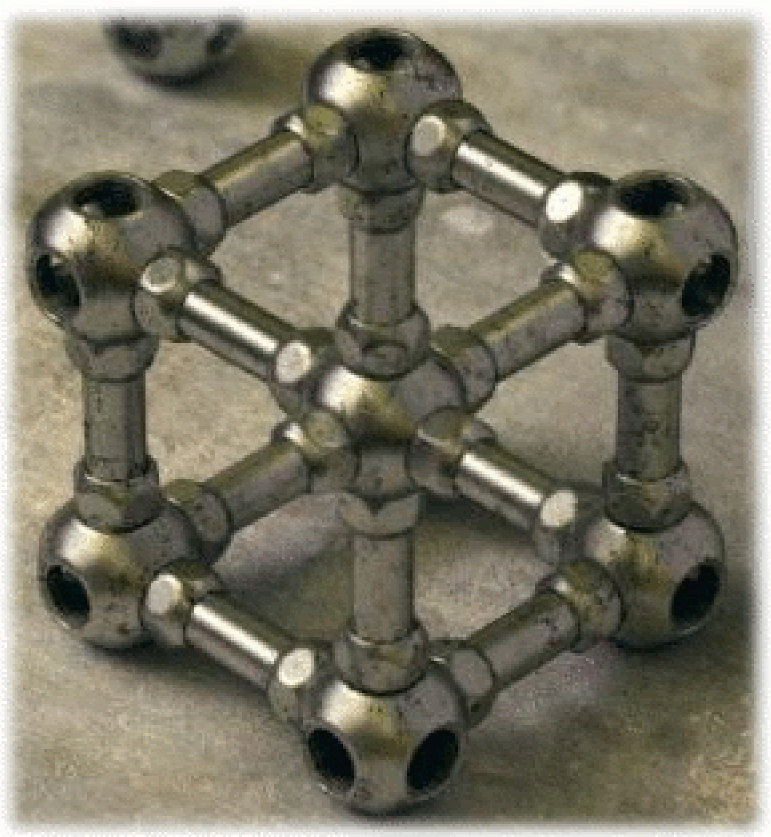


Stochastic Abstract Policies for Knowledge Transfer in Robotic Navigation Tasks [1]

Author(s):

Tiago Matos
Yannick Plaino Bergamo
Valdinei Freire da Silva
Anna Helena Reali Costa

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

Stochastic Abstract Policies for Knowledge Transfer in Robotic Navigation Tasks

Tiago Matos, Yannick Plaino Bergamo, Valdinei Freire da Silva, and Anna Helena Reali Costa

Laboratório de Técnicas Inteligentes (LTI/EPUSP)
Escola Politécnica, Universidade de São Paulo
São Paulo, SP, Brazil
{tiago.matos,yannick}@usp.br, valdinei.freire@gmail.com, anna.reali@poli.usp.br

Abstract. Most work in navigation approaches for mobile robots does not take into account existing solutions to similar problems when learning a policy to solve a new problem, and consequently solves the current navigation problem from scratch. In this article we investigate a knowledge transfer technique that enables the use of a previously know policy from one or more related source tasks in a new task. Here we represent the knowledge learned as a stochastic abstract policy, which can be induced from a training set given by a set of navigation examples of state-action sequences executed successfully by a robot to achieve a specific goal in a given environment. We propose both a probabilistic and a nondeterministic abstract policy, in order to preserve the occurrence of all actions identified in the inductive process. Experiments carried out attest to the effectiveness and efficiency of our proposal.

1 Introduction

Automated Planning and Learning are essential components of intelligent behavior. Broadly speaking, they deal with the methods by which an intelligent agent can determine the policy (action strategy) needed to successfully achieve a goal. However, often the determination of a policy for solving a certain task has to be done without any use of the acquired knowledge to solve similar tasks. Intuitively, generalization from closely related, but solved, problems may often produce policies that make good decisions in many states of a new unsolved problem. Knowledge transfer explores this intuition and could then be used in these situations. The core idea of knowledge transfer is that experience gained in performing one task can help solve a related, but different, task.

In this paper, we are concerned with the problem of transfer the knowledge that was learned in one problem to another problem so that this knowledge gives effective guidance when solving the new problem. Here we represent the knowledge learned as a stochastic abstract policy, which can be learned from a training set given by a set of state-action pairs obtained by solving one or more instances of a given problem. Such policies are not optimal but they are general and they can be applied effectively, producing good results.

