Accelerating reinforcement learning by reusing abstract policies

Yannick Plaino Bergamo [1], Tiago Matos [1], Valdinei Freire da Silva [1], Anna Helena Reali Costa [1]

[1]Laboratório de Técnicas Inteligentes (LTI/EPUSP)
Escola Politécnica, Universidade de São Paulo
São Paulo, SP, Brazil

{yannick,tiago.matos}@usp.br, valdinei.freire@gmail.com, anna.reali@poli.usp.br

Abstract. *Reinforcement learning (RL) provides a general approach for developing intelligent agents that are able to optimize their behaviors in stochastic environments. Unfortunately, most work in RL is based on propositional representations, making it difficult to apply it to more complex real-world tasks in which states and actions are more naturally represented in relational form. Moreover, most work in RL does not take into account existing solutions to similar problems when learning a policy to solve a new problem, and consequently solves the new problem from scratch, what can be very time consuming. In this article we explore the powerful possibilities of using relational descriptions so that we can learn abstract policies, and in reusing these policies to improve initial performance of an RL learner in a similar new problem. Experiments carried out attest the effectiveness of our proposal.*

1. Introduction

Reinforcement learning (RL) techniques have demonstrated great success in the development of intelligent agents with the ability to optimize their behavior in stochastic environments. However, RL methods still suffer from some important problems, among which we highlight the slow convergence and the inability to reuse previously acquired knowledge in similar tasks.

Several solutions have been proposed to reuse knowledge – previously learned in the resolution of a similar problem or given by a specialist – to improve initial performance of an RL learner in a new problem. Some works propose the use of a heuristic function properly chosen, which is used to advise appropriate actions to be performed to guide the state space exploration during the learning process [Bianchi et al. 2007, Burkov and Chaib-draa 2007, Bianchi et al. 2008, Knox and Stone 2010]. Others propose to aggregate similar states in macro-states and use macro-actions to transit between macro-states. Then, macro-states and macro-actions are used to solve similar tasks [Drummond 2002]. There are also authors who have proposed to divide the task into subtasks, and find policies that solve those subtasks. These policies represented in an abstract form are then used to solve similar problems, together with a hierarchical representation of the aggregation states of the problem [Uther and Veloso 2002].

There is still work towards representing abstract states and actions in a relational form. Relational representations facilitate formulating broad collections of related tasks as a single domain, yielding natural generalization between these related tasks. Taylor and Stone [Taylor and Stone 2009] provide a thorough survey on the transfer of learning using propositional descriptions of the states. Cocora et al [Kersting et al. 2007] propose to learn relational decision trees as abstract navigation strategies from example paths, and then the navigation policy learned in one environment is directly applied to unknown environments. Based on samples experienced by the agent when acting optimally in the source task, in [Madden and Howley 2004] propositional symbolic learners generalize the optimal policy. In [Sherstov and Stone 2005] action relevance is used in order to reduce the action set to be explored in the target task.

In this paper we also explore the intuition that generalization from closely related, but solved, problems may often produce policies that make good decisions in many states of a new unsolved problem. Our proposal consists in abstracting policies from solved problems and then using these abstractions to significantly enhance the performance of the learning process in a new problem not yet solved.

The paper is organized as follows. Section 2 briefly reviews basic concepts of Markov decision processes (MDP), a sound theoretical framework to deal with uncertainty, and reinforcement learning, a method to learn policies by trial-and-error in problems modeled as MDPs. In Section 3 we provide a brief overview of relational reinforcement learning, so that we can define more precisely the concept of an abstract policy, which is the structure we use here to capture generalizations from problems. Then, in Section 4 we present our proposals to enable knowledge transfer to improve initial performance of an RL agent learning in a new problem. Section 5 reports an experiments carried to validate ours proposals, and section 6 summarizes our conclusions.

