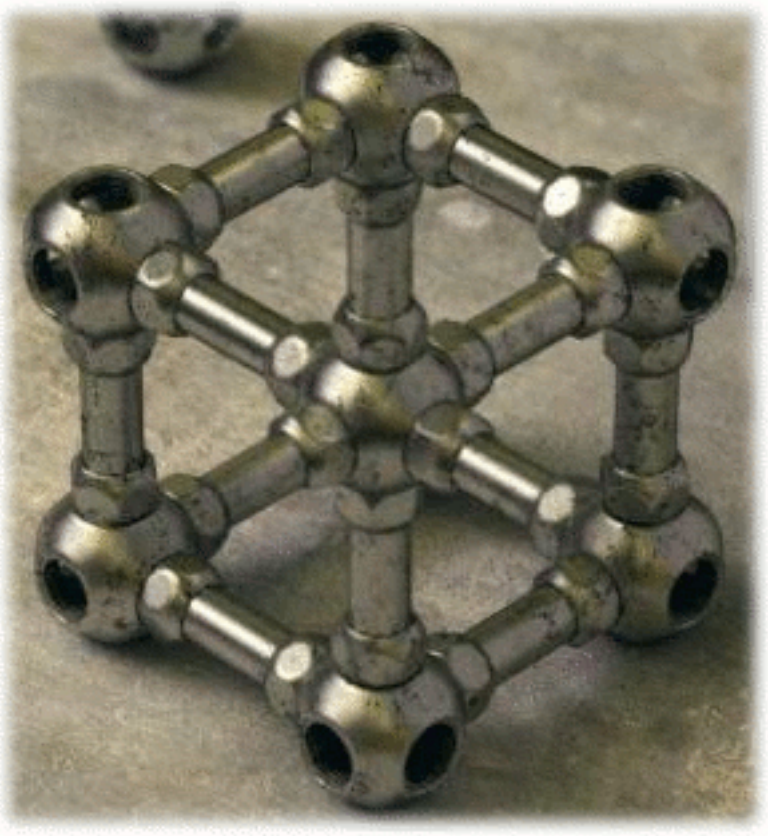

Simultaneous Abstract and Concrete Reinforcement Learning [1]

Author(s):

Tiago Matos
Yannick P. Bergamo
Valdinei F. da Silva
Fabio G. Cozman
Anna H. Reali Costa

[1]This work was supported by Fapesp Project LogProb, grant 2008/03995-5, São Paulo, Brazil.

Simultaneous Abstract and Concrete Reinforcement Learning

Tiago Matos and **Yannick P. Bergamo** and **Valdinei F. da Silva** and
Fabio G. Cozman and **Anna H. Reali Costa**
Escola Politécnica, Universidade de São Paulo
São Paulo, SP, Brazil
{tiago.matos, yannick}@usp.br, valdinei.freire@gmail.com,
fgcozman@usp.br, anna.reali@poli.usp.br

Abstract

Suppose an agent builds a policy that satisfactorily solves a decision problem; suppose further that some aspects of this policy are abstracted and used as starting point in a new, different decision problem. How can the agent accrue the benefits of the abstract policy in the new concrete problem? In this paper we propose a framework for *simultaneous* reinforcement learning, where the abstract policy helps start up the policy for the concrete problem, and both policies are refined through exploration. We report experiments that demonstrate that our framework is effective in speeding up policy construction for practical problems.

1 Introduction

In this paper we consider the following general question. An agent builds a policy that solves a particular decision problem, say a policy that takes a robot from one point to a goal. The agent now faces a similar problem: perhaps the robot is in an adjacent building; or perhaps an online trader is facing a crisis that reminds her of last year's crisis. The agent wishes to abstract the policy produced for the first problem so as to have a reasonable initial policy, possibly to be refined, for the second problem. In this paper we consider abstract policies that are encoded through relational representations: the concrete policy for the first problem is analyzed and logical variables replace constants of the first problem. The abstract policy must then be applied to the second problem. Our question is, How best to do that?

No matter how smart we may be in constructing the abstract policy from the first problem's concrete policy, a direct application of the abstract policy in the second problem will certainly generate sub-optimal behavior. Because we have no guarantees as to the performance of the abstract policy, we should refine the second problem's concrete policy. The obvious technique to use is reinforcement learning, where the agent refines the policy by exploration of the task. We can go a step further, recognizing that the abstract policy can itself be quickly tuned through exploration, so as to maximize benefits from abstraction. The challenge is to create a framework that both guarantees convergence to optimality and uses as much prior knowledge and experiments as possible. In this paper we propose a framework for simultaneous reinforcement learning of the abstract *and* the concrete policies that does just that.

In doing so, we bypass some of the difficulties of reinforcement learning, namely the slow convergence to optimal policy and the inability to reuse previously acquired knowledge. The prior knowledge for the second problem is exactly the abstract policy. 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. At the same time, we avoid the possible bad performance of a known policy in a new problem, by applying reinforcement learning as much as possible.

Our experiments show that such a framework produces excellent results. An abstract policy can offer effective guidance and benefit from exploration; moreover, learning in the abstract level converges faster than in the concrete level.