More precisely, our approach starts from a set of navigation examples of state-action sequences executed successfully by a robot to achieve a specific goal in a given environment. These examples can be achieved through the execution of a plan developed to solve this task or provided by a teacher. Then, we induce a stochastic navigation policy in the form of a relational decision tree from these examples. This relational decision tree represents an abstract policy, which expresses preferences over navigation actions and can be used by the robot to decide the navigation actions both in the original task and in similar tasks or domains.

Our contribution lies in an extension of the work of Kersting et al. [1], so as not to consider in the leaf of the relational decision tree only the abstract action more frequently found in the inductive process, but to use a stochastic abstract action. We propose both a probabilistic and a nondeterministic abstract policy in order to preserve the occurrence of all ground actions identified in the inductive process.

Several papers have been looking at ways to abstract the knowledge acquired in solving a problem and use it to solve other problems. Taylor and Stone [2] provide a thorough survey on the transfer of learning using propositional descriptions of the states. In several works the source task is learned in some ground state space where an abstract policy is generated, and this policy is applied to the target task in order to improve learning performance, especially in the early episodes. Based on samples experienced by the agent when acting optimally in the source task, in [3] propositional symbolic learners generalize the optimal policy. In [4] action relevance is used in order to reduce the action set to be explored in the target task. In [5] a heuristic policy is used as a knowledge to be applied in the policy learning process in a new task or domain.

Regarding the first order state description, few examples are found in the literature for the domain of spatial navigation. In order to fully describe the relational situation in which an agent is, one must map the environment completely. So as not to do so, [1] and [6] rely on relational local perception and structured environments, where (near-) optimal abstract policies can be defined. Here we propose two different approaches for transfer knowledge from one problem to another, and this knowledge is represented by abstract policies that are based on local relational perceptions made in structured environments.

The remainder of this paper is structured as follows. In Section 2 we give an overview of Relational Markov Decision Processes and show how to learn and use abstract navigation policies. Section 3 covers a detailed specification of our approach, describing how to learn and to use stochastic abstract policies. We present experimental results and analysis in Section 4 and discuss the implications of these. Finally, in Section 5 we discuss how this technique can be expanded in the future, and draw conclusions based on what we have outlined.

2 Learning Relational Navigation Policies

Several domains are naturally described in terms of objects and relations among objects, therefore they can be compactly described with relational representations.

On the other hand, stochastic domains, such as robotic navigation domains that concern us here, are properly modeled by Markov Decision Processes. Relational Markov Decision Processes (RMDPs) combine relational logic with MDPs to powerfully exploit relational structures. An RMDP describes a mathematical model of interactions between an agent and a stochastic, dynamic environment when the world is described by objects and relations among them. The advantage of the relational representation is abstraction.

We give here a very brief review of the key aspects of RMDPs and their solutions. For a thorough treatment, we refer the reader to [7,8].

2.1 Relational Markov Decision Process

The key idea in Relational Markov Decision Processes (RMDPs) [8] is to merge relational logic with MDP. Let $\mathcal{P} = \{p_1/\alpha_1, \ldots, p_n/\alpha_n\}$ be a set of predicates p_i with their arities α_i, $\mathcal{C} = \{c_1, \ldots, c_k\}$ a set of constants, and $\mathcal{A}^{'} = \{a_1/\alpha_1, \ldots, a_m/\alpha_m\}$ a set of actions a_i with their arities α_i where $a_i = H_{a_i} \xleftarrow{p_{a_i}:A_{a_i}} B_{a_i}$, and A_{a_i} is an atom representing the name and the arguments of the action a_i, B_{a_i} is the precondition, H_{a_i} is a state representing the successful outcome, and p_{a_i} is the probability that an outcome succeed. An atom is called ground if it contains no variables. Let $\mathcal{S}^{'}$ be the set of all conjunctions of ground atoms over $\mathcal{P}$ and $\mathcal{C}$, and $\mathcal{A}$ the set of all ground atoms over $\mathcal{A}^{'}$ and $\mathcal{C}$. An RMDP is a tuple $M = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is a subset of $\mathcal{S}^{'}$, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a probabilistic transition function, and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a reward function. For a given state $s \in \mathcal{S}$ there is a corresponding set of feasible actions $\mathcal{A}(s) \subseteq \mathcal{A}$ which is calculate through the action precondition.

