Comparative Analysis of Abstract Policies to Transfer Learning in Robotics Navigation[*]

Valdinei Freire, Anna Helena Reali Costa
Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav.3, 158, Cidade Universitária
05508-970 São Paulo, SP, Brazil
{valdinei.freire, anna.reali}@usp.br

Abstract

Reinforcement learning enables a robot to learn behavior through trial-and-error. However, knowledge is usually built from scratch and learning may take a long time. Many approaches have been proposed to transfer the knowledge learned in one task and reuse it in another new similar task to speed up learning in the target task. A very effective knowledge to be transferred is an abstract policy, which generalizes the learned policies in source tasks to extend the domain of tasks that can reuse them. There are inductive and deductive methods to generate abstract policies. However, there is a lack of deeper analysis to assess not only the effectiveness of each type of policy, but also the way in which each policy is used to accelerate the learning in a new task. In this paper we propose two simple inductive methods and we use a deductive method to generate stochastic abstract policies from source tasks. We also propose two strategies to use the abstract policy during learning in a new task: the hard and the soft strategy. We make a comparative analysis between the three types of policies and the two strategies of use in a robotic navigation domain. We show that these techniques are effective in improving the agent learning performance, especially during the early stages of the learning process, when the agent is completely unaware of the new task.

Introduction

Reinforcement Learning (RL) is a very powerful technique for autonomous and lifelong learning. RL allows an agent to learn behaviors by the acquisition of experience through interactions with a dynamic environment, on the basis of trial and error. However, this learning is usually built from scratch, taking a long time for the agent to learn to behave appropriately.

Seeking to reduce learning time in RL, many works have focused on transfer of knowledge between tasks. The determination of the knowledge to be transferred from one task to another has often been treated with techniques of transfer learning (Taylor and Stone 2009; Torrey and Shavlik 2009). Transfer learning allows knowledge to be achieved not only within tasks, but also across tasks. The goal is that the knowledge gained in source tasks helps in learning a new target task. The work reported in the literature can be organized in basically three different aspects: (i) the restriction they use about the similarity that must be respected between the source and target tasks, (ii) the use of one or more source tasks, and (iii) the type of knowledge to be transferred between tasks.

Regarding the first aspect, while some approaches are quite restrictive and require equal representation of states for all tasks, others are more flexible, but may require the designer to hand code a mapping between different task representations. The approaches from Fernandez and Veloso (Fernández and Veloso 2006) and Lazaric et al. (Lazaric, Restelli, and Bonarini 2008) requires that source and target tasks have the same state and action spaces, i.e., only the reward function can change from one task to another. On the other hand, the approaches from Torrey et al. (Torrey 2009), Taylor and Stone (Taylor, Stone, and Liu 2007), and Fachantidis et al. (Fachantidis et al. 2013) have need for mapping between tasks. As for the representation, in general the models are simple and with little semantics (e.g. enumerated states). This complicates knowledge transfer across tasks. An alternative is to use a factored state representation (as we do here), which allows agents to share experiences among similar states (Boutilier, Dearden, and Goldszmidt 2000; Konidaris, Scheidwasser, and Barto 2012), or even by using a more powerful representation such as relational representations (Koga, Freire, and Costa 2014; Croonenborghs, Driessens, and Bruynooghe 2007; Dzeroski, De Raedt, and Driessens 2001)

The second aspect deals primarily with the fact of whether using several previous solutions to source tasks as knowledge to transfer or using just one case, even if it is a generalization of a set of source tasks. Examples of case-based reasoning are the use of options (Sutton, Precup, and Singh 1999), which are a set of policies able to solve small subtasks that the agent might use when facing a similar situation, and analogical model formulation (Klenk, Aha, and Molineaux 2011), which uses a set of solutions to different problems to retrieve an analogous solution to the new problem. Celiberto Jr. et al. (Celiberto et al. 2011) also used cases as heuristics to achieve transfer learning combining case-based reasoning with heuristically accelerated RL. Fernández and

[*]We are gratefull to FAPESP (proc. n. 2012/19627-0) and CNPq (proc. n. 311058/2011-6).

Veloso (Fernández and Veloso 2006) proposed an RL algorithm with the reuse of a policy library, which contains concrete policies of previous tasks. Then, during the learning process, the probabilities of reusing a policy in the library are adjusted according to their usefulness. On the knowledge generalization side, the focus is on how to combine and to represent the solutions of a number of past experiences, extracting their similarities and hence generalizing previous knowledge. One example is the TILDE algorithm (Blockeel and De Raedt 1998), which induces a first-order logical decision tree (that represents a policy) from examples of solved tasks. Martín and Geffner (Martín and Geffner 2004) also explored this kind of approach by creating generalized policies, which are policies that, based on a number of solved problem instances, are suitable to solve any problem in a domain.