2. Markov decision processes

We will consider that our task can be modelled as a Markov Decision Process (MDP) [Puterman 1994]. At each (discrete) time step an agent observes the state of the system, chooses an action causing the system to evolve to a new state, returning a reward to the agent. Specifically an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, r \rangle$, where:

- $\mathcal{S}$, a discrete set of *states*;
- $\mathcal{A} = \bigcup_{\sigma \in \mathcal{S}} \mathcal{A}_\sigma$, a discrete set of *actions*, where $\mathcal{A}_\sigma$ is the set of allowable actions in state $\sigma \in \mathcal{S}$;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, a *transition function* such that $T(\sigma, \alpha, \sigma') = \mathbb{P}(s_{t+1} = \sigma' | s_t = \sigma, a_t = \alpha)$;
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$, a *reward function*, such that $r_{t+1} = r(\sigma, \alpha, \sigma')$ is the reward received when reaching, at time $t + 1$, state σ' having previously chosen action α at state σ;

the task of the agent is to find a *policy*, i.e. a sequence of decision rules, $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, such that $\pi(\sigma, \alpha) = \mathbb{P}(a_t = \alpha | s_t = \sigma)$. An *optimal policy* π^* is one that maximizes some function R_t of the future rewards $r_{t+1}, r_{t+2}, \ldots$ A common definition, which will be used for our discussion, is to consider the sum of *discounted rewards* $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $0 < \gamma < 1$ is the discount factor.

In our discussion we will be especially interested in a subclass of MDPs that are *episodic*, i.e. the reinforcement function implicitly defines a set $\mathcal{G}$ of one or more goal states, when the goal is reached the environment is restarted in some initial state which could be chosen according to some probability distribution. In this case we

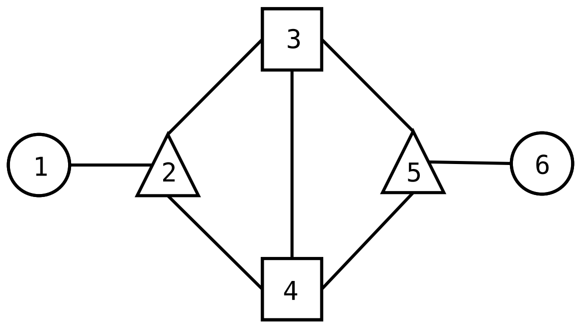

Figure 1. An example environment with 6 states.

also define [do Lago Pereira et al. 2008] a *probabilistic planning domain* to be the tuple $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, T \rangle$, and a *probabilistic planning problem* as the tuple $\mathcal{P} = \langle \mathcal{D}, \mathcal{G} \rangle$.

Example 1. Consider the environment represented on figure 1. Each geometric shape is a location that the agent can occupy, and if two locations are connected by an edge it means that the agent can reach one coming from the other. The state of the environment is represented by the location occupied by the agent[1]. We can define $\mathcal{S} = \{\sigma_1, \dots, \sigma_6\}$, where for example σ_2 represents the configuration in which the agent is on the triangle with the number 2. The set of actions $\mathcal{A} = \bigcup_{i=1}^{6} A_{\sigma_i} = \bigcup_{i=1}^{6} \{\alpha_{ij}, j \in \{1, \dots, 6\}\}$, represents a choice of transition for the agent, i.e. α_{12} means that the agent chooses to go to the triangle 2 when on state 1. Note that we also considered the possibility for the agent to stay in the state, denoted by α_{ii}. Let's assume for simplicity that the environment is deterministic, then $T(\sigma_i, a, \sigma_j)$ is equal to 1 if $a = \alpha_{ij}$ and zero otherwise. Our planning domain $\mathcal{D}$ is fully specified. Now suppose that we would like to learn how to reach state σ_6. We could define $r(s, a, s')$ to be equal to 0 if $s = s' = \sigma_6$ and -1 otherwise; with the reinforcement so defined, the agent maximizes R_t if he reaches σ_6 in the smallest number of steps and stays there afterwards. Thus the reinforcement function implicitly defines $\mathcal{G} = \{\sigma_6\}$, and our planning problem $\mathcal{P}$ is also specified. Since σ_2 contains two actions that are optimal, there are several possibilities for the optimal policy. We could have for example $\pi^*(\sigma_4, \alpha_{45}) = 1$, $\pi^*(\sigma_2, \alpha_{21}) = 0$ and $\pi^*(\sigma_2, \alpha_{23}) = \pi^*(\sigma_2, \alpha_{24}) = \frac{1}{2}$.

2.1. Reinforcement learning

Several methods are available for solving MDPs, in the present work we will consider Reinforcement Learning [Sutton and Barto 1998], which is particularly useful when T is not known to the agent. Within this framework it is common to define a *state action value function* $Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ that for a given policy π gives an estimate of the expected value $\mathbb{E}_\pi[R_t | s_t = \sigma, a_t = \alpha]$. This function is learned by direct interaction with the environment: at each time step the agent performs an action, observes the outcome and uses some strategy to update the estimate of the Q function; we refer to the book by Sutton and Barto for the details. With a good estimate of the Q function, the agent can maximize R_t by choosing $a_t = \arg\max_\alpha Q(\sigma, \alpha)$. It should be pointed out that both Q and π should be considered functions indexed by the time step t; this happens since Q is updated at (possibly) each time step, and π might change by, for example, taking into

[1]Note that there is a difference between a geometric shape that represents a location and a state: the state is a configuration of the environment, where the agent is at some location. This distinction will be particularly important on section 3.2

consideration the updated values of the Q function. In order to avoid heavy notation we won't make this dependence explicit.