The paper is organized as follows. Section 2 reviews basic concepts of Markov decision processes and reinforcement learning, including Relationally Factored MDP, the basis for the concepts of abstract states and abstract actions. Section 3 discusses abstract policies, how to induce and use them. Section 4 presents our proposals 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 Markov Decision Processes and Reinforcement Learning

We assume that our decision problems can be modeled as Markov Decision Processes (MDPs) (Puterman 1994). At each (discrete) time step an agent observes the state of the system, chooses an action and moves to another state. For our purposes an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, r \rangle$, where:

- $\mathcal{S}$ is a discrete set of *states*;
- $\mathcal{A} = \bigcup_{\sigma \in \mathcal{S}} \mathcal{A}_\sigma$ is a discrete set of *actions*, where $\mathcal{A}_\sigma$ is the set of allowable actions in state $\sigma \in \mathcal{S}$;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is a *transition function* such that $T(\sigma, \alpha, \sigma') = \mathbb{P}(s_{t+1} = \sigma' | s_t = \sigma, a_t = \alpha)$;

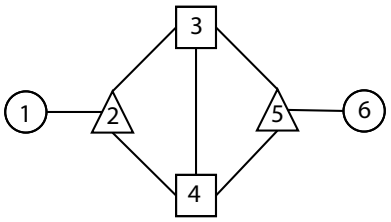

Figure 1: An example environment with 6 states.

- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is a *reward function*, such that $r_{t+1} = r(\sigma, \alpha, \sigma')$ is the reward received when reaching state σ' at $t + 1$ having previously chosen action α at state σ.

The task of the agent is to find a *policy*. A general policy π is a function from $\mathcal{S}$ into the power set of $\mathcal{A}$ that maps a state into a *set* of actions: $\pi : \mathcal{S} \mapsto 2^{\mathcal{A}}$. If a policy maps each state into a singleton (that is, into a single action), we say the policy is *deterministic*. If a policy maps some states into sets of actions containing more than one action, we say the policy is *nondeterministic*. An *optimal deterministic policy* π^* (or, more compactly, an optimal policy) is a deterministic policy that maximizes some function R_t of the future rewards $r_{t+1}, r_{t+2}, \ldots$ A common definition, which we use, 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*.

We are particularly interested in a subclass of MDPs that are *episodic*: there is a set $\mathcal{G}$ of goal states, and when the goal is reached the problem is restarted in some initial state chosen according to a probability distribution. We also define (do Lago Pereira, de Barros, and Cozman 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 $\mathcal{P} = \langle \mathcal{D}, \mathcal{G} \rangle$.

Consider the environment depicted in 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 of the agent. Define $\mathcal{S} = \{\sigma_1, \ldots, \sigma_6\}$, where for instance σ_2 represents the configuration in which the agent is on the triangle with the number 2. The set of actions is $\mathcal{A} = \bigcup_{i=1}^{6} A_{\sigma_i} = \bigcup_{i=1}^{6} \{\alpha_{ij}, j \in \{1, \ldots, 6\}\}$, where for instance α_{12} means that the agent chooses to go to the triangle 2 when on state σ_1. Note that the agent can stay in the state; such an action is denoted by α_{ii}. 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 reward so defined, the agent maximizes R_t if she reaches σ_6 in the smallest number of steps and stays there afterwards. Thus the reward function implicitly defines $\mathcal{G} = \{\sigma_6\}$, and our planning problem $\mathcal{P}$ is also specified. Because σ_2 contains two actions that are optimal, there are several possibilities for the optimal policy. We have for example $\pi^*(\sigma_2) = \alpha_{23}$ or $\pi^*(\sigma_2) = \alpha_{24}$. We can encode this into a nondeterministic policy as $\pi^*(\sigma_2) = \{\alpha_{23}, \alpha_{24}\}$. To determine a particular action out of this nondeterministic policy, we can randomize; for instance, we might select either α_{23} or α_{24} each with probability $1/2$.

Given an MDP, there exists a deterministic policy that maximizes discounted rewards (Puterman 1994); there are several methods for finding such a policy. When T is not given, one may resort to Reinforcement Learning (RL) (Sutton and Barto 1998). 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, she observes the outcome and uses some strategy to update the estimate of the Q function. With a good estimate of the Q function, the agent can maximize R_t by choosing $a_t = arg\,max_\alpha\, Q(\sigma, \alpha)$.

Back to the example of Figure 1 and with $\mathcal{P}$ defined as before, and 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 by following the optimal policy. As an illustration of the challenges we face when the goal is changed, suppose that now we want to reach state σ_1. The symmetry of the figure makes it clear that the problem is very similar to the one we solved before, but the reward 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 $Q^{\pi^*}(\sigma_4, \alpha_{45}) = -1 - \gamma - \gamma^2 - \gamma^3$.

At any given time, the agent must choose between exploring the environment, in order to improve its estimate of Q, or exploiting her current knowledge to maximize his reward. A difficulty of this whole enterprise is that if the goal, specified through the reward function, is changed, the agent must learn everything from scratch. Indeed, reinforcement learning suffers from slow convergence to optimal policy, and inability to reuse previously acquired knowledge. Two directions can be taken in order to accelerate policy learning: *i*) choosing wisely which actions to be explored first and *ii*) sharing experiences among states through abstraction. In both case, either *a priori* domain knowledge must be introduced or knowledge must be reused from previously solved tasks in similar domains.