The task of the agent is to learn a policy $\pi : \mathcal{S} \times \mathcal{A}$ for selecting its next action $a \in \mathcal{A}(s)$ to be performed on the current state $s \in \mathcal{S}$, i. e., $\pi(s) = a$. The learned policy should maximize $V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t | \pi, s_0 = s]$, where $\gamma \in [0, 1[$ is a discount factor and r_t is the reward received in time t, i. e., the policy π^* is optimal if $V^{\pi^*}(s) \geq V^{\pi^{'}}(s)$, $\forall s \in S$ and $\forall \pi^{'}$.

If the model of the problem is fully known, we can use planning techniques such as SPUDD [9] to determine π^*, otherwise we can use reinforcement learning techniques [10] for this purpose.

2.2 Modeling a Robotic Navigation Problem as an RMDP

A robotic navigation problem can elegantly be represented using an RMDP. Consider[1] the simple environment shown in Figure 3(a): we define a set of $n = 19$ constant locations $\mathcal{C} = \{$`l1`, `l2`, $\cdots$, `l19`$\}$. Unary predicates indicate the location type such as `isCorridor(Li)`, `isRoom(Li)`, `isCenter(Li)`, `isNearDoor(Li)`,

[1] Squares symbolize centers of rooms, e. g. ($isRoom(l2) \wedge isCenter(l2)$), triangles symbolize doors of rooms, e. g. ($isRoom(l3) \wedge isNearDoor(l3)$), circles symbolize doors of corridors, e. g. ($isCorridor(l5) \wedge isNearDoor(l5)$), and black pentagons symbolize corridors, but not in front of a door – here, for simplicity, we call center, e. g. ($isCorridor(l1) \wedge isCenter(l1)$).

whereas a binary predicate `isConnected(Li,Lj)` indicates that `Li` is connected to `Lj`. A special unary predicate `in(Li)` indicates the location of the robot. Terms starting in lowercase represent constants, and in uppercase are variables.

We consider a set $\mathcal{A}^{'}$ of specialized navigation actions that move the robot from one location to another, e. g., `gotoRCRD(Li,Lj)` moves the robot from `Li` in the center of a room to a location `Lj` near a door in a room, `gotoRDCD(Li,Lj)` moves the robot from a location `Li` near a door in a room to a location `Lj` near a door in a corridor, etc. The generic action `gotoXXYY(Li,Lj)` shortens the conditions required for the current location `XX` and the next location `YY`, e. g., $\texttt{XX} = \texttt{RC}$ when $(\texttt{in(Li)} \wedge \texttt{isRoom(Li)} \wedge \texttt{isCenter(Li)})$, $\texttt{XX} = \texttt{CD}$ when $(\texttt{in(Li)} \wedge \texttt{isCorridor(Li)} \wedge \texttt{isNearDoor(Li)})$ and so on; likewise, $\texttt{YY} = \texttt{CC}$ when $(\texttt{isCorridor(Li)} \wedge \texttt{isCenter(Li)})$, and so on.

The precondition of `gotoXXYY(Li,Lj)` is $(\texttt{in(Li)} \wedge \texttt{isConnected(Li,Lj)})$ and both locations `Li` and `Lj` attend the required conditions `XX` and `YY`.

If the result of an action is successful the robot reaches the desired location $(\texttt{in(Lj)})$ with probability p, and if the action fails the robot remains in the same location $(\texttt{in(Li)})$ with probability $1 - p$.

The use of RMDPs offers many possibilities for abstraction. An abstract state X is defined by the conjunction of predicates that have at least one term as a variable. The ground state $s \in S$ is an instance of X if there is a substitution $\theta = \{\texttt{V}_1/\texttt{t}_1, \ldots, \texttt{V}_k/\texttt{t}_k\}$ such that $X\theta = s$, where V_i is a variable and t_i is a term. For instance, given the abstract state $X = (\texttt{in(L)} \wedge \texttt{isRoom(L)} \wedge \texttt{isNearDoor(L)})$ and the substitution $\theta = \{\texttt{L/l1}\}$, then the state $s = X\theta = (\texttt{in(l1)} \wedge \texttt{isRoom(l1)} \wedge \texttt{isNearDoor(l1)})$ is an instance of X. Abstract actions can be described similarly by introducing variables in their arguments, preconditions, and postconditions.