Two problems arise from this approach. First, the agent must choose between exploring more the environment, in order to improve its estimate of Q, and exploiting his current knowledge to maximize his reward; second, if the goal, defined by the reinforcement function, is changed, the agent must learn everything from scratch. Our concern will be to alleviate the latter problem by abstracting a previously learned policy for a similar problem, using a relational language, to guide the initial exploration of the environment. By this we would like to provide a better initial performance for the agent, that would otherwise need to start exploring the environment randomly, while gradually replacing the previously learned policy with the one induced by the Q function he is currently learning.

Example 2. Back to the example of figure 1 and with $\mathcal{P}$ defined as in example 1, assuming the discounted reward model we can easily see that for example $Q^{\pi^*}(\sigma_4, \alpha_{45}) = -1 - \gamma$ and $Q^{\pi^*}(\sigma_4, \alpha_{43}) = -1 - \gamma - \gamma^2$. Note that α_{43} is not an optimal action; this is not a problem since Q^{π^*} tells us the value of R_t that we expect to obtain by choosing the action in the argument and then follow the optimal policy. As an illustration of the problem we face when the goal is changed, suppose that now we want to reach state σ_1. The symmetry of the figure makes it very clear how the problem is very similar to the one we solved before, but the reinforcement function has now changed to $r(s, a, s') = 0$ if $s = s' = \sigma_2$ and $r(s, a, s') - 1$ otherwise. We need to learn the Q^{π^*} from scratch since now for example $Q^{\pi^*}(\sigma_4, \alpha_{45}) = -1 - \gamma - \gamma^2 - \gamma^3$.

3. Relational representation

We begin this section with some definitions from first order logic. Then we provide a brief overview of relational reinforcement learning. With these concepts we finally make the concept of an abstract policy more precise.

3.1. Relational logic

In this subsection we present some definitions and notations from first order logic. The exposition is tailored to our needs; for a more formal treatment we refer to e.g. [Lloyd 1987].

A *relational alphabet* $\Sigma = P \cup C$ is composed of a set of predicates P and a set of *constants* C. If $\mathtt{t}_1, \ldots, \mathtt{t}_n$ are *terms*, i.e. each one is a variable or a constant, 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*. A *conjunction* is a set of atoms; in our discussion each variable in a conjunction will be implicitly assumed to be existentially quantified. We denote by $vars(A)$ the set of variables of a conjunction A. *Background Knowledge* BK is a set of Horn Clauses. A *substitution* θ is a set $\{\mathtt{X}_1/\mathtt{t}_1, \ldots, \mathtt{X}_n/\mathtt{t}_n\}$, binding each variable $\mathtt{X}_i$ to a term $\mathtt{t}_i$; it can be applied to a term, atom or conjunction. If A and B are conjunctions and there is a substitution θ such that $B\theta \subseteq A$, we say that A is *θ-subsumed* by B, denoted by $A \preceq_\theta B$. A term is called *ground* if it contains no variables; in a similar way we define ground atoms and ground conjunctions. The *Herbrand base* B_Σ is the set of all possible ground atoms that can be formed with the predicates and constants in Σ.

3.2. Relationally factored Markov Decision Processes

A relationally factored Markov Decision Process (RMDP) [van Otterlo 2004] is a tuple $\langle \Sigma, BK, T, r \rangle$, where $\Sigma = D \cup P \cup A$, is a relational alphabet such that:

- D is a set of constants representing the objects of the environment;
- P is a set of predicates used to describe relations and properties among objects;
- A is a set of action predicates.

The set of states $\mathcal{S}$ of the RMDP is the set of all $\sigma \subseteq B_{P \cup D}$ satisfying the integrity constraints imposed by BK. The set of actions is $\mathcal{A} = B_{A \cup D}$. With $\mathcal{S}$ and $\mathcal{A}$ defined, T and r have the same meaning as described in section 2.