Some proposals use of heuristic functions to select actions so as to guide the exploration process (Burkov and Chaib-draa 2007; Bianchi, Ribeiro, and Costa 2008; Knox and Stone 2010). Because it is possible that the heuristic function does not work well, some random exploration must also be done in order to guarantee convergence. Then, even if an unfit heuristic function is chosen, the optimal policy can still be learned. Other proposals have similar states (with respect to transition function, reward function, Q-value functions, optimal policies) share experiences (Li, Walsh, and Littman 2006). An interesting way to reuse knowledge of previously solved tasks in the same domain is to em-

ploy temporal hierarchies (Barto and Mahadevan 2003). In that approach, a task is divided into subtasks from which optimal policies can be learned independently; the task is solved by appropriately compounding subtasks. When a subtask is learned, it can be transferred easily inter and intra-task with the use of abstraction (Uther and Veloso 2002; Drummond 2002).

There is also work towards representing abstract states and actions in a relational form, an effort that is closely related to our concerns in this paper. Relational representations facilitate formulating broad collections of related tasks as a single domain, yielding natural abstraction between these related tasks. Taylor and Stone (2009) provide a thorough survey on the transfer of learning using propositional descriptions of the states. In the domain of robotic navigation, Cocora et al (2006) 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 a given decision problem, Madden and Howley (2004) show how propositional symbolic learners can generalize the optimal policy for other problems. In Sherstov and Stone (2005) action relevance is used in order to reduce the action set to be explored in the target problem.

Our main interest is to abstract details of a previously constructed policy, and to use this abstract policy in a new problem. The abstract policy is here expressed in a relational language. Hence we need a few definitions from first order logic. For a more formal treatment we refer to 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 $\mathtt{p(t_1 \ldots t_n)}$ 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.

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 for MDPs.

From the definition we can see that an RMDP is an MDP where the states and the actions are represented (in factored form) through a relational language. All definitions for MDPs are still valid here, and so are the methods of solution.

Consider again Figure 1, but now using an RMDP model. There are many ways we can choose Σ; as an example, $D = \{\mathtt{c1}, \mathtt{t2}, \mathtt{s3}, \mathtt{s4}, \mathtt{t5}, \mathtt{c6}\}$; $P = \{\mathtt{in/1}, \mathtt{circle/1}, \mathtt{square/1}, \mathtt{triangle/1}, \mathtt{connected/2}\}$; $A = \{\mathtt{goto/2}, \mathtt{stay/1}\}$. For instance, if the agent is occupying location 1 in the map (represented as object `c1`), we can describe the state of the environment as $\sigma_1 = \{\mathtt{in(c1)}, \mathtt{circle(c1)}, \mathtt{connected(c1,t2)}, \mathtt{triangle(t2)}\}$. If the agent makes a choice to go from location `c1` to location `t2`, we write this by `goto(c1,t2)`.

We note that OO-MDPs (Diuk, Cohen, and Littman 2008) offer a similar formalism for relational representation of MDPs. The difference between these formalisms lies in the transition dynamics of the environment and in the attributes representing states. In OO-MDPs, only information that can be perceived by the agent is used; in RMDPs, predicates defined by a designer is used instead.

3 Abstracting Policies

A relational representation for MDPs can be used to encode aggregate states and actions. We start by defining a few useful concepts:

- 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.
- Denote by $\mathcal{S}_{\hat{\sigma}}$ the set of ground states covered by $\hat{\sigma}$, i.e., $\mathcal{S}_{\hat{\sigma}} = \{\sigma \in \mathcal{S} | \sigma \preceq_\theta \hat{\sigma}\}$.
- Similarly define $\mathcal{A}_{\hat{\alpha}} = \{\alpha \in \mathcal{A} | \alpha \preceq_\theta \hat{\alpha}\}$.

Regarding our running example, take two abstract states as follows. First, $\hat{\sigma}_1 = \{\mathtt{in(X)}, \mathtt{connected(X,Y)}, \mathtt{circle(Y)}\}$ represents all states in which the agent is in a location connected to a circle, i.e. $\mathcal{S}_{\hat{\sigma}_1} = \{\sigma_2, \sigma_5\}$. Second, $\hat{\sigma}_2 = \{\mathtt{in(X)}, \mathtt{circle(X)}\}$, 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 can contemplate the action `goto(X,Y)`, and in $\hat{\sigma}_2$ he can decide to stay on that set by taking action `stay(X)`.

We can now define an *abstract policy* (van Otterlo 2004) as a list of abstract action rules of the form $\hat{\sigma} \rightarrow \{\hat{\alpha}_i\}_i$ for $\hat{\sigma} \in \widehat{\mathcal{S}}$, $\hat{\alpha}_i \in \widehat{\mathcal{A}}$; that is, rules that map an abstract state into a set of abstract actions.