Finally, regarding the type of knowledge to be transferred between tasks, most methods provide the transfer of value functions (Liu and Stone 2006; Taylor, Stone, and Liu 2007), features extracted from the value functions (Banerjee and Stone 2007; Konidaris, Scheidwasser, and Barto 2012), heuristics (Bianchi et al. 2013), and policies (Ramon, Driessens, and Croonenborghs 2007; Koga et al. 2013) or partial policies (Mehta et al. 2008; Konidaris, Scheidwasser, and Barto 2012).

The idea here is to learn actuation policies in source tasks and, after abstracting the knowledge embedded in these policies, to transfer them to assist learning a new target task, thus speeding up the RL learning. We focus on methods which are policy-based, i.e., all the transferred knowledge is encoded by policies, since policies encompass better properties of generalization than value functions (Littman 1994). However, there are several ways to abstract the knowledge embedded in a policy. It is an open question how to define the best approach for abstraction and the best strategy to apply this abstracted knowledge in a new problem. Based on previous work, we use a single generalization of solutions of source tasks as knowledge to be transferred between tasks, since this generalization is more effective than the use of policies libraries (Koga, Freire, and Costa 2014). Thus, the aim of this paper is to evaluate the effectiveness of policies generated by different abstraction approaches for acceleration of new learning in a simulated robotic navigation domain. We also propose two strategies to use the abstract policy during learning in a new task: the hard and the soft strategy. We use a simulated robotic navigation domain for the experiments, and we evaluate how a single abstract policy obtained from a set of source tasks using different abstraction methods can effectively guide and accelerate learning onto a new target task.

Reinforcement Learning and Abstraction

Our robot uses RL to refine behavior through interactions with an environment (Sutton and Barto 1998). An elegant and very used formalism to model problems is Markov Decision Process, MDP. In this section we first formalize MDP and present a very popular RL algorithm for solving them, the Q-learning algorithm, which is the basis of our experimental evaluations. The reader familiar with these concepts can skip this section. Then, we describe the concepts involved in abstract policies.

Markov Decision Process

Our problem is modeled as a Markov Decision Process (MDP). A finite discrete-time fully observable MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ (Puterman 1994), where:

- $\mathcal{S}$ is a finite set of fully observable states of the process;
- $\mathcal{A}$ is a finite set of all the possible actions to be executed at each state;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function that specifies the probability $\mathcal{T}(s, a, s')$ that the system moves to state s' when action a is executed in state s;
- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is a reward function that produces a finite numerical value $r = \mathcal{R}(s)$ when state s is reached.

An MDP agent is continuously in a cycle of perception and action: at each time t the agent observes state $s \in \mathcal{S}$ and decides which action $a \in \mathcal{A}$ to perform, the execution of this action causes the transition to a new state s' according to the transition probability function $\mathcal{T}$ and the agent receives a reward r. This cycle is repeated until a stopping criterion is met; for example: when a goal state is reached.

Q-Learning

To solve an MDP is to find an *optimal policy* π^* that maximizes a function $\mathcal{R}_t$ of future rewards. Here we use $\mathcal{R}_t = \sum_{t=0}^{\infty} \gamma^t r_t$, where $0 \leq \gamma < 1$ is the *discount factor*. In RL tasks the agent does not know the transition probabilities $\mathcal{T}$ and the reward function $\mathcal{R}$, and a series of RL algorithms can be used to find a policy.

We use the very popular Q-learning algorithm (Sutton and Barto 1998) to find π, in which at time step t the experience $\langle s_t, a_t, s_{t+1}, r_t \rangle$ is used to update the Q-value estimate Q_t of applying a_t in s_t, and receiving reward r_t:

$$\begin{aligned} Q_{t+1}(s_t, a_t) \leftarrow & (1 - \alpha) Q_t(s_t, a_t) \\ & + \alpha \left(r_t + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1}, a) \right), \end{aligned} \tag{1}$$

where γ is the discount factor, and $0 \leq \alpha \leq 1$ is the learning rate. The greedy deterministic policy $\pi(s_t) = \arg\max_{a \in \mathcal{A}} Q_t(s_t, a)$ assigns the best action a to state s_t.

State Abstraction

So far nothing has been said about state representation in MDPs. We propose a robotic system that consists of two levels: a *concrete level*, in which interactions actually occur of the robot with the environment, and an *abstract level*, in which generalizations of past solutions are used by the robot to speed up the learning of a new problem. At the concrete level the states are represented by enumeration, while at the abstract level we consider special sensors, which encode semantic information. Here, we consider a concrete level under which the Markov property is guaranteed, i.e., the state transition depends only on the current state and current action, $P(s_{t+1}|a_t, s_t, s_{t-1}, \ldots, s_1, s_0) = P(s_{t+1}|a_t, s_t)$. Although the Markov property is difficult to guarantee in continuous environments, if appropriate discretization is considered, Q-learning can converge to near optimal policies. In a robotic navigation environment, such concrete level can be obtained through a global metric sensor such as GPS system or SLAM. Different approaches, based on semantic maps and qualitative navigation (Toro et al. 2014) can also be used. In our experiment, we consider a simulated discrete environment, which guarantees Markov property.