From the definition we can see that an RMDP is an MDP where the states and the actions are represented (factored) through a relational language. All definitions from section 2 are still valid here, and so are the methods of solution. The advantage of working with such a representation is the possibility of aggregating sets of states and actions by using abstract states. Specifically, an *abstract state* $\hat{\sigma} \in \widehat{\mathcal{S}}$ is a conjunction over P, D and BK; an *abstract action* $\hat{\alpha} \in \widehat{\mathcal{A}}$ is an atom over A and D. We can also define $\mathcal{S}_{\hat{\sigma}}$, the set of ground states covered by $\hat{\sigma}$, as $\mathcal{S}_{\hat{\sigma}} = \{\sigma \in \mathcal{S} | \sigma \preceq_\theta \hat{\sigma}\}$. Similarly we define $\mathcal{A}_{\hat{\alpha}} = \{\alpha \in \mathcal{A} | \alpha \preceq_\theta \hat{\alpha}\}$

Example 3. Let's review example 1 by using an RMDP model. There are many ways we can choose Σ; as an example let's consider $D = \{\texttt{c1}, \texttt{t2}, \texttt{s3}, \texttt{s4}, \texttt{t5}, \texttt{c6}\}$; $P = \{\texttt{in/1}, \texttt{circle/1}, \texttt{square/1}, \texttt{triangle/1}, \texttt{connected/2}\}$; $A = \{\texttt{goto/2}, \texttt{stay/1}\}$. The meaning of each predicate should be intuitive, we give one example to make it clearer: if the agent is occupying location 1 in the map, which we are representing as the object c1, we can describe the state of the environment as $\sigma_1 = \{$in(c1),circle(c1),connected(c1,t2),triangle(t2)$\}$. If the agent makes a choice to go from location c1 to location t2, we write this by goto(c1,t2). We can also consider some abstract states, for example: $\hat{\sigma}_1 = \{$in(X),connected(X,Y),circle(Y)$\}$ represents all states in which the agent is in a location connected to a circle, i.e. $\mathcal{S}_{\hat{\sigma}_1} = \{\sigma_2, \sigma_5\}$; $\hat{\sigma}_2 = \{$in(X),circle(X)$\}$, represents all states in which the agent is in a location that is a circle, i.e. $\mathcal{S}_{\hat{\sigma}_2} = \{\sigma_1, \sigma_6\}$. In $\hat{\sigma}_1$ the agent could consider the action goto(X,Y), and in $\hat{\sigma}_2$ he could consider to stay on that set, i.e. the action stay(X).

3.3. Abstract policies

We can now define an *abstract policy* as a list of abstract action rules [van Otterlo 2004] $\hat{\sigma} \xrightarrow{p_{\hat{\sigma},i}} \hat{\alpha}_i$, for $\hat{\sigma} \in \widehat{\mathcal{S}}$, $\hat{\alpha}_i \in \widehat{\mathcal{A}}$ satisfying $vars(\hat{\alpha}_i) \subseteq vars(\hat{\sigma})$; $p_{\hat{\sigma},i}$ is the probability of choosing $\hat{\alpha}_i$ when on a state covered by $\hat{\sigma}$.

There are several way of obtaining such a list. One way would be to solve the RMDP in the abstract setting [van Otterlo 2004, Kersting et al. 2004]. Another possibility, that would avoid giving the dynamics of the environment in a relational language, could be to learn an abstract policy by inducting a logical decision tree [Blockeel and De Raedt 1997] from the examples generated by applying the learned optimal policy for a specific problem: the set of abstract states is the set made by the union of all conjunctions in a path from the root to a leaf, and the set of abstract actions is the set of all abstract actions contained in the leafs. An abstract policy given by a teacher could also be an option.

An abstract policy induces a policy in the ground MDP, which we will name π_a.

Given a state σ, we find the first[2] $\hat{\sigma}$ such that $\sigma \preceq_\theta \hat{\sigma}$. We then have a set of abstract actions $\hat{\alpha}_i$, and associated probabilites. We could use those ones to assign the probability $\pi_a(\sigma, a)$, where $a \in \{\alpha \in \mathcal{A}_\sigma | \alpha \preceq_\theta \hat{\alpha}_i\}$. It should be pointed out that depending on how the abstract policy was obtained there could be no $\hat{\sigma}$ in the list for a given σ. In this case we could select randomly from $\mathcal{A}_\sigma$.