Consider the RMDP for the example in Figure 1; an abstract policy could be:

(1) `in(X),circle(X)` $\rightarrow$ `stay(X)`
(2) `in(X),connected(X,Y),circle(Y)` $\rightarrow$ `goto(X,Y)`
(3) `in(X),connected(X,Y),triangle(Y)` $\rightarrow$ `goto(X,Y)`

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

3.1 Abstract policy from a given concrete policy

If we are given an RMDP, we can obtain an optimal abstract policy through symbolic dynamic programming (van Otterlo 2004; Kersting, Otterlo, and Raedt 2004). In this paper we "learn" an abstract policy by inducing a logical decision tree (Blockeel and De Raedt 1998) from examples generated by the previously produced concrete policy. The set of abstract states is read off of this logical decision tree, as 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 leaves. The idea here is to learn an abstract policy from a problem and then to use it as a starting point in a new, different problem.

Take a set of examples E, where an example consists of a state-action pair taken from a previously constructed policy. We use a variant of the TILDE (Blockeel and De Raedt 1998) algorithm to induce an abstract policy from E.

(ND-)TILDE is an inductive algorithm that produces an abstraction of the given concrete policy, and represents the abstract policy by a first order logical decision tree (FOLDT) (Blockeel and De Raedt 1998). FOLDT is an adaptation of a decision tree for first order logic, where the tests in the nodes are conjunctions of first order literals. The ND-TILDE algorithm is shown in Algorithm 1. ND-TILDE creates a FOLDT based on the set of training examples E. The tests candidates are created (step 2 of the algorithm) from a set of refinement operators (Blockeel and De Raedt 1998) that are previously defined by an expert. Each refinement operator generates a set of first order literals as candidate tests for division of the set of examples E. The best test to divide the set E is the test that reduces the entropy. The optimal test is chosen (step 3) using the default gain ratio (Quinlan 1993). If the partition induced on E indicates that the division should stop (procedure STOP_CRIT in step 5), a leaf is created. The tree leaves created (step 6) contain atomic sentences that represent abstract actions. If more than one atomic sentence represent the remaining examples in E, TILDE chooses the atomic sentence that represents the largest number of examples, while ND-TILDE generates nondeterministic policies; that is, ND-TILDE associates with a tree leaf the set of all abstract actions indicated by the examples in E for the abstract state corresponding to the respective leaf. If the partition induced on E does not indicate that the division process should stop, for each one of the partitions induced (step 8) the function ND-TILDE is recursively called (step 9). An internal node is created using the optimal test τ as test, and the two children are the sub-trees induced by each call of ND-TILDE (step 9).

Algorithm 1 Algorithm ND-TILDE.

```
1: function ND-TILDE (E: set of examples): returns a decision tree
2:    T = GENERATE_TESTS_FOL(E)
3:    τ = OPTIMAL_SPLIT_FOL(T,E)
4:    ϵ = partition induced on E by τ
5:    if STOP_CRIT(E,ϵ) then
6:       return leaf(INFO_FOL(E))
7:    else
8:       for all E_j in ϵ do
9:          t_j = ND-TILDE(E_j)
10:      end for
11:      return inode(τ, (j, t_j))
12:   end if
13: end function
```

3.2 Case study: Abstract policies in a robotic navigation problem

We now introduce our main testbed, to be further explored in our experiments (Section 5). Consider a *concrete* robotic navigation problem, where the environment is shown in Figure 2.

Suppose first the concrete specification of the problem employs grounded predicates from the following vocabulary: $P = \{$`isCorridor/1`, `isRoom/1`, `isCenter/1`, `isNearDoor/1`, `in/1`, `isConnected/2`$\}$ and $A = \{$`gotoRDRD/2`, `gotoCDCD/2`, `gotoCCCC/2`, `gotoRCRD/2`, `gotoRDRC/2`, `gotoRDCD/2`, `gotoCDRD/2`, `gotoCCCD/2`, `gotoCDCC/2`$\}$. The meaning of each predicate should be clear; for instance, if the agent is occupying location 1 in the map (represented as object `l1`), we can describe the state as $\sigma_{l1} = \{$`in(l1)`, `isCorridor(l1)`, `isCenter(l1)`, `isConnected(l1,l5)`, `isCorridor(l5)`, `isNearDoor(l5)` $\}$. If the agent makes a choice to go from location `l1` that is `isCorridor(l1)` and `isCenter(l1)`, to location `l5` that is `isCorridor(l5)` and `isNearDoor(l5)`, we write this action by `gotoCCCD(l1,l5)`. The abbreviations `RC`, `CD` indicate respectively (`isRoom/1`, `isCenter/1`), (`isCorridor/1`, `isNearDoor/1`), and so on. The precondition of `gotoXXYY(Li,Lj)` is (`in(Li)` $\wedge$ `isConnected(Li,Lj)`) and both locations `Li` and `Lj` attend the required conditions `XX` and `YY`.

Suppose the goal is `l2`; that is, the reward function indicates that reaching `l2` from any possible initial position is the only valuable purpose for the agent.

Suppose also that we take this concrete MDP and we find the optimal policy for it. We now run ND-TILDE from examples generated by the optimal policy (that is, the optimal policy for navigation from any location to `l2`). Figure 3 shows the abstract policy induced using ND-TILDE from examples generated by the optimal policy for the robot navigating from any location `Li` to `l2` in the environment in Figure 2.