The abstract level considers a set of sensors that provide semantic information directly related to the concrete level, but adds the ability of generalization if the same meaning can be shared between tasks, environments or states. Then, in the abstract level, states are represent by a vector of k measurable features in the environment given by such sensors. Let μ be the function $\mu : \mathcal{S} \rightarrow \prod_{i=1}^{k} \mathcal{X}_i$, where $\mathcal{X}_i$ is the set of values of the domain of feature i. Then, a state becomes an instantiation of features and can be written as a vector $s = (x_1, \cdots, x_k)$ such that $\forall x_i \in \mathcal{X}_i$. Without loss of generality, here we consider only binary features, i.e., $\mathcal{X}_i = \{0, 1\}$. Our state in the abstract level is then a conjunction of binary features, $s = x_1 \wedge x_2 \cdots \wedge x_k$.

By using feature conjunctions to represent states, two major aspects appear. First, each feature usually has an associated semantics, which allows the generalization of acquired knowledge from one task to be used in another task with the same semantics. Second, if features describe the states partially, i.e., there exist $s, s' \in \mathcal{S}$ such that $\mu(s) = \mu(s')$, then the set of features itself already represents an abstraction of the states that share the same description.

A state space abstraction consists in partitioning the state space by a function $\sigma : \mathcal{S} \rightarrow \mathcal{S}_{Abs}$, where $\mathcal{S}_{Abs}$ is the set of abstract states. If a feature conjunction describes a state partially, a natural abstraction consists in: (*i*) if $\mu(s) = \mu(s')$, then $\sigma(s) = \sigma(s')$; and (*ii*) if $\mu(s) \neq \mu(s')$, then $\sigma(s) \neq \sigma(s')$. We define $\rho \in \mathcal{S}_{Abs}$ as an abstract state, i.e., $\sigma(s) = \rho$, with $s \in \mathcal{S}$ and $\rho \in \mathcal{S}_{Abs}$. $\mathcal{S}^{\rho} \subseteq \mathcal{S}$ is the set of concrete states s that are mapped into the same abstract state ρ, i.e., if $\sigma(s) = \sigma(s') = \rho$, then $s, s' \in \mathcal{S}^{\rho}$.

Stochastic Abstract Policies

Let $\mathcal{S}^1$ and $\mathcal{S}^2$ be the set of concrete states for two problems, MDP_1 and MDP_2. In our approach, we transfer knowledge across problems that not only have the same state space, i.e., $\mathcal{S}^1 = \mathcal{S}^2$, but also across those that are different, $\mathcal{S}^1 \neq \mathcal{S}^2$, but maintain the property of being described by the same subset of k features associated with the same semantics. In both problems, concrete states are described in the abstract level by a set of k features with the same associated semantics, therefore defining a space of abstract states equivalent for both problems, i.e., $\mathcal{S}^1_{Abs} = \mathcal{S}^2_{Abs} = \mathcal{S}_{Abs}$. Here every task uses of the same action space, i.e., $\mathcal{A}^1 = \mathcal{A}^2 = \mathcal{A}$. Then, transfer is done by defining a stochastic abstract policy in the source tasks and somehow using such a policy in the target task. We explore two paradigms: induced and deduced policies to build abstract policies.

Induced Policies

Induced policies are obtained by inducing an abstract policy from a set of optimal concrete deterministic policies $\Pi = \{\pi^1, \pi^2, \ldots, \pi^N\}$, $\pi^i : \mathcal{S}^i \rightarrow \mathcal{A}$, associated to a set of source tasks; in the limit Π may contain only one policy.

We propose the induction of two types of abstract policies: a probabilistic induced policy and a non-deterministic induced policy. The former defines a probabilistic abstract policy $\pi_{Abs_{Prob}} : \mathcal{S}_{Abs} \times \mathcal{A} \rightarrow [0, 1]$, whereas the latter defines a policy that maps abstracts states into a subset of actions, i.e., $\pi_{Abs_{ND}} : \mathcal{S}_{Abs} \rightarrow 2^{\mathcal{A}}$.

For every $\rho \in \mathcal{S}_{Abs}, a \in \mathcal{A}$, the probabilistic induced policy is defined by:

$$\pi_{Abs_{Prob}}(\rho, a) = \frac{\sum_{i=1}^{N} |\{s : s \in \mathcal{S}^i \wedge \sigma(s) = \rho \wedge \pi^i(s) = a\}|}{\sum_{i=1}^{N} |\{s : s \in \mathcal{S}^i \wedge \sigma(s) = \rho\}|},$$

where $\mathcal{S}^i$ is the set of states in the i-th task and π^i is the optimal policy for task i. Therefore, the probabilistic policy takes into account every state in every task which is represented by the abstract state $\rho \in \mathcal{S}_{Abs}$, and averages equally over the policy of such states.