Example 4. Considering the RMDP model of example 3, one abstract policy could be:

$$\texttt{in(X),circle(X)} \xrightarrow{1.0} \texttt{stay(X)} \tag{1}$$

$$\texttt{in(X),connected(X,Y),circle(Y)} \xrightarrow{1.0} \texttt{goto(X,Y)} \tag{2}$$

$$\texttt{in(X),connected(X,Y),triangle(Y)} \xrightarrow{1.0} \texttt{goto(X,Y)} \tag{3}$$

Note how the order is important, since rule (1) applied to state σ_1 would suggest action α_{11}, while rule (3) would suggest α_{12}.

4. Using the abstract policy to guide initial exploration

4.1. Initial considerations

Let's start by recalling that we are dealing with a probabilistic planning domain, and that we want to provide a better initial performance by using some policy that uses previously acquired knowledge, while gradually learning the optimal one. We briefly add some notation to make our argument more compact. Since we are dealing with planning problems, we can consider that each task is represented by a tuple $\langle \sigma_0, \sigma_g \rangle$, where $\sigma_0 \in \mathcal{S}$ and $\sigma_g \in \mathcal{G}$. Let's also define $\mathcal{T}(\pi, \langle \sigma_0, \sigma_g \rangle) = \{s_1 s_2 \ldots s_n \text{ for some } n | s_1 = \sigma_0, s_n = \sigma_g, T(s_i, a_i, s_{i+1}) \neq 0 \text{ for some } a_i \text{ such that } \pi(s_i, a_i) \neq 0\}$, i.e. the set of all possible transition histories that start on state σ_0, and, by following policy π, reach at some time step the goal state σ_g.

Example 5. With $\mathcal{P}$ and π^* defined as in example 1, we have $\mathcal{T}(\pi^*, \langle \sigma_3, \sigma_6 \rangle) = \{\sigma_3\sigma_5\sigma_6\}$ and $\mathcal{T}(\pi^*, \langle \sigma_2, \sigma_6 \rangle) = \{\sigma_2\sigma_3\sigma_5\sigma_6, \sigma_2\sigma_4\sigma_5\sigma_6\}$.

In the following discussion π^* is the optimal policy for the task $\langle \sigma_0, \sigma_g \rangle$; π_r is a random policy that at each state σ chooses an action with uniform distribution from the set $\mathcal{A}_\sigma$; π_a is the policy induced by the abstract policy, as defined on section 3.3. Also let's define π_Q to be the policy that acts greedily with respect to the Q function, i.e. $\pi_Q(\sigma, \alpha^*) = 1$ if $\alpha^* = \arg\max_\alpha Q(\sigma, \alpha)$.

4.2. Using the abstract policy

The usual approach in reinforcement learning is to start learning the Q function estimate by random exploration of the environment, while gradually making use of the Q function to choose actions according to π_Q; algorithms such as ϵ-greedy and softmax action selection work in this way [Sutton and Barto 1998]. In our method we are going to make use of the previously learned knowledge contained in π_a for the initial exploration. However, for reasons that we will later explain, we are not going to fully replace π_r by π_a.

[2]Note that the abstract policy was defined as a list of action rules, i.e. order matters. In the case of a tree this mapping is also unambiguous.

We start by giving a better explanation as to why π_a should be better than π_r in the initial learning phase. If, facing a new task $\langle\sigma_0, \sigma_g\rangle$, the agent starts using π_r, he will consider all transitions in $\mathcal{T}(\pi_r, \langle\sigma_0, \sigma_g\rangle)$. On the other hand, we have $\mathcal{T}(\pi_a, \langle\sigma_0, \sigma_g\rangle) \subset \mathcal{T}(\pi_r, \langle\sigma_0, \sigma_g\rangle)$, and if we assume that π_a was learned considering a similar task, the actions that will be taken into consideration by the agent have the property they were good to solve a similar problem before. Therefore he should be able to obtain a better initial performance, i.e. a higher value of R_t in the first episodes when compared to the value he would obtain by only considering π_r.

Example 6. Let's go back to example 1, and let's consider π_a induced by the abstract policy of example 4. As previously hinted on example 2, if we change the goal from σ_6 to σ_2 the agent needs to learn everything from scratch. However note that for example $\mathcal{T}(\pi_a, \langle\sigma_3, \sigma_1\rangle) = \{\sigma_3\sigma_2\sigma_1, \sigma_3\sigma_5\sigma_6\}$, while $\{\sigma_3\sigma_2\sigma_1, \sigma_3\sigma_5\sigma_6, \sigma_3\sigma_4\sigma_2\sigma_1, \sigma_3\sigma_5\sigma_4\sigma_2\sigma_1\} \subset \mathcal{T}(\pi_r, \langle\sigma_3, \sigma_1\rangle)$, which is a countably infinite set.