3.3 Grounding abstract policies

An abstract policy $\hat{\pi}_a$ can induce a policy π_a in a given concrete MDP. We propose to do so as follows. Given a state σ, we find the first[1] $\hat{\sigma}$ such that $\sigma \preceq_\theta \hat{\sigma}$. We then have a set of abstract actions $\hat{\alpha}_i$. Note that an abstract action may be

[1]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.

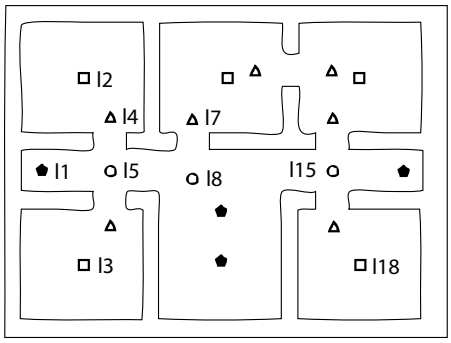

Figure 2: Relational representation for the problem used to induce the abstract policy. Squares denote centers of rooms, e. g. (`isRoom(l2)` $\wedge$ `isCenter(l2)`), triangles denote doors in rooms, e. g. (`isRoom(l3)` $\wedge$ `isNearDoor(l3)`), circles denote doors in corridors, e. g. (`isCorridor(l5)` $\wedge$ `isNearDoor(l5)`), and black pentagons denote centers of corridors, e. g. (`isCorridor(l1)` $\wedge$ `isCenter(l1)`).

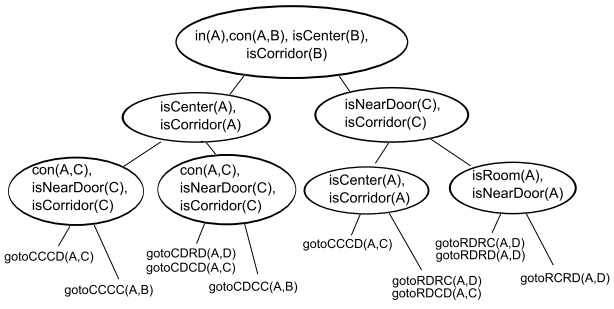

Figure 3: Abstract policy induced by the ND-TILDE algorithm for the navigation problem in the environment depicted in Figure 2; goal is to reach `l2` from any location. The conjunction of predicates in a path from the root to the leaf defines an abstract state. Here `con/2` stands for `isConnected/2`. Tree leaves define abstract actions.

mapped into a set of concrete actions for the underlying concrete decision problem. Therefore, we have a policy where a state is mapped into a set of actions. To produce a particular concrete sequence of actions, we select randomly (with uniform probability) an abstract action from the set of possible abstract actions, and again we select randomly (with uniform probability) a concrete action from the set of concrete actions associated with the selected abstract action. Obviously, other schemes may be used to produce a sequence of concrete actions.

It is important to note that, depending on how the abstract policy was obtained, there might be no $\hat{\sigma}$ in the list for a given σ. In this case we could select randomly from $\mathcal{A}_\sigma$.

As an example, consider again the robot navigation problem described in Section 3.2 and the corresponding abstract policy $\hat{\pi}_a$ (shown in Figure 3) for the goal of reaching `l2`. If the robot were in state σ_{l1} (see Section 3.2) and we applied the abstract policy of Figure 3, we would have:

- θ = {`A/l1,B/l5`}, and (`in(l1)`, `isConnected(l1,l5)`, `isCenter(l5)`, `isCorridor(l5)`) is FALSE.
- θ = {`C/l5`}, and (`isNearDoor(l5)`, `isCorridor(l5)`) is TRUE.
- θ = {`A/l1`}, and (`isCenter(l1)`, `isCorridor(l1)`) is TRUE.
- $\hat{\pi}_a$ = {`gotoCCCD(A,C)`}, and $\hat{\alpha}$ = `gotoCCCD(A,C)` = {`gotoCCCD(l1,l5)`}, resulting in the indication of $\alpha =$ `gotoCCCD(l1,l5)` to be applied in σ_{l1}.

We now explain how the abstract policy can be combined with a learning algorithm to speed up the learning process.

4 Simultaneous Reinforcement Learning

Recall that we have a given concrete decision problem, that we henceforth refer to as the *source problem*, for which we have a policy, and we have generated an abstract policy from the latter. We now have another decision problem, the *target* problem, for which we can instantiate the abstract policy. The hope is that by using the abstract policy, we obtain a reasonable policy for the target problem. However, we have no guarantees. Therefore we must refine this initial policy through exploration: we must apply reinforcement learning to the concrete target problem. From the point of view of reinforcement learning, the appeal of this scheme is that we are speeding up convergence, as the abstract policy provides a sensible start for the process. Later we report on experiments that do show this to happen in practice.

However, just applying reinforcement learning to the concrete target level is *not* the best possible use of the available exploration data. One of the challenges of the abstraction process is to capture whatever is "useful" from the source problem; surely the target problem may be such that recommendations by the abstract policy are incorrect for some particular states.

The idea then is to use reinforcement learning to refine the abstract policy as well. Because the abstract policy is simpler (less states, less actions) than the concrete target policy, convergence to optimality can be expected to be faster in the abstract level than in the concrete level. We should 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. We do this in this section.