The non-deterministic induced policy does not consider the number of states that choose a certain action as optimal; given an action a and abstract states ρ, it only considers if there exists a concrete states s which is represented abstractly by ρ and has a as optimal policy, i.e., $\sigma(s) = \rho$ and $\pi^*(s) = a$. Therefore, for every $\rho \in \mathcal{S}_{Abs}, a \in \mathcal{A}$, the non-deterministic induced policy is defined by:

$$\pi_{Abs_{ND}}(\rho) = \{a : \exists \pi^i \in \Pi, \exists s \in \mathcal{S}^i | \sigma(s) = \rho \wedge \pi^i(s) = a\},$$

i.e., $a \in \pi_{Abs_{ND}}(\rho)$ iff a is optimal in any task for a state s so that $\sigma(s) = \rho$, i.e., $s \in \mathcal{S}^{\rho}$.

Deduced Policy

Although the induced policies are built from optimal concrete policies, the quality of such induced policies can not be assured, since they represent combinations of different solutions for different states that are represented by the same abstract state in the abstract level.

An alternative to the induced policies is to search directly in the space of abstract policies to find the best one. It is easy to define a gradient such that a local optimal policy can be found; for example, the AbsProb-PI algorithm (da Silva, Pereira, and Costa 2012) and its extension AbsProb-PI-multiple (Koga, Freire, and Costa 2014) can find such a local optimal policy, named here $\pi_{Abs_{Grad}}$.

Remember that we are interested in maximizing the expected sum of rewards $V^{\pi}(s) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \pi, s_0 = s]$. Given a probabilistic policy π, V^{π} can be defined by:

$$V^{\pi}(s) = r(s) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} T(s, a, s') \pi(s, a) V^{\pi}(s').$$

While the optimization problem is well defined when considering concrete policies by maximizing uniformly $V^{\pi}(s)$ for all $s \in \mathcal{S}$; when abstract policies are considered the optimization problem is well defined only if an initial state distribution $\beta : \mathcal{S} \rightarrow [0, 1]$ is considered by maximizing $\mathcal{V}^{\pi} = \sum_{s' \in \mathcal{S}} \beta(s) V^{\pi}(s)$ (da Silva, Pereira, and Costa 2012).

The algorithms AbsProb-PI and AbsProb-PI-multiple define the policy gradient of $\mathcal{V}^{\pi}$ and move in the direction of that gradient to find a local-optimal policy. To find an abstract policy that generalizes a set of tasks $\{1, 2, \ldots, N\}$, the MDP of each task is grouped in a bigger MDP, where: (*i*) $\mathcal{S} = \bigcup_{i=1}^{N} \mathcal{S}^i$, (*ii*) if $s \in \mathcal{S}^i$, then $\beta(s) = \frac{\beta^i(s)}{N}$; and (*iii*) the set of actions, reward function and transition function are the same of the original tasks. Despite AbsProb-PI iand AbsProb-PI-multiple are planning algorithms, learning RL algorithms that makes use of policy gradient is also found in the literature (Sehnke et al. 2008; Bhatnagar et al. 2009).

Transferring policies among tasks

If the source tasks and target tasks make use of the same set of features to describe abstract states, then the abstract policies obtained by the solution of source tasks can be used directly in the target tasks. It is important to notice that such abstract policy may not be optimal in the target task, and, more than that, might even worsen the behavior of the robot in the target task. Thus, one should only use it as an advise at the beginning of the robot learning in the new task, since at the beginning of learning nothing is known about the new task, and it is better to follow an advice than acting randomly. Although a transfer policy may not always helps, we consider it does so on average.

Clearly only concrete states are visited by the real system. Learning must proceed by processing, at time t, the experience $\langle s_t, a_t, r_t, s_{t+1}\rangle$ that is used to update the Q-value estimate in the concrete level. The concrete state s_t is related to the abstract state $\sigma(s_t) = \rho$. Here we propose to use the abstract policy obtained from similar source tasks in the $\epsilon-$greedy choice that the Q-Learning algorithm makes about the action to be performed.