As to why the agent should still consider π_r and gradually change to π_Q, we should note that not necessarily we have $\mathcal{T}(\pi^*, \langle\sigma_0, \sigma_g\rangle) \subset \mathcal{T}(\pi_a, \langle\sigma_0, \sigma_g\rangle)$, i.e. π_a could lead to non-optimal actions, or could not be able to solve the task. Therefore the agent still needs to explore actions not suggested by π_a, thereby justifying the use of π_r. Since proper convergence of the Q functions guarantees that $\pi_Q = \pi^*$ [Sutton and Barto 1998], in the long run the agent should act only according to π_Q to maximize his reward.

To make things more formal, we will consider that at each time step t the agent uses one of the policies in the set $\{\pi_Q, \pi_r, \pi_a\}$ probabilistically, i.e., we assume that we have two functions $\rho_t : \mathcal{S} \mapsto [0,1]$ and $\gamma_t : \mathcal{S} \mapsto [0,1]$ such that:

$$\pi(\sigma, \alpha) = \begin{cases} \pi_Q(\sigma, \alpha), & \text{with probability } \rho_t(\sigma) \\ \pi_r(\sigma, \alpha), & \text{with probability } [1-\rho_t(\sigma)]\gamma_t(\sigma) \\ \pi_a(\sigma, \alpha), & \text{with probability } [1-\rho_t(\sigma)][1-\gamma_t(\sigma)] \end{cases} \tag{4}$$

From what we explained, both γ_t and ρ_t should converge to 1, in the limit, with a faster convergence rate for γ_t in order to guarantee that the environment was properly explored before using exclusively π_Q. As it can be seen, if we take γ_t to be identically equal to 1, we go back to the usual methods in reinforcement learning that we mentioned in the beginning of this subsection

5. A navigation problem

As an example of what we expect to obtain, consider the map on figure 2. We assigned some reference points represented by geometrical shapes, each having a meaning with respect to the semantic of the map, e.g. a square is the center of a room. We then modeled the map relationally, in a way similar to that of example 3.

Specifically, we define $P = \{$`isCorridor/1`, `isRoom/1`, `isCenter/1`, `isNearDoor/1`, `in/1`, `isConnected/2`$\}$; and $A = \{$`gotoRDRD/2`, `gotoCDCD/2`, `gotoRCRD/2`, `gotoRDRC/2`, `gotoRDCD/2`, `gotoCDRD/2`, `gotoCCCD/2`, `gotoCDCC/2`$\}$. To make notation more compact, we are considering the following abbreviations: `cd(X)` $=$ $\{$`isCorridor(X),isNearDoor(X)`$\}$; `cc(X)` $=$

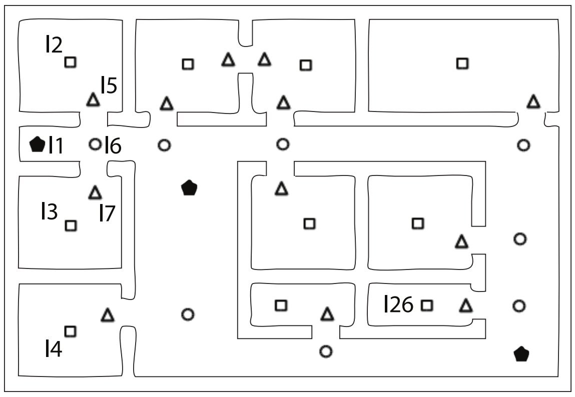

Figure 2. Map used in our navigation problem.

{`isCorridor(X),isCenter(X)`}; `rc(X)` = {`isRoom(X),isCenter(X)`}; `rd(X)` = {`isRoom(X),isNearDoor(X)`}.

The meaning of each predicate should be intuitive, we give one example to make it clearer: if the agent is occupying location 1 in the map, which we are representing as the object `l1` $\in D$, we can describe the state as σ_{11} = {`in(l1)`, `isCorridor(l1)`, `isCenter(l1)`, `isConnected(l1,l6)`, `isCorridor(l6)`, `isNearDoor(l6)`}. If the agent chooses to go from location σ_{11} in the map of Figure 2 to location σ_{16} we write this action by `gotoCCCD(l1,l6)`.