Hence the abstract level works as the generous mother who, having learned a few lessons with the first baby, now applies her knowledge to the second baby to the extent that it is possible, quickly learning from her new mistakes, until this second baby grows to optimality and leaves the nest.

We now turn these intuitions into reality, and in the next section we show that the framework works in practice.

We start with some additional notation. Because 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 $\sigma_g \in \mathcal{G}$. Define

$$\begin{aligned}\mathcal{T}(\pi, \langle \sigma_0, \sigma_g \rangle) \quad = \quad & \{s_1 s_2 \ldots s_n \text{ for some } n | \\ & s_1 = \sigma_0, s_n = \sigma_g, \\ & T(s_i, a_i, s_{i+1}) \neq 0 \text{ for some } a_i \\ & \quad \text{such that } \pi(s_i) \neq \emptyset\},\end{aligned}$$

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.

For instance, with $\mathcal{P}$ and π^* defined for the example of Figure 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 set of tasks $\{\langle \sigma_0, \sigma_g \rangle\}$ such that the goal $\sigma_g \in \mathcal{G}$ is fixed and $\sigma_0 \in \mathcal{S}$; π_r is a random policy that at each state σ chooses an action with uniform distribution over the set $\mathcal{A}_\sigma$; π_a is a (deterministic) policy induced in the ground MDP by a (non-deterministic) abstract policy $\hat{\pi}_a$. Also define π_Q to be the (deterministic, concrete) policy that uses the estimates in the Q function to choose an action according to $\pi_Q(\sigma) = \textit{arg max}_\alpha \, Q(\sigma, \alpha)$.

The agent's interaction with the environment clearly occurs in the concrete level, however the experience $\langle \sigma_t, \alpha_t, \sigma_{t+1}, r_{t+1} \rangle$ is used in both, abstract and concrete, so that values can be updated in both levels. We propose that both learning processes, abstract and concrete, follow an ϵ-greedy exploration/exploitation strategy.

In the abstract level we have:

$$\hat{\pi}_a(\hat{\sigma}) = \begin{cases} \hat{\pi}_{a_s}(\hat{\sigma}) & \text{with probability } \epsilon_a, \\ \hat{\pi}_{a_t}(\hat{\sigma}) & \text{with probability } 1 - \epsilon_a, \end{cases} \quad (1)$$

where: $\hat{\pi}_{a_s}$ is an abstract policy obtained using the concrete source policy, $\hat{\pi}_{a_t} = \textit{arg max}_\alpha \, Q(\hat{\sigma}, \hat{\alpha})$ is a policy learned in the abstract level, and $0 \leq \epsilon_a \leq 1$ is a parameter that defines the exploration/exploitation tradeoff in the abstract level of our framework.

In the concrete level we have:

$$\pi_c(\sigma) = \begin{cases} \pi_a(\sigma) & \text{with probability } \epsilon_1(1 - \epsilon_2), \\ \pi_r(\sigma) & \text{with probability } \epsilon_1 \epsilon_2, \\ \pi_Q(\sigma) & \text{with probability } 1 - \epsilon_1, \end{cases} \quad (2)$$

where: π_c is the deterministic policy applied by the agent in state σ; π_a is a deterministic policy induced from $\hat{\pi}_a(\hat{\sigma})$ with $\sigma \preceq_\theta \hat{\sigma}$, $0 \leq \epsilon_1 \leq 1$, $0 \leq \epsilon_2 \leq 1$ are parameters that define the exploration/exploitation tradeoff in the concrete level of our framework; and π_r is a random policy where an action $a \in \mathcal{A}_\sigma$ is chosen randomly.

That is: in the concrete level, learn by using policy π_a and gradually replace it with π_Q, while using π_r less often, just as to guarantee exploration and convergence to optimality. In the abstract level, learn by using policy $\hat{\pi}_{a_s}$ and gradually replace it with $\hat{\pi}_{a_t}$ as the learning process evolves.

When the agent starts learning, π_Q is equal to π_r, since the Q function is usually initialized with the same value to all state-action pairs. In this case for some task $\langle \sigma_0, \sigma_g \rangle$ the agent 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 the agent will consider have the property that they used to be 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.

As to why the agent needs to gradually change to π_Q note that we may not 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. With this, the standard proofs for convergence of reinforcement can be adapted to prove convergence to optimality, because there is always a guarantee of full exploration (using π_r).

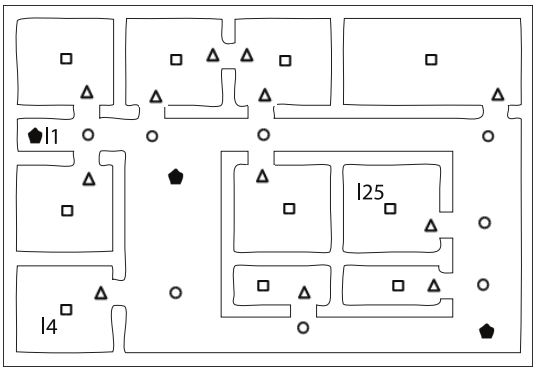

Figure 4: Target problem: initial locations are: `l1` and `l4`; goal location is `l25`.