We propose the following scheme to apply an abstract policy in a concrete problem. Given a *probabilistic* abstract policy $\pi_{Abs_{Prob}} : \mathcal{S}_{Abs} \times \mathcal{A} \rightarrow [0, 1]$, an observed concrete state $s_t \in \mathcal{S}$ and its corresponding abstract state $\sigma(s_t) = \rho \in \mathcal{S}_{Abs}$, the robot selects an action $a \in \mathcal{A}$ according to these probabilities, and then applies it in the concrete state s_t. If, on the other hand, the robot is given a non-deterministic abstract policy $\pi_{Abs_{ND}} : \mathcal{S}_{Abs} \rightarrow 2^{\mathcal{A}}$, and it observes a concrete state s_t and determines its corresponding abstract state $\sigma(s_t) = \rho \in \mathcal{S}_{Abs}$, then the robot selects with uniform distribution in $\pi_{Abs_{ND}}(\rho) \subseteq \mathcal{A}$ the action $a \in \pi_{Abs_{ND}}(\rho)$ to be performed.

We make use of Q-Learning algorithm to learn in the concrete level, whereas making use of the abstract policy to guide exploration. Given a time horizon τ to make use of the abstract policy, we consider two approaches: hard and soft. Algorithm 1 shows the hard approach, where before time τ only the abstract policy is used, while learning within Q-Learning. After time τ, an ϵ-greedy strategy is used: the best action $a_{Best} = \arg\max_a Q_t(s_t, a)$ is selected for a proportion $1 - \epsilon$ of the trials, and an action is selected at random (with uniform probability) for a proportion ϵ. Algorithm 2 shows the soft approach, in which before time τ the abstract policy is used in combination with an ϵ-greedy strategy; such a combination is given by λ that starts with rate 1 for the abstract policy, and decays linearly until rate 0. After this time τ, only the ϵ-greedy strategy is used in the Q-learning algorithm.

Algorithm 1 Hard Strategy to Transfer Policy.

```
Require: Given an abstract policy $\pi_{Abs}$, a maximum number of time steps $\tau$, an exploration rate $\epsilon$, a learning rate $\alpha$
 1: $Q(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
 2: for each time step $t \leq \tau$ do
 3:     observe state $s_t$
 4:     execute action $a_t$ by following policy $\pi_{Abs}$
 5:     observe reward $r_t$ and state $s_{t+1}$
 6:     update $Q(s_t, a_t)$ according to Eq.1
 7: end for
 8: for each time step $t > \tau$ do
 9:     observe state $s_t$
10:     choose action $a_t$ by following $\epsilon$-greedy strategy
11:     execute action $a_t$
12:     observe reward $r_t$ and state $s_{t+1}$
13:     update $Q(s_t, a_t)$ according to Eq.1
14: end for  return $Q$-function
```

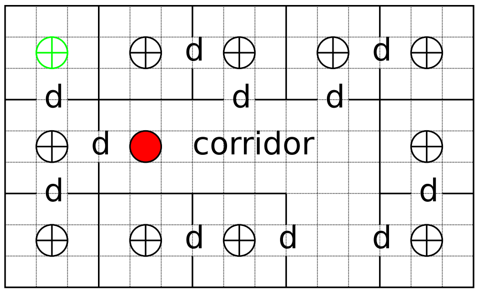

Figure 1: The simulated source environment.

Experiments

To evaluate each type of policy for the knowledge transfer process we conducted experiments in two simulated robotic-navigation environments (Figures 1 and 2). The agent learns in the source tasks (environment in Figure 1) and transfer the knowledge to the target tasks (environment in Figure 2).

For each environment we define a number of goals, which represent different tasks, i.e., different MDPs to the agent. To analyze and compare the performances of different types of policies and transfer learning strategies, we evaluate the three versions of abstract policies: $\pi_{Abs_{Prob}}$, $\pi_{Abs_{ND}}$ and $\pi_{Abs_{Grad}}$.

Environments

The environment is composed of three types of objects: rooms, doors (d) and a corridor. While the agent accesses some rooms immediately from the hall, others can only be reached through a predecessor room.

The first environment (Figure 1) is composed of: 135 concrete states; 11 rooms; and 11 goals positions (symbol $\oplus$). The second environment (Figure 2) which represents a larger MDP problem (four times larger than the learning environment) and will be used as the target environment for the knowledge transfer is composed of: 540 discrete concrete

Algorithm 2 Soft Strategy to Transfer Policy.

```
Require: Given an abstract policy $\pi_{Abs}$, a maximum number of time steps $\tau$, an exploration rate $\epsilon$, a learning rate $\alpha$
 1: $Q(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
 2: for each time step $t \leq \tau$ do
 3:     observe state $s_t$
 4:     define rate $\lambda = \frac{\tau - t}{\tau}$
 5:     with probability $\lambda$, choose action $a_t$ by following policy $\pi_{Abs}$
 6:     with probability $1-\lambda$, choose action $a_t$ by following $\epsilon$-greedy strategy
 7:     execute action $a_t$
 8:     observe reward $r_t$ and state $s_{t+1}$
 9:     update $Q(s_t, a_t)$ according to Eq.1
10: end for
11: for each time step $t > \tau$ do
12:     observe state $s_t$
13:     choose action $a_t$ by following $\epsilon$-greedy strategy
14:     execute action $a_t$
15:     observe reward $r_t$ and state $s_{t+1}$
16:     update $Q(s_t, a_t)$ according to Eq.1
17: end for  return $Q$-function
```