We started by learning the optimal policy to reach center room σ_{12} in the map, and then generated some examples by applying the optimal policy in a few tasks, where the goal state was kept fixed and the starting state was selected randomly. We then obtained an abstract policy by inducting a first order logical decision tree (FOLDT), using the TILDE algorithm [Blockeel and De Raedt 1997]. In the FOLDT each path from the root to a leaf represents an abstract state, and each leaf is a set of abstract actions. Recall from section 3.3 that each abstract action rule is of the form $\hat{\sigma} \xrightarrow{p_{\hat{\sigma},i}} \hat{\alpha}_i$; in our case we considered $p_{\hat{\sigma},i}$ to be the uniform distribution. To show the resulting abstract policy we converted the FOLDT into a list by doing a depth-first search; the result is shown below:

$$\texttt{in(X),cd(X),cc(Y)} \xrightarrow{0.33} \texttt{gotoCDCD(X,Z)}$$
$$\texttt{in(X),cd(X),cc(Y)} \xrightarrow{0.33} \texttt{gotoCDCC(X,Y)}$$
$$\texttt{in(X),cd(X),cc(Y)} \xrightarrow{0.33} \texttt{gotoCDRD(X,Z)}$$
$$\texttt{in(X),cd(X)} \xrightarrow{1.0} \texttt{gotoCDCD(X,Z)}$$
$$\texttt{in(X),connected(X,Y),rc(Y)} \xrightarrow{0.33} \texttt{gotoRDCD(X,Z)}$$
$$\texttt{in(X),connected(X,Y),rc(Y)} \xrightarrow{0.33} \texttt{gotoRDRC(X,Y)}$$
$$\texttt{in(X),connected(X,Y),rc(Y)} \xrightarrow{0.33} \texttt{gotoRDRD(X,Z)}$$
$$\texttt{in(X),rc(Z)} \xrightarrow{1.0} \texttt{gotoRCRD(X,W)}$$
$$\texttt{in(X)} \xrightarrow{1.0} \texttt{gotoCCCD(X,W)}$$

In order to take full advantage of all available information in the abstract policy, we further

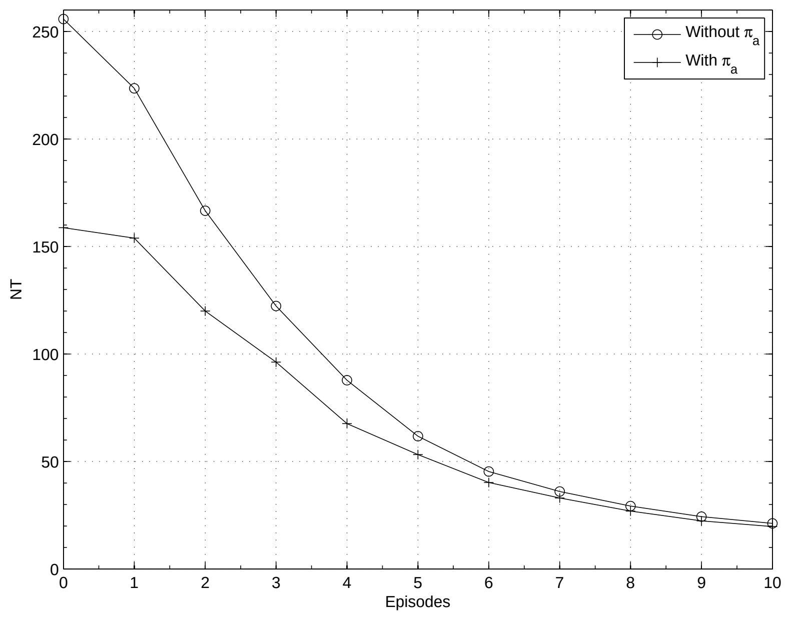

Figure 3. Average number of transitions NT required to reach the goal, with and without the use of the abstract policy π_a.

considered that a ground action suggested by the abstract policy could not be suggested again by π_a in the same episode, i.e. if during some episode the agent is at state σ and if the abstract policy suggests action $\alpha \in \mathcal{A}_\sigma$, then the next time, during the same episode, being the agent at state σ and using π_a for choosing the action to be taken, the set of possible ground actions suggested by π_a is reduced to $\mathcal{A}_\sigma - \{\alpha\}$. By episode we mean a sequence of steps that takes the robot from an initial location to the goal state, as discussed in section 2.