2.3 Generating an Abstract Policy

An abstract policy is a mapping from abstract states to abstract actions. An abstract policy captures a generalization of optimal (or sub-optimal) policies according to the structural characteristics of the problem. Abstract policies may be induced from a set of examples E. Here, examples are formed by state-action pairs taken from a set of navigation experiences generated by applying an optimal policy. Given a goal state in the robot navigation problem, we find an optimal policy in the ground MDP. Then we define a set of initial states from which the robot acts optimally to generate state-action sequences that lead to the goal state [1]. Each state-action pair of these sequences is inserted in E. The task then is to induce an abstract policy based on E.

TILDE is an inductive algorithm that makes abstraction of the experiences, and represents the abstract policy induced in a first order logical decision tree (FOLDT) [11]. 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 TILDE algorithm [11] is shown in Algorithm 1. TILDE creates a FOLDT based on the set of training examples E. Test candidates are created (step 2 of the algorithm) from a set of refinement operators [11], which are previously defined by an expert.

Each refinement operator generate a set of first order literals as test candidate for the division of the set of examples E. The best test to divide the set E is the test that reduces a measure of entropy. The optimal test is chosen (step 3) using the default gain ratio [12]. 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 represents the remaining examples in E, TILDE chooses the atomic sentence that represents the largest number of examples. If the partition induced on E does not indicate that the division should stop, for each one of the partitions induced (step 8) the function 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 TILDE (in step 9). Figure 1 shows the FOLDT induced from the examples generated by applying the optimal policy for the task of the robot going from any location L to location l2 in the environment shown in Figure 3(a).

Algorithm 1. Algorithm TILDE

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

2.4 Using the Abstract Policy to Guide a Robot

The FOLDT induced by TILDE represents an abstract policy to the robot navigation problem. Each node of the FOLDT captures a logical test, which either succeeds or fails when applied to a particular (ground) state. If it succeeds, we use the left subtree; otherwise the right one. Leaves in the FOLDT induced by TILDE represent a single abstract action to be taken in the abstract state, which is defined by the conjunction of all preceding tests from the root to the leaf. Note that each abstract action can represent a set of ground actions.

We perform a forward search guided by the learned abstract navigation policy. The robot starts in some location, observes the current (ground) state, and decides the (ground) action to perform next by evaluating the abstract policy on the observed state. The observed ground state imposes a substitution θ to the abstract state and to the abstract action suggested by the FOLDT. In order to

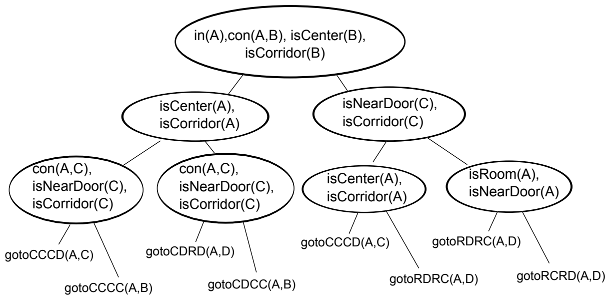

in(A),con(A,B), isCenter(B), isCorridor(B)
isCenter(A), isCorridor(A)
isNearDoor(C), isCorridor(C)
con(A,C), isNearDoor(C), isCorridor(C)
con(A,C), isNearDoor(C), isCorridor(C)
isCenter(A), isCorridor(A)
isRoom(A), isNearDoor(A)
gotoCCCD(A,C)
gotoCCCC(A,B)
gotoCDRD(A,D)
gotoCDCC(A,B)
gotoCCCD(A,C)
gotoRDRC(A,D)
gotoRDRC(A,D)
gotoRCRD(A,D)

Fig. 1. An abstract policy induced by the TILDE algorithm for the navigation problem in the environment shown in Figure 3(a). The conjunction of predicates in a path from the root to the leaf defines an abstract state. Tree leaves define abstract actions.

decide the ground action to be performed next by the robot, we choose uniformly among all ground actions defined by the substitution θ to the abstract action suggested in the FOLDT leaf.