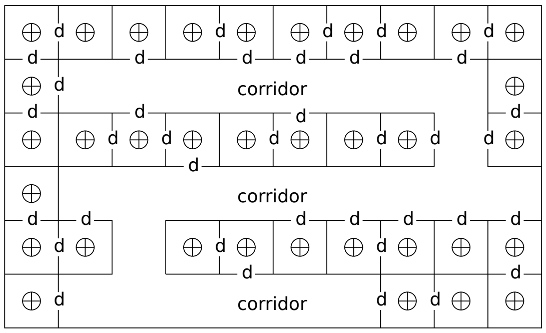

Figure 2: The simulated target environment.

states; 35 rooms; and 35 goal positions. We represent different tasks by simply choosing different goal positions; then, there exists 11 source tasks (Figure 1), and 35 target tasks (Figure 2).

For the abstract state description we consider features related to local observations and the goal position. The local observation feature set is represented by $\mathcal{F}_L = \{\text{see\_empty\_space}, \text{see\_door\_far}, \text{see\_room}, \text{see\_corridor}\}$ and the features related to the goal position are represented by $\mathcal{F}_G = \{\text{near\_goal}, \text{almost\_near\_goal}\}$. The features in $\mathcal{F}_G$ describe the distance of a state to the goal position. A state is near_goal if it is at least a Manhattan distance of 5 to the goal, and a state is almost_near_goal if it is at least a Manhattan distance of 8 and is not near_goal.

From the set $\mathcal{F}_L$ we define the feature sets $\mathcal{F}_L^h$ and $\mathcal{F}_L^{op}$, which represent features of the current state that are or not toward the goal. The set $\mathcal{F}_L^h$ is defined by considering the set $\mathcal{F}_L$, but adding the suffix 'head_to_goal' to the original feature notation, and in the same way, the set $\mathcal{F}_L^{op}$ is defined by considering the suffix 'opposite_to_goal'. The final set that will be used to describe a state is $\mathcal{F} = (\mathcal{F}_L^h \cup \mathcal{F}_L^{op} \cup \mathcal{F}_G \cup \{\text{in\_room}\})$ totalizing 11 features, where in_room is a local observation feature without goal relation and indicates that the agent is in one of the rooms (or in the corridor). As an example of the state description defined previously, consider the robot to be the red circle in the environment shown in Figure 1, and the goal in green to be the current goal state. Then, the agent observes the following abstract state: $\text{see\_door\_far\_head\_to\_goal} \wedge \text{see\_empty\_space\_opposite\_to\_goal} \wedge \text{almost\_near\_goal}$.

Table 1: Performance of each type of policy.

	source environment	target environment
π_{rand}	-88.97	-97.07
$\pi_{Abs_{ND}}$	-44.61	-80.41
$\pi_{Abs_{Prob}}$	-36.91	-70.03
$\pi_{Abs_{Grad}}$	-30.65	-70.34
π^*	-14.86	-29.13

Setup

Initially we solve the 11 tasks in the source environment and generate the three types of policies for them: probabilistic induced policy ($\pi_{Abs_{Prob}}$), non-deterministic induced policy ($\pi_{Abs_{ND}}$) and probabilistic deduced policy ($\pi_{Abs_{Grad}}$). To find the policy $\pi_{Abs_{Grad}}$ and to evaluate the performance of the policies, we consider a uniform probability distribution for the initial state. We set $r(s) = 0$ if state s is the goal in the current task, and $r(s) = -1$ otherwise, and we consider $\gamma = 0.99$ in every policy.

In the transfer learning experiment, we consider the target environment and the 34 tasks together. In every episode, the agent starts with a random task and in an initial state, using a uniform distribution; the episode ends when the agent reaches the goal position or if 5,000 steps elapse, which one occurs first. Then, in fact, the agent faces a bigger task involving a total of $540 \times 35 = 18,900$ states.

Learning is conducted during 30×10^6 steps, with fixed $\epsilon = 0.1$, $\gamma = 0.9$ and $\alpha = 0.1$. The parameter τ was tuned for each pair of abstract policy and transfer strategy: (i) π_{rand}; $\tau = 9 \times 10^6$ (hard) and $\tau = 19 \times 10^6$ (soft); (ii) $\pi_{Abs_{ND}}$; $\tau = 3.5 \times 10^6$ (hard) and $\tau = 11 \times 10^6$ (soft); (iii) $\pi_{Abs_{Prob}}$; $\tau = 3.5 \times 10^6$ (hard) and $\tau = 6 \times 10^6$ (soft); (iv) $\pi_{Abs_{Grad}}$; $\tau = 3.5 \times 10^6$ (hard) and $\tau = 6 \times 10^6$ (soft); Each experiment was reproduced 50 times.