We assumed that the abstract policy could guide the robot in a sub-optimal way to the goal state, so that we initialized $\rho_0 = 0$ and $\gamma_0 = 0$, and at each episode we incremented ρ by 0.05 and γ by 0.06, according to what we discussed in section 4.2.

After this initial setup, we used the abstract policy to solve a new task. Specifically, we considered task $\langle \sigma_{11}, \sigma_{126} \rangle$, and we looked at the average number NT of transitions (steps) required to reach the goal at each episode. We repeated this task 1000 times and took the average value for each episode. The result for the first 10 episodes can be seen on Figure 3.

The strategy that uses π_a starts with a smaller average number of transitions and gradually decreases this value due to the learning process. The strategy that uses only π_r starts with a higher average number of steps. This difference in the beginning of the curves shows that the abstract policy can transfer knowledge between problems, thus obtaining a better initial performance as claimed.

During the learning process the distance between the curves is reduced (approximately in the fifth episode). This is due to the fact that π_a is gradually replaced by π_r during the learning process (by increasing γ_t) to guarantee the convergence of the Q function estimate.

6. Conclusions

In this work we have proposed a method to take into account existing solutions to similar problems when learning a policy to solve a new problem, so that we could improve initial performance of an RL learner. We explore the powerful possibilities of using relational representations, which facilitate policy abstraction and generalization between collections of related tasks. Our proposal consisted in abstracting policies from solved problems and then using these abstractions in a combined way in an RL learner, in order to significantly enhance the initial performance of the learning process in a new problem not yet solved.

Empirical evaluation of our approach in the robotic navigation problems were carried out. Experimental results showed that the performance of the learning algorithm can be improved even using a very simple abstract policy.

An important topic to be investigated in future works is concurrently to improve the abstract policy using experiences conducted in the current learning process so that this policy could generalize a wider class of similar problems and improve the current and future learning processes with better policy directions.

Acknowledgments

We are grateful to FAPESP (grants 10/02379-9, 09/04489-9, 09/14650-1, 8/03995-5) and to CNPQ (grant 475690/2008-7) for the support.

References

Bianchi, R. A. C., Ribeiro, C. H. C., and Costa, A. H. R. (2007). Heuristic selection of actions in multiagent reinforcement learning. In *IJCAI*, pages 690–695.

Bianchi, R. A. C., Ribeiro, C. H. C., and Costa, A. H. R. (2008). Accelerating autonomous learning by using heuristic selection of actions. *Journal of Heuristics*, 14(2):135–168.

Blockeel, H. and De Raedt, L. (1997). Top-down induction of logical decision trees. In *Artificial Intelligence*.

Burkov, A. and Chaib-draa, B. (2007). Adaptive play q-learning with initial heuristic approximation. In *ICRA*, pages 1749–1754.

do Lago Pereira, S., de Barros, L., and Cozman, F. (2008). Strong probabilistic planning. In Gelbukh, A. and Morales, E., editors, *MICAI 2008: Advances in Artificial Intelligence*, volume 5317 of *Lecture Notes in Computer Science*, pages 636–652. Springer Berlin / Heidelberg.

Drummond, C. (2002). Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, 16:59–104.

Kersting, K., Otterlo, M. V., and Raedt, L. D. (2004). Bellman goes relational. In *In ICML*, pages 465–472. ACM.

Kersting, K., Plagemann, C., Cocora, A., Burgard, W., and Raedt, L. D. (2007). Learning to transfer optimal navigation policies. *Advanced Robotics: Special Issue on Imitative Robots*, 21(13):1565—1582.

Knox, W. B. and Stone, P. (2010). Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*.

Lloyd, J. W. (1987). *Foundations of Logic Programming, 2nd Edition*. Springer Verlag.

Madden, M. G. and Howley, T. (2004). Transfer of experience between reinforcement learning environments with progressive difficulty. *Artif. Intell. Rev.*, 21:375–398.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

Sherstov, A. A. and Stone, P. (2005). Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685.

Uther, W. T. B. and Veloso, M. M. (2002). Ttree: Tree-based state generalization with temporally abstract actions. In *In Proceedings of SARA-2002*, pages 260–290.

van Otterlo, M. (2004). Reinforcement learning for relational MDPs. In Nowe, A., Lenaerts, T., and Steenhaut, K., editors, *Proceedings of the Machine Learning Conference of Belgium and the Netherlands, BeNeLearn '04*, pages 138–145, Brussels. Brussels.