However, it may happen that the substitution θ to the chosen abstract action generates a set of ground actions $A\theta$ whose elements do not belong to the set of executable actions in the observed ground state, i. e., $A\theta \cap A(s) = \emptyset$. In this case we choose uniformly among all ground actions in the set $A(s)$. Finally, having decided what ground action to perform, the robot performs the action and repeats the observation-decision-actuation cycle.

Note that TILDE creates a tree that contains only one abstract action in each leaf. Unfortunately, important information is lost in this abstraction process. Therefore, the application of an abstract policy may cause some problems. In the next section we discuss these issues and propose a modification in the TILDE algorithm to overcome these problems.

3 Generating a Stochastic Abstract Policy

The process of generalization causes loss of some important information to solve the problem, since it groups together different actions to be applied in the same abstract state. For example, in the environment of the Figure 3(a) where the goal is to reach the location `l2`, the abstract state that groups together ground states defined by locations `l5` and `l8` will leads to the same tree leaf (represented by the third leaf from left to right in Figure 1). However, the training examples given when the robot was at location `l5` indicated that the optimal ground action was `gotoCDRD(l5,l3)`, and when the robot was at location `l8` the optimal ground action was `gotoCDCD(l8,l5)`. In this case, during the creation of the leaf, TILDE will choose the abstract action that represents the largest number of examples, and the distinction between these situations will be lost.

Also related to this same problem, if the abstract policy generated in this way is used to directly control the robot navigation, there may be cases in which the goal is never reached. To illustrate these cases, consider again the environment shown in Figure 3(a) and the abstract policy shown in Figure 1: this abstract policy will never drive the robot to the goal location `l2` whenever he comes to the location `l8`, since the abstract policy will guide the robot to the location `l7` – the abstract policy gives action `gotoCDRD(l8,l7)` – rather than to the location `l5` (which leads to the goal).

Due to these facts we propose a modification to the abstraction algorithm TILDE that can handle the problems showed previously. The modification (named X-TILDE) enables the construction of a tree that represents a stochastic abstract policy. Each leaf now contains a set of abstract actions. We propose two versions of the X-TILDE algorithm: P-TILDE (Probabilistic TILDE) in which we associate a probability of choosing each abstract action belonging to a leaf, and ND-TILDE (Nondeterministic TILDE) in which there is no preference in the choice of an abstract action in a leaf. Kersting et al. [1] also noted some of these problems that we raised, and in order to circumvent them, they proposed the use of an auxiliary structure and some heuristics in the process of using the policy.

In the leaf level, the TILDE algorithm can face two situations: i) the actions of all the training examples corresponding to the same abstract state indicate the same abstract action; or ii) the actions of some of them suggest another abstract actions. In the first case, the choice is trivial, while in the latter case, the algorithm TILDE chooses to associate to the leaf the abstract action whose corresponding ground action occurred more often in the examples.

We propose changing TILDE so that less information is lost. Instead of assigning only one abstract action to each tree leaf (step 6 of the Algorithm 1), our proposal is to associate a set of abstract actions to each leaf (see Figure 2). We propose two variations of versions, called synthetically X-TILDE: P-TILDE for when X is P, and ND-TILDE in the other case. P-TILDE keeps the abstract actions and their respective frequency of occurrence in the tree leaf, and we make use of it whenever we reuse the abstract policy. On the other hand, the ND-TILDE keeps in the leaf all the abstract actions derived from the training examples, and they have an equal chance of being chosen when we reuse the abstract policy. Using the X-TILDE algorithm the problems previously cited are circumvented. The stochastic abstract policy learned for the states derived for locations `l5` and `l8` suggests two abstract actions as policy: `gotoCDRD(A,D)` and `gotoCDCD(A,C)`, as shown in Figure 2(c).

Therefore, when using the abstract policy we can envisage three different ways for the robot to decide which ground action to apply, depending on the algorithm used to generate the FOLDT:

– TILDE: to choose uniformly among all ground actions defined by the substitution θ to the abstract action suggested in the FOLDT leaf.

