actions are described by predicates:
$P_A = \{$`goToDoorAppGoal(di)`, `goToRoomAppGoal(ri)`, `goToCorridorAppGoal(ci)`, `goToMarkerAppGoal(mi)`, `goToEmptyAppGoal`, `goToDoorAwayGoal(di)`, `goToRoomAwayGoal(ri)`, `goToCorridorAwayGoal(ci)`, `goToMarkerAwayGoal(mi)`, `goToEmptyAwayGoal`$\}$
and their names have clear meanings. If an agent chooses an abstract action that cannot be instantiated (for example, the agent chooses to move to door, but none is seen), the agent tries to move randomly using one of the avaliable actions; in the example in Figure 2, if the agent is in state 11, two actions can be instantiated: `goToDoorAppGoal(di)` and `goToDoorAwayGoal(di)`.

However, because the agent can only see empty space inside rooms and fiducial markers outside rooms, the agent takes into consideration such restrictions and explore just 8 abstract actions for each abstract state. On the other hand, because the set of concrete actions is too large, the agent only tries actions that can be executed in a given concrete state when learning in the concrete layer.

5.2 Results

To assess how efficient the proposed RL framework is, we evaluated the performance of three types of RL agents. The first is an agent using the classical Sarsa(λ) algorithm to solve a task, used as a reference. The second uses the S2L-RL framework, but without any previous knowledge, whereas the third also uses S2L-RL to solve a target task, but with the aid of some abstract policies of previously solved source tasks.

We consider a set of 5 tasks for the experiments, always with the goal inside a room. The initial state probability distribution b^0 is uniform for all states. The parameters used for all agents are: $\epsilon = 0.2$, $\mu = 0.05$, $\gamma = 0.95$, $\lambda = 0.9$ and $\Delta\tau = 0.05$. All these are constants and do not change during the learning process. The initial value of τ is set to $\tau = 0$ for the agent without reuse and to $\tau = 10$ for the agent with reuse of previous knowledge. This is because the former agent begins with two policies that are not ready and both are to be built, π_c and π_{ab}. Therefore an initial value $\tau = 0$ assigns equal probabilities to both of them. On the other hand, the agent with reuse already starts with a number of past policies, which are already built. The higher initial value of τ ensures that in the beginning of the learning process, these particular policies can be chosen more often.

For each task and agent, we run a total of 1000 episodes with a maximum number of steps per episode of 500. An episode starts in an initial state chosen following b^0 and it ends when the agent reaches the goal state or takes the maximum number of steps. Then a new episode is started and the process continues. When solving a task, the agent with reuse takes four abstract policies built previously with S2L-RL as its past policies. These policies are the solutions to all tasks, but the current one, in our set of 5 tasks. This process is repeated 30 times (totaling 150 executions per agent) and the average accumulated reward, averaging all tasks is shown in Figure 3.

We notice a significant improvement in performance, especially in the initial portion of the learning process, when comparing S2L-RL to Sarsa. This is mainly due to the fact that the abstract policy is built faster, as Section 3 shows.