5 Experiments

We have run a set of experiments focused on the robotics navigation domain discussed in Section 3.2.

Initially we considered a source problem where the robot navigated from any location `Li` to the goal location `l2` in the environment shown in Figure 2. The policy that solved this problem was abstracted into the abstract source policy $\hat{\pi}_{a_s}$ depicted in Figure 3.

Experiments focused on a target problem in which the robot navigates in the environment shown in Figure 4. Two tasks are executed: $task_1 = \langle \sigma_1, \sigma_{25} \rangle$ and $task_2 = \langle \sigma_4, \sigma_{25} \rangle$. In the first task, $task_1$, the abstract source policy can guide the robot to the goal, albeit in a sub-optimal manner; in the second task, $task_2$, the abstract source policy cannot guide the robot to the goal. We defined the reward function as follows: a reward of 1 is granted when the robot gets to the goal; otherwise a reward of 0 is granted.

Tasks were repeatedly solved when performing reinforcement learning. For each task, at most 4000 steps were run, where a step is a transition from one state to another. A sequence of steps that takes the robot from an initial location to the goal is an episode. Every time the robot reaches the goal, it is placed again at the initial location; that is, the initial state of each task.

Table 1: Abstract states defined for abstract level.

in(A),con(A,B),con(A,C),con(A,D),con(A,E), isCorridor(A),isNearDoor(A),isRoom(B), isNearDoor(B),isRoom(C),isNearDoor(C), isCorridor(D),isCenter(C),isCorridor(E), isNearDoor(E)
in(A),con(A,B),con(A,C),con(A,D), isCorridor(A),isNearDoor(A),isRoom(B), isNearDoor(B), isCorridor(C), isNearDoor(C),isCorridor(D),isCenter(C)
in(A),con(A,B),con(A,C),con(A,D), isCorridor(A),isNearDoor(A),isCorridor(B), isNearDoor(B),isCorridor(C),isNearDoor(C), isRoom(D),isNearDoor(D)
in(A),con(A,B),con(A,C),con(A,D), isRoom(A),isNearDoor(A),isRoom(B), isNearDoor(B),isCorridor(C),isNearDoor(C), isRoom(D),isCenter(D)
in(A),con(A,B),con(A,C),con(A,D), isCorridor(A),isCenter(A),isCorridor(B), isNearDoor(B),isCorridor(C),isNearDoor(C), isCorridor(D),isNearDoor(D)
in(A),con(A,B),con(A,C),con(A,D), isRoom(A),isNearDoor(A),isRoom(B), isNearDoor(B),isRoom(C),isNearDoor(C), isRoom(D),isCenter(D)
in(A),con(A,B),con(A,C),isRoom(A), isNearDoor(A),isRoom(B),isNearDoor(B)), isRoom(C),isCenter(C)
in(A),con(A,B),con(A,C),isRoom(A), isNearDoor(A),isRoom(B),isCenter(B), isCorridor(C),isNearDoor(C)
in(A),con(A,B),con(A,C),isRoom(A), isCenter(A),isRoom(B),isNearDoor(B), isRoom(C),isNearDoor(C)
in(A),con(A,B),con(A,C),isCorridor(A), isCenter(A),isCorridor(B),isNearDoor(B), isCorridor(C),isNearDoor(C)
in(A),con(A,B),isCorridor(A),isCenter(A), isCorridor(B),isNearDoor(B)
in(A),con(A,B),isRoom(A),isCenter(A), isRoom(B),isNearDoor(B)

For each task, three approaches were compared. The first one, denoted by ALET, is a straightforward Q-learning algorithm with ϵ-greedy where the exploration policy is random; that is, $\pi^1_Q(\sigma)$ is followed with probability $1-\epsilon_1$ while $\pi^1_r(\sigma)$ with probability ϵ_1. The second one, denoted by ND, is a Q-learning algorithm with ϵ-greedy where the exploration policy is produced by switching between the random policy π^2_r and the policy induced by the abstract source policy π^2_a:

$$\pi^2_c(\sigma) = \begin{cases} \pi^2_a(\sigma) & \text{with probability } \epsilon_1(1-\epsilon_2), \\ \pi^2_r(\sigma) & \text{with probability } \epsilon_1\epsilon_2, \\ \pi^2_Q(\sigma) & \text{with probability } 1-\epsilon_1. \end{cases}$$

The third approach, denoted by QSA, follows our proposal (Equations (1) and (2)). While ALET and ND run Q-learning only at the concrete level, we emphasize that QSA uses our machinery to obtain reinforcement learning both at the concrete and the abstract levels.

The abstracted problem is encoded by the states indicated in Table 1. Note the reduced size of this state space (the cardinality is just 12). We used $\epsilon_1 = 0.5$, $\epsilon_2 = 0.3$, and $\epsilon_a = 0.7$.

Results for $task_1$ and $task_2$ are summarized respectively by Figures 5 and 6. Each task was repeated 40 times. At each 250 steps in the learning process, the current policy was evaluated in a validation phase. 30 episodes were executed in each evaluation using a greedy policy. Therefore, each point in these graphics is an average of 1200 episodes.