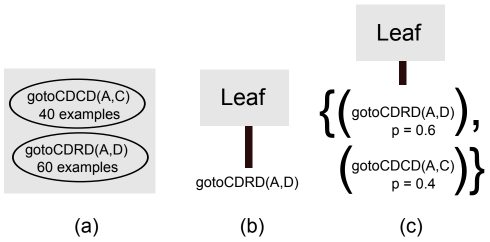


Fig. 2. (a) Number of examples and their respective action candidates for the third leaf (from left to right) of the FOLDT of Figure 1. **(b)** Leaf created by TILDE using only the abstract action `gotoCDRD(A,D)`. **(c)** Leaf created by P-TILDE, with both abstract actions, `gotoCDRD(A,D)` and `gotoCDCD(A,C)`, and their respective frequency of occurrence. In the case of ND-TILDE there is no value associated to each abstract action in the set of abstract actions in the leaf.

- P-TILDE: first to choose probabilistically among all abstract actions in the FOLDT leaf and then uniformly among all ground actions defined by the substitution θ to the chosen abstract action;
- ND-TILDE: first to choose uniformly among all abstract actions in the FOLDT leaf and then also uniformly among all ground actions defined by the substitution θ to the chosen abstract action.

4 Experiments

The states in the maps of Figures 3(a) and 3(b) were defined based on the map's morphological skeleton. Each state has a reference point which is the point where three or more segments of the morphological skeleton intersect. Having chosen the reference point, the states are delimited by the Voronoi diagram of the reference point, i.e., any given point (x, y) of the map will belong to the state S with the nearest reference point.

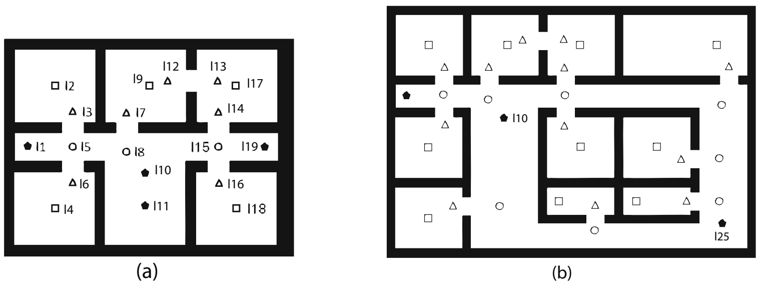


Fig. 3. (a) An example of a relational representation of a simple navigation problem. **(b)** New navigation problem used to show the policy reuse.

In all experiments it was used the same FOLDT, generated by the examples taken from the optimal policy application, whose goal was to reach the center of the upper right room of the map in Figure 3(a) (location `l2`).

The experiments were divided in three parts: in the first one we applied the abstract policy to the same environment and the same task used to build the tree; in the second one we kept the same environment but considered different goals; in the third one we considered a new environment (Figure 3(b)). In all cases we compare the abstract policies with a random policy which, for every given state s, the robot selects uniformly among the actions in the set $A(s)$ and executes it.

The use of the abstract policy in a new task can lead to situations that did not exist in the source problem used to induce the abstract policy. Because of this, the abstract policy can not suggest an adequate action for this situation. One occurrence of this problem occurs in the second set of experiments, where for the abstract state (`in(A)`$\wedge$`isConnected(A,B)`$\wedge$`isCenter(B)`$\wedge$`isCorridor(B)`) the abstract policy gives `gotoCDCC(A,B)` as abstract action. This abstract action is appropriated in four of the goal states (`l2, l4, l9 and l17`), but is not the correct action in the last one (`l18`). In this situation, during the induction of the abstract policy the abstraction algorithm has not received an example covering this situation. Every time the location `l15` is reached the abstract policy will guide the robot to a center of a corridor, and the correct goal `l18` will never be reached.

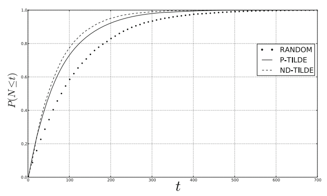
The same kind of problem occurs in the third set of experiments. In this case the abstract policy faces a new situation where for the abstract state (`in(A)` $\wedge$ `isConnected(A,B)` $\wedge$ `isCenter(B)` $\wedge$ `isCorridor(B)` $\wedge$ `isConnected(A,C)` $\wedge$ `isNearDoor(C)` $\wedge$ `isCorridor(C)`) the abstract action `gotoCDCC(A,D)` should be considered to allow the robot to reach all centers of rooms in the environment. The abstract policy lacks this information and because of this the environment is divided in two distinct regions: the first encompassing most of the top states and the second the lower states. Both regions are connected by the locations `l10` and `l25`, but there is no path linking a location in one region to another location of the other region, i. e., there is only a path connecting locations of the same region. Because of this, in this set of experiments we selected the pairs of initial and goal states belonging to the same region.