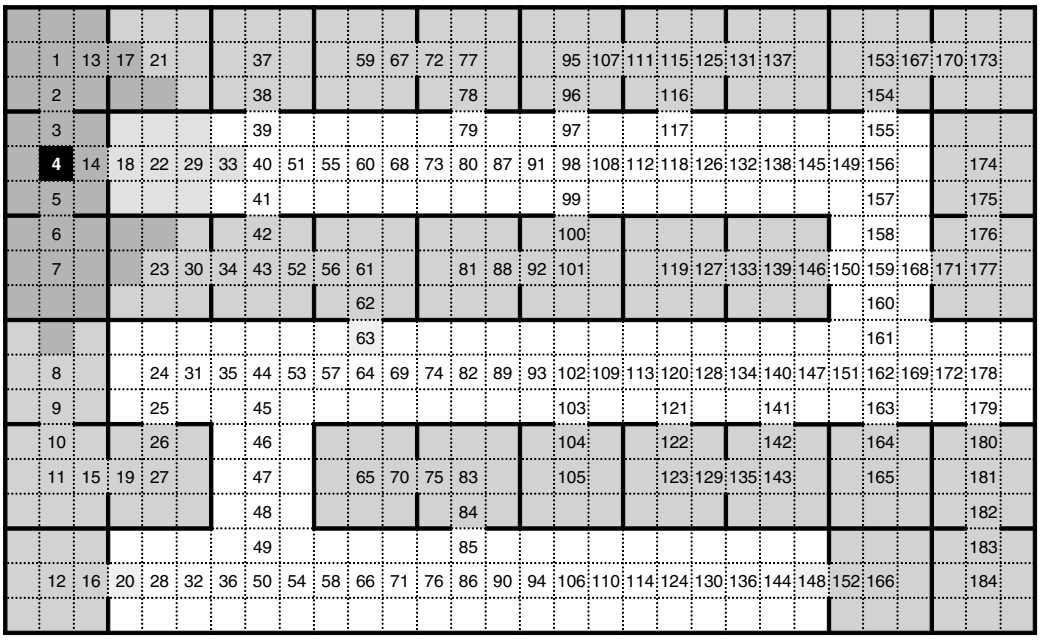

Figure 2: ***Robot navigation environment*** **- thick lines represent walls; darker cells, rooms; and white cells, corridors. As an example cell number 4 is the goal; the states considered near the goal are also marked in darker colors.**

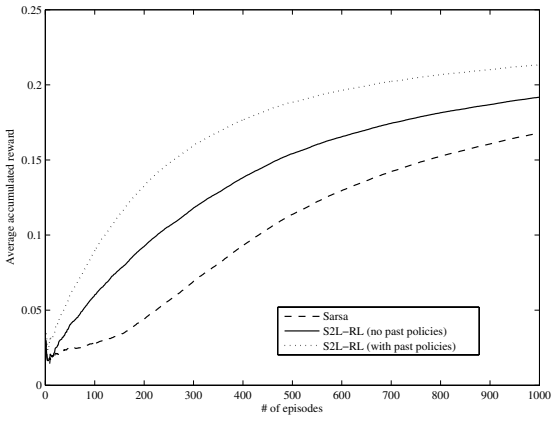

Figure 3: Comparison between Sarsa and S2L-RL. S2L-RL simultaneously learns an abstract and a concrete policy and can also reuse past abstract policies to accelerate learning.

The abstract policy reaches a maximum value that is lower than the maximum of the optimal policy, but as it is reached earlier, it can provide a good guidance to the agent. This thus yields higher reward gains while the ground policy is not ready. This means that not only does the agent learn faster but also it accumulates in average more reward during the whole learning process.

Furthermore, when the agent takes advantage of previous knowledge, the performance is even better. As it is expected that the chosen abstraction holds some properties across different tasks, the past policies can guide the agent in the very beginning of the learning process, when almost no knowledge is acquired yet.

However it is important not to overuse these policies. The measure W, the average reinforcement received per episode, gracefully controls the balance between all policies (the two being learned - concrete and abstract - and the past ones), cutting off the policy reuse when it is no longer necessary. Along the 1000 episodes, the average use of the concrete policy is around 86%, 10% for the past policies and the remaining 4% for the new abstract policy. The evolution of the usage of each policy over the time is shown in Figure 4. We can see that the contribution of past and abstract policies is concentrated in the first 200 episodes and thereafter the concrete policy assumes the actions.

6. CONCLUSIONS

In this paper we have proposed a framework, referred to as S2L-RL, for simultaneous reinforcement learning over abstract and concrete levels. We have described experiments that indicate that this framework does combine a speed up in learning convergence and a convergence to optimality. More precisely: while an abstract layer can be used to boost the learning speed, a concrete layer guarantees convergence. But beyond accelerating learning of an optimal policy, the experiments showed the synergy of both levels working together, as S2L-RL can improve over both levels. With regard to transfer learning, experiments also showed that S2L-RL can transfer effectively between similar tasks.

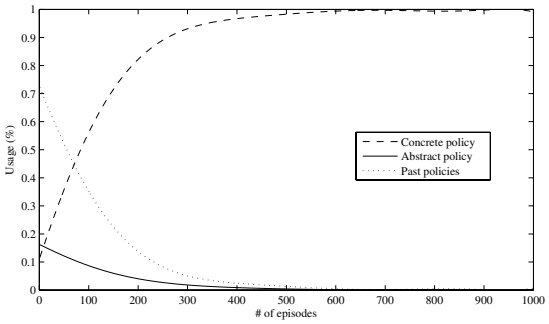

Figure 4: Evolution of policy usage, showing the average usage of each policy for 150 executions of S2L-RL.