In $task_1$ the ND approach already attains the best abstract policy possible, while the QSA approach gets to this abstract policy through the learning process. Both approaches have better performance than ALET in $task_1$. QSA clearly wins over ALET and ND in $task_2$, due to the combined effect of abstract and concrete reinforcement learning.

Table 2 shows the average number of learning steps (and the standard deviation) to get to the optimal policy for each task.

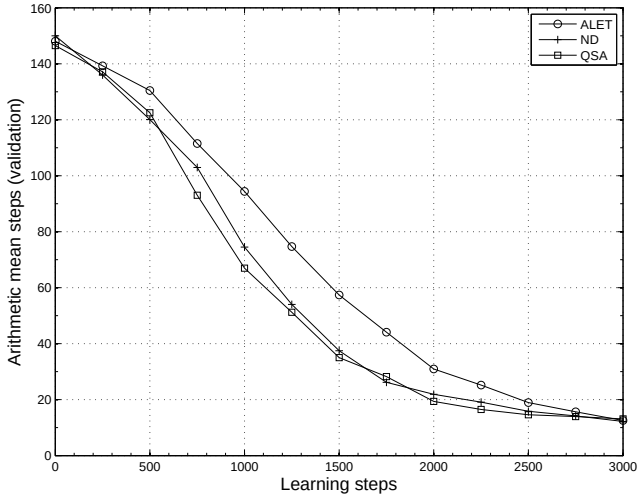

Figure 5: Average steps to reach the goal. `l1` is the robot initial location and the goal is to reach the location `l25`.

Table 2: The mean number of learning steps to attain the optimal concrete policy.

	$task_1$	$task_2$
ALET	1872 ± 762	1426 ± 590
ND	1548 ± 654	1220 ± 553
QSA	1597 ± 453	946 ± 413

6 Conclusions

In this paper we have proposed a framework for simultaneous reinforcement learning over abstract and concrete levels of a target decision problem, where the abstract level

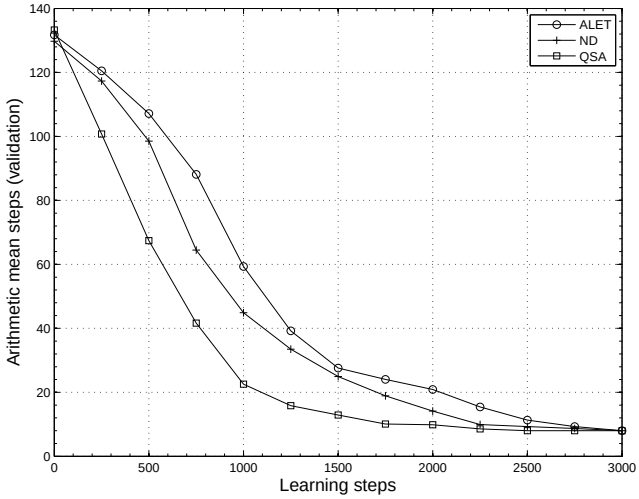

Figure 6: Average steps to reach the goal. `l4` is the robot initial location and the goal is to reach the location `l25`.

is inherited from a previously solved source decision problem. We have contributed with a variant of the TILDE algorithm, ND-TILDE, for generation of nondeterministic abstract policies. But more importantly, we have contributed with a scheme for simultaneous update of abstract and concrete policies that takes into account prior knowledge, exploration data, and that guarantees convergence to optimality.

These contributions are valuable to the theory of abstraction, as they show how abstraction can be useful in practice — indeed our experiments confirm that our framework has better performance than competing approaches. The contributions are also valuable to the theory of reinforcement learning, as they show how to speed up learning using previously acquired policies and abstraction.

A variety of further results can be pursued. We are particularly interested in examining schedules for gradual reduction of ϵ in learning (so that exploration data eventually takes over completely), and in a larger set of practical problems that show the limits of policy transfer.

Acknowledgments

This research was partially supported by FAPESP (08/03995-5, 09/04489-9, 09/14650-1, 10/02379-9) and by CNPq (475690/2008-7, 305395/2010-6, 305512/2008-0).

References

Barto, A. G., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4):341–379.

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. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2):285–297.

Burkov, A., and Chaib-draa, B. 2007. Adaptive play q-learning with initial heuristic approximation. In *Int. Conf. on Robotics and Automation*, 1749–1754.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proc. of 25th Int. Conf. on Machine Learning*.

do Lago Pereira, S.; de Barros, L.; and Cozman, F. 2008. Strong probabilistic planning. In *Mexican Int. Conf. on Artificial Intelligence*, volume 5317 of *LNCS*. 636–652.

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 *Proc. of 21th Int. Conf. on Machine Learning*, 465–472.

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*.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for mdps. In *Proc. of 9th Int. Symposium on Artificial Intelligence and Mathematics*, 531–539.

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

Quinlan, J. R. 1993. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann.

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

Uther, W. T. B., and Veloso, M. M. 2002. Ttree: Tree-based state generalization with temporally abstract actions. In *Proc. of Symposium on Abstraction, Reformulation, and Approximation*, 260–290.

van Otterlo, M. 2004. Reinforcement learning for relational MDPs. In *Proc. of Machine Learning Conf. of Belgium and the Netherlands*, 138–145.