By letting N be the number of transitions used to achieve the goal, we show for every experiment and for every policy a plot of the cumulative distribution of N, i.e. $P(N \leq t)$, the mean $\mu(N)$, the standard deviation $\sigma(N)$ and the number E_{succ} of episodes where the robot successfully reached the goal, with the corresponding total number of episodes E_{tot}.

Since the mean of N will depend on the relative distance of each $\langle initialstate, goalstate\rangle$ pair used for each episode, we consider as a measure of performance the ratio $\mu(N)_{X-TILDE}/\mu(N)_{RANDOM}$. Hence a number less than one will represent an advantage for the abstract policy with respect to the random policy. The lower this number, the better the performance.

4.1 Experiment 1: Same Environment – Same Goal State

In the first experiment we compare the abstract policies against the random policy in the same conditions that were used to generate the training examples in the FOLDT induction, i. e., the robot should navigate from any location to location `l2` in the environment of Figure 3(a). We randomly selected five initial states for the task, and ran 2000 episodes for each one. Figure 4 shows the cumulative distribution, $\mu(N)$ and $\sigma(N)$ for each policy. The performance ration was: $\frac{\mu(N)_{P-TILDE}}{\mu(N)_{RANDOM}} = 0.69$ and $\frac{\mu(N)_{ND-TILDE}}{\mu(N)_{RANDOM}} = 0.60$.



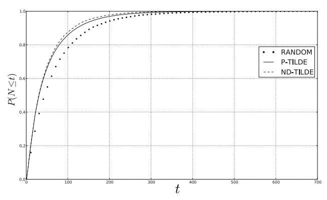
Policy	μ	σ	E_{succ}
RANDOM	113.19	106.94	9986
P-TILDE	77.64	77.10	9998
ND-TILDE	68.76	64.91	9999

$E_{tot} = 10000$

Fig. 4. Cumulative distribution, $\mu(N)$ and $\sigma(N)$ for the first set of experiments

4.2 Experiment 2: Same Environment – Different Goal States

In the second experiment we kept the same environment but changed the goal state. We chose as goal states the remaining reachable center of rooms in the environment shown in Figure 3(a) (`l4,l9,l17,l18`). For each goal we ran 2000 episodes with randomly selected initial states. Figure 5 shows the cumulative distribution, $\mu(N)$ and $\sigma(N)$ for each policy. The performance ration was: $\frac{\mu(N)_{P-TILDE}}{\mu(N)_{RANDOM}} = 0.75$ and $\frac{\mu(N)_{ND-TILDE}}{\mu(N)_{RANDOM}} = 0.70$.



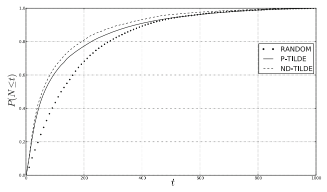
Policy	μ	σ	E_{succ}
RANDOM	68.27	74.45	17992
P-TILDE	51.45	58.30	17999
ND-TILDE	48.11	51.59	18000

$E_{tot} = 18000$

Fig. 5. Cumulative distribution, $\mu(N)$ and $\sigma(N)$ for the second set of experiments

4.3 Experiment 3: New Environment

For each $\langle initialstate, goalstate \rangle$ pair we ran 2000 episodes. Figure 6 shows the cumulative distribution, $\mu(N)$ and $\sigma(N)$ for each policy. The performance ratio was: $\frac{\mu(N)_{P-TILDE}}{\mu(N)_{RANDOM}} = 0.78$ and $\frac{\mu(N)_{ND-TILDE}}{\mu(N)_{RANDOM}} = 0.68$.