Results

Table 1 shows the performance of each policy in the source tasks, where they were induced or deduced, and in the target tasks, where they were executed (without learning). Table 1 also shows the performance of the random policy (π_{rand}) and the optimal policy (π^*) as a reference.

As it was expected, policy $\pi_{Abs_{Grad}}$ presented the best performance among all abstract policies in the source environment; however, $\pi_{Abs_{Grad}}$ and $\pi_{Abs_{Prob}}$ have similar per-

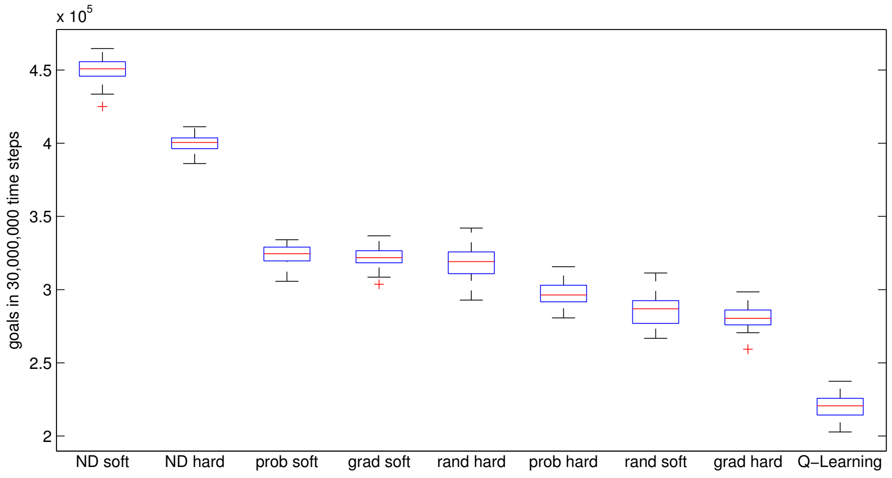

Figure 3: Box plot of accumulated goals occurrence for 50 executions within each pair of abstract policy and transfer strategy, and Q-learning (without learning).

formance in the target environment. Policy $\pi_{Abs_{ND}}$ is the worst abstract policy, but still much better than the random policy. Figure 3 shows the box plot for each pair of abstract policy and transfer strategy over accumulated number of goals in the whole experiment. Some interesting points can be noted in these results. First, despite the policy $\pi_{Abs_{ND}}$ has shown the worst performance when executed in the environment (without learning – see Table 1), it proved better than the other policies when it was used for transfer learning, which was a surprising result. Second, among abstract policies, the soft strategy is better than the hard strategy; but not for random policy. Third, usually ϵ-greedy is considered a good strategy for the exploration-exploitation dilemma for learning, but our experiments show that the hard strategy was a better option. Fourth, abstract policies $\pi_{Abs_{Grad}}$ and $\pi_{Abs_{Prob}}$ did not prove to be better than random policy for transfer learning. Fifth, all of the policies transfered proved to be better than Q-Learning with fixed ϵ.

Figure 4 shows the performance of each transfer pair $\langle$abstract policy,transfer strategy$\rangle$ averaged over 50 executions and 500,000 steps. All of the experiments show the same learning behavior: there is a critical point, when performance improves faster. Excepting random policy, the earlier the critical point, the better the performance in the final steps of the experiment. Even if the critical point occurs late when transferring random policy, the final performance competes with the best final performance.

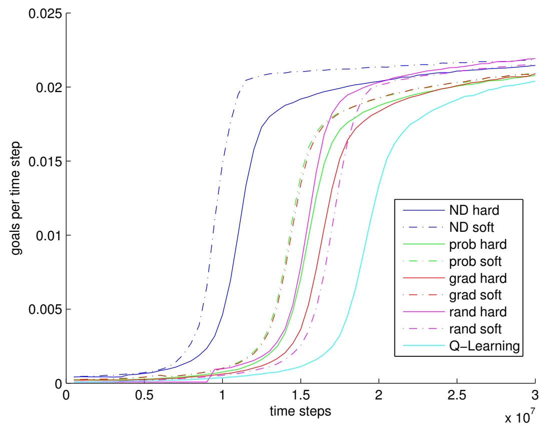

Figure 4: Performance in time. The average over 50 executions and 500,000 time steps for each pair of abstract policy and transfer strategy.

Conclusion

Non-deterministic induced policies showed to be the best policy to transfer learning, even if they are not the best choice when executed directly and without learning in the target environment. They work like an improved version of random policies, since only actions that proved to be optimal once are taken into account in the learning exploration. Surprisingly, deduced policies did not show good performance; which was not expected, since they are built under an optimality criterion.