The results here presented offer a new approach to boost learning speed in RL, giving us guidance on how to explore coarse abstraction (even if such a coarse abstraction does not guarantee Bellman's principle of optimality). Any advance into better abstractions, abstract value function estimation, or abstract policy search can be directly applied to our framework.

7. ACKNOWLEDGMENTS

This research was partially supported by FAPESP (2011/19280-8, 2012/02190-9, 2012/19627-0) and CNPq (311058/2011-6, 305395/2010-6).

8. REFERENCES

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

[2] D. D. Castro, A. Tamar, and S. Mannor. Policy gradients with variance related risk criteria. In *Proc. 29th Int. Conf. on Machine Learning (ICML-12)*, pages 935–942, New York, NY, USA, 2012. Omnipress.

[3] V. F. da Silva, F. d. A. Pereira, and A. H. R. Costa. Finding memoryless probabilistic relational policies for inter-task reuse. In *Proc. 14th Int. Conf. on Information Processing and Management of Uncertainty - IPMU'12*, volume 298, pages 107–116. Springer Berlin Heidelberg, 2012.

[4] T. Degris, M. White, and R. Sutton. Off-policy actor-critic. In *Proc. 29th Int. Conf. on Machine Learning (ICML-12)*, abs/1205.4839, New York, NY, USA, 2012. Omnipress.

[5] M. Deisenroth and C. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proc. 28th Int. Conf. on Machine Learning (ICML-11)*, pages 465–472, New York, NY, USA, 2011. ACM.

[6] F. Fernández, J. García, and M. Veloso. Probabilistic Policy Reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, July 2010.

[7] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. *Proc. 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems - AAMAS '06*, pages 720–727, 2006.

[8] R. Gaudel and M. Sebag. Feature selection as a one-player game. In *Proc. 27th Int. Conf. on Machine Learning (ICML-10)*, pages 359–366. Omnipress, 2010.

[9] A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. How. Online discovery of feature dependencies. In *Proc. 28th Int. Conf. on Machine Learning (ICML-11)*, pages 881–888, New York, NY, USA, 2011. ACM.

[10] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1015–1023. 2009.

[11] A. Lazaric and M. Ghavamzadeh. Bayesian multi-task reinforcement learning. In *Proc. 27th Int. Conf. on Machine Learning (ICML-10)*, pages 599–606. Omnipress, 2010.

[12] M. L. Littman. Memoryless policies: theoretical limitations and practical results. In *3rd Int. Conf. on Simulation of Adaptive Behavior: from animals to animats 3*, pages 238–245. MIT Press, 1994.

[13] T. Matos, Y. P. Bergamo, V. F. da Silva, and A. H. R. Costa. Simultaneous Abstract and Concrete Reinforcement Learning. In *Proc. 9th Symposium of Abstraction, Reformulation, and Approximation - SARA'11*, pages 82–89. AAAI Press, 2011.

[14] M. V. Otterlo. Reinforcement learning for relational MDPs. In *Machine Learning Conference of Belgium and the Netherlands*, pages 138–145, 2004.

[15] M. V. Otterlo. *The Logic of Adaptative Behaviour*. IOS Press, Amsterdam, 2009.

[16] C. Painter-Wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. In *Proc. 29th Int. Conf. on Machine Learning (ICML-12)*, pages 1391–1398, New York, NY, USA, 2012. Omnipress.

[17] J. Pazis and R. Parr. Generalized value functions for large action sets. In *Proc. 28th Int. Conf. on Machine Learning (ICML-11)*, pages 1185–1192, New York, NY, USA, 2011. ACM.

[18] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[19] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proc. 29th Int. Conf. on Machine Learning (ICML-12)*, abs/1206.4621, New York, NY, USA, 2012. Omnipress.

[20] Y. Sun, F. Gomez, M. Ring, and J. Schmidhuber. Incremental basis construction from temporal difference error. In *Proc. 28th Int. Conf. on Machine Learning (ICML-11)*, pages 481–488, New York, NY, USA, 2011. ACM.

[21] A. Tamar, D. D. Castro, and R. Meir. Integrating partial model knowledge in model free RL algorithms. In *Proc. 28th Int. Conf. on Machine Learning (ICML-11)*, pages 305–312, New York, NY, USA, 2011. ACM.