Policy	μ	σ	E_{succ}
RANDOM	177.97	172.73	7971
P-TILDE	138.84	176.39	7901
ND-TILDE	120.49	155.25	7974
$E_{tot} = 8000$			

Fig. 6. Cumulative distribution, $\mu(N)$ and $\sigma(N)$ for the third set of experiments

5 Conclusion and Future Works

We presented a new approach for the policy reuse in robot navigation problems. The policy reuse is obtained by using a FOLDT that represents an abstract policy. The FOLDT is induced using the TILDE algorithm. Our approach, named X-TILDE, extends the TILDE algorithm and creates an FOLDT that represents a stochastic abstract policy. This stochastic abstract policy retains more information about the structure of the navigation problems and solves situations that the abstract policy generated by the original TILDE algorithm can not solve.

The quality of the abstract policy generated by the TILDE algorithm depends on the structure of the problem and the relational description chosen. Whereas the TILDE algorithm does not provide solution to many problems, our experiments showed that our approach can solve a much larger set of problems. The experiments also showed that the abstract policy generated by X-TILDE has a better performance than a random policy in a new task, which leads us to conclude that there was indeed a transfer of knowledge between tasks.

As the experiments showed, the performance of P-TILDE is worse than the performance of ND-TILDE. We believe it is because the P-TILDE algorithm classifies states based on the ground policy, but aggregates them based only on their relational descriptions. Since ND-TILDE does not take into account the distribution of occurrence of actions, the results showed that it is better to use a more conservative approach. On the other hand, the random policy, which is the most conservative of all policies, showed worse results. This leads us to investigate in the future a better compromise between these policies and to find ways to determine the best balance between the approaches.

Despite being more robust, our approach does not guarantee that every task can be performed. This depends on the experience in the source task, and may fail in situations that did not occur in the source task. In [4] random perturbations are applied to the source task so that even in unrepresentative tasks we can have the transfer of knowledge robustly. Another important topic to be investigated is to define how the source problems should be selected, which should be simple but should able to generate useful policies for more complex tasks.

The main motivation for reusing a policy in a robot navigation domain is to guide the robot in solving similar problems. If the abstract policy learned is not perfect, the agent must learn from the knowledge that was transferred. In this case, the abstract policy can be used as a advice to guide the exploration in

RL algorithms. This combination aims to accelerate the learning process of an optimal political in the target task [13].

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, 305512/2008-0).

References

1. Kersting, K., Plagemann, C., Cocora, A., Burgard, W., Raedt, L.D.: Learning to transfer optimal navigation policies. Advanced Robotics: Special Issue on Imitative Robots 21, 1565–1582 (2007)
2. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research 10, 1633–1685 (2009)
3. Madden, M.G., Howley, T.: Transfer of experience between reinforcement learning environments with progressive difficulty. Artif. Intell. Rev. 21, 375–398 (2004)
4. Sherstov, A.A., Stone, P.: Improving action selection in MDP's via knowledge transfer. In: Proc. of the 20th National Conference on Artificial Intelligence (2005)
5. Bianchi, R., Ribeiro, C., Costa, A.: Accelerating autonomous learning by using heuristic selection of actions. Journal of Heuristics 14, 135–168 (2008), doi:10.1007/s10732-007-9031-5
6. Lane, T., Wilson, A.: Toward a topological theory of relational reinforcement learning for navigation tasks. In: Proc. of the 18th Int. Florida Artificial Intelligence Research Society Conference (2005)
7. Kersting, K., Otterlo, M.V., Raedt, L.D.: Bellman goes relational. In: Brodley, C.E. (ed.) Proc. of the 21st Int. Conference on Machine Learning, Banff, Alberta, Canada, pp. 465–472 (2004)
8. Otterlo, M.V.: The logic of adaptive behavior: knowledge representation and algorithms for the Markov decision process framework in first-order domains. PhD thesis, University of Twente, Enschede (2008)
9. Hoey, J., St-Aubin, R., Hu, A.J., Boutilier, C.: Spudd: Stochastic planning using decision diagrams. In: Proc. of Uncertainty in Artificial Intelligence, Stockholm, Sweden (1999)
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
11. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101, 285–297 (1998)
12. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
13. Matos, T., Bergamo, Y.P., da Silva, V.F., Cozman, F.G., Costa, A.H.R.: Simultaneous abstract and concrete reinforcement learning. In: Proc. of the 9th Symposium on Abstraction, Reformulation and Approximation (2011)