Despite the experiments have been done in a unique type of problem, conclusions demand transfer learning to be made carefully, since policies can be ranked differently depending on whether policy is used only for execution or to transfer learning. Our experiments may stimulate further research to explain why we got such results. There is still much to be explored in the exploitation-exploration dilemma.

References

Banerjee, B., and Stone, P. 2007. General game learning using knowledge transfer. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, 672–677. Menlo Park: AAAI Press/IJCAI.

Bhatnagar, S.; Sutton, R. S.; Ghavamzadeh, M.; and Lee,

M. 2009. Natural actorcritic algorithms. *Automatica* 45(11):2471–2482.

Bianchi, R.; Martins, M.; Ribeiro, C.; and Costa, A. 2013. Heuristically-accelerated multiagent reinforcement learning. *Cybernetics, IEEE Transactions on* PP(99):1–1.

Blockeel, H., and De Raedt, L. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(12):285–297.

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

Celiberto, Jr., L. A.; Matsuura, J. P.; De Mantaras, R. L.; and Bianchi, R. A. C. 2011. Using cases as heuristics in reinforcement learning: a transfer learning application. In *Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI '11)*, 1211–1217. Menlo Park: AAAI Press/IJCAI.

Croonenborghs, T.; Driessens, K.; and Bruynooghe, M. 2007. Learning relational options for inductive transfer in relational reinforcement learning. In *Proccedings of the 17th Conference on Inductive Logic Programming (ILP '07)*, 88–97. New York: Springer.

da Silva, V. F.; Pereira, F. A.; and Costa, A. H. R. 2012. Finding memoryless probabilistic relational policies for inter-task reuse. In *Proceedings of the 14th International Conference on Information Processing and Management of Uncertainty (IPMU '12)*, volume 298 of *Communications in Computer and Information Science*, 107–116. Springer.

Dzeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43(1/2):7–52.

Fachantidis, A.; Partalas, I.; Tsoumakas, G.; and Vlahavas, I. 2013. Transferring task models in reinforcement learning agents. *Neurocomputing* 107:23–32.

Fernández, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*, 720–727.

Klenk, M.; Aha, D. W.; and Molineaux, M. 2011. The case for case-based transfer learning. *AI Magazine* 32(1):54.

Koga, M. L.; da Silva, V. F.; Cozman, F. G.; and Costa, A. H. R. 2013. Speeding-up reinforcement learning through abstraction and transfer learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '13)*, 119–126.

Koga, M. L.; Freire, V.; and Costa, A. H. R. 2014. Stochastic abstract policies: Generalizing knowledge to improve reinforcement learning. *Cybernetics, IEEE Transactions on* PP(99):1–1. DOI: 10.1109/TCYB.2014.2319733.

Konidaris, G.; Scheidwasser, I.; and Barto, A. 2012. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research* 13:1333–1371.

Lazaric, A.; Restelli, M.; and Bonarini, A. 2008. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, 544–551. New York, NY, USA: ACM.

Littman, M. L. 1994. Memoryless policies: theoretical limitations and practical results. In *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior: from animals to animats 3*, 238–245. Brighton: MIT Press.

Liu, Y., and Stone, P. 2006. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the twenty-first Natl. Conference on Artificial Intelligence*, 415–420. AAAI Press.

Martín, M., and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 9–19.

Mehta, N.; Natarajan, S.; Tadepalli, P.; and Fern, A. 2008. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning* 73(3):289–312.

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

Ramon, J.; Driessens, K.; and Croonenborghs, T. 2007. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proceedings of the eighteenth Eur. Conference Machine Learning (ECML '07)*, volume 4701 of *Lecture Notes in Computer Science*. 699–707.

Sehnke, F.; Osendorfer, C.; Rückstieß, T.; Graves, A.; Peters, J.; and Schmidhuber, J. 2008. Policy gradients with parameter-based exploration for control. In Kurkova, V.; Neruda, R.; and Koutník, J., eds., *Artificial Neural Networks - ICANN 2008*, volume 5163 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 387–396.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. Cambridge, MA: MIT Press.

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(1-2):181–211.

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

Taylor, M. E.; Stone, P.; and Liu, Y. 2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research* 8(1):2125–2167.

Toro, W. M.; Cozman, F. G.; Revoredo, K. C.; and Costa, A. H. R. 2014. Probabilistic relational reasoning in semantic robot navigation. In *Proceedings of the 10th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2014)*, volume 1259, 37–48. CEUR Workshop Proceedings. http://ceur-ws.org/Vol-1259/.

Torrey, L., and Shavlik, J. 2009. Transfer Learning. *Handbook of Research on Machine Learning Applications* 1–22.

Torrey, L. 2009. *Relational transfer in reinforcement learning*. Ph.D. Dissertation, University of Wisconsin-Madison.