Learning Relational Navigation Policies

Alexandru Cocora*, Kristian Kersting*, Christian Plagemann†, Wolfram Burgard†, Luc De Raedt*
*Machine Learning Lab, †Autonomous Intelligent Systems Group
Institute for Computer Science, University of Freiburg
Georges-Koehler-Allee, Building 079, 79110 Freiburg, Germany
{cocora,kersting,plagem,burgard,deraedt}@informatik.uni-freiburg.de

***Abstract*— Navigation is one of the fundamental tasks for a mobile robot. The majority of path planning approaches has been designed to entirely solve the given problem from scratch given the current and goal configurations of the robot. Although these approaches yield highly efficient plans, the computed policies typically do not transfer to other, similar tasks. We propose to learn relational decision trees as *abstract* navigation strategies from example paths. Relational abstraction has several interesting and important properties. First, it allows a mobile robot to generalize navigation plans from specific examples provided by users or exploration. Second, the navigation policy learned in one environment can be transferred to unknown environments. In several experiments with real robots in a real environment and in simulated runs, we demonstrate the usefulness of our approach.**

I. Introduction

For various tasks such as delivery, guidance, rescue etc, mobile service robots need to navigate through the environment. In the past, the vast majority of navigation approaches have dealt with solving these problems from scratch during operation. Whereas such approaches yield highly efficient paths [1], [2], they typically do not take into account solutions to similar problems. In addition, these navigation plans cannot easily be communicated to humans, which makes it hard to instruct the robot about typical navigation behaviors. In this paper, we consider the problem of learning abstract navigation plans for mobile robots by generalizing from navigation paths, which were successful in previous or similar situations. The key idea is to utilize labels assigned to the individual places in the environment and to generalize sequences of these labels corresponding to the places traversed by the robot while performing its task.

The problem of planning trajectories of mobile robots has been studied intensively in the past, as the capability of effectively planning its motions is "eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world" [3]. The different types of planning problems can coarsely be classified according to the information provided to the robot. The classical path planning problem is the situation in which the robot has perfect knowledge about the environment as well as its starting point and its goal position. More complex problems emerge when the robot only possesses partial knowledge. For example, when the location of the target is unknown, the robot has to search for the target. In situations, in which the environment is unknown but the target location is known, D* [4] or LRTA* [5] are popular algorithms to guide the robot to the goal location. Throughout this paper, we consider the more complex situation in which the location of the target point is not given a priori. Such a situation, for example occurs, when a robot has to find the entrance hall in a hotel or in a large office building. For such problems, different algorithms including depth-first search and uninformed LRTA* (see Koenig [6] for a comprehensive comparison) have been proposed. Moreover, in most situations the actions of the robot are non deterministic. Here, approaches based on Markov decision processes (MDPs) [7] have been proposed [8]. MDPs provide a sound theoretical framework to deal with uncertainty related to the robot's motor and perceptive actions during both planning and plan execution phases.

Whereas these techniques provide highly elegant and often also efficient solutions to the corresponding problems, they do not have the ability to improve their performance by learning from past experience within similar tasks (e.g., entrance halls found in other office buildings). Our approach alleviates this situation by adopting techniques from *relational reinforcement learning* [9], [10], i.e., reinforcement learning within a relational representation to learn general search preferences for navigation problems. More precisely, our technique starts from a set of specific example navigation plans, which can either be computed by solving a relational MDP or can be obtained from a helpful teacher, where it is assumed that a set of labels can be assigned to each position in the configuration space of the robot. Such labels can be obtained robustly by analyzing sensor measurements and their temporal evolution, see [11], [12]. The observed labels are used to form (relational) state descriptions of a relational Markov decision process (RMDP). We then apply relational learning techniques for generating abstractions. As a result, we obtain a relational decision tree, which expresses preferences about navigation actions. These preferences can then be used by the robot to generate navigation actions. As our experiments show, the navigation algorithm can deal with noise in the observed labels and gracefully degrades to random search when the noise level increases.

In the past few years, relational representations in machine learning and AI received a lot of attention, see. e.g. [13], and is known under the name *statistical relational learning* (SRL). This appears to be an appropriate time to apply SRL techniques within robotics. The advantages of the SRL approach are threefold. First, the learned navigation preferences can be directly transfered to alternative instances of the same

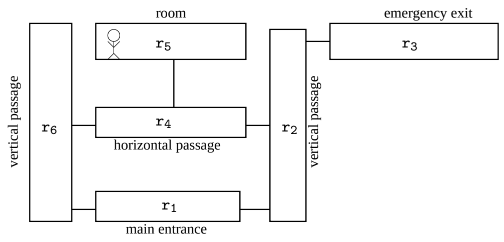


Fig. 1. A map of hotel h. The robot in room $\mathtt{r}_5$ is supposed to find its way to the main entrance $\mathtt{r}_1$ of the hotel.

navigation problem (e.g., searching for another exit in the same building). Additionally, they can directly be transferred from one environment to another one and in this way enable a robot to efficiently carry out similar navigation tasks in even unknown environments. Finally, our approach can be applied to settings in which the example plans are not generated by a robot but are provided by a human and have no guarantees of being optimal. This setting is often called behavioral cloning or learning by imitation. Our experiments show that also in such cases our approach can be beneficial.

The paper is organized as follows. We first briefly review MDPs and RMDPs. Then, we show how to learn abstract navigation policies in the framework of RMDPs, and how to use them to guide the search of a mobile robot. Section IV reports on several experiments carried out on a real robot as well as in simulation, which shows that our approach leads to efficient navigation plans.

II. Markov Decision Processes

Our running example will be the *hotel* world where the task of the robot is to find its way out of an unknown hotel. Consider hotel h in Figure 1. The mobile robot is in room $\mathtt{r}_5$. At each time, the robot can go from one room to another, say $\mathtt{r}_5$ to $\mathtt{r}_4$. The action is probabilistic, i.e., with probability p the action succeeds and the robot will be in $\mathtt{r}_4$, and with probability $1-p$ the action fails and the robot stays in $\mathtt{r}_5$.

A natural formalism to encode the hotel world are *Markov decision processes* (MDPs) [14]. MDPs are tuples $M = \langle S, A, T, R\rangle$, where S is a set of states such as $\mathtt{r}_1, \mathtt{r}_2, \ldots, \mathtt{r}_5$, A is a set of actions such as $\mathtt{goto}(\mathtt{r}_5, \mathtt{r}_4)$, $T : S \times A \times S \to [0,1]$ a *transition model*, and $R : S \times A \times S \to [0,1]$ a *reward model*. The set of actions *applicable* in a state $s \in S$ is denoted $A(s)$. A transition from state $i \in S$ to $j \in S$ caused by some action $a \in A(i)$ occurs with probability $T(i,a,j)$ and a reward $R(i,a,j)$ is received, e.g., 10, when entering $\mathtt{r}_5$, and 0 otherwise. T defines a proper probability distribution if for all states $i \in S$ and all actions $a \in A(i)$: $\sum_{j\in S} T(i,a,j) = 1$. A deterministic *policy* $\pi : S \to A$ for M specifies which action $a \in A(s)$ to executed when the agent is in state $s \in S$, i.e. $\pi(s) = a$. Given a policy π for M, and a *discount factor* $\gamma \in [0,1]$, which discounts future rewards, the *state value function* $V^\pi : S \to \mathbb{R}$ represents the value of being in a state following policy π, w.r.t. expected rewards. A policy π^* is optimal if $V^{\pi^*}(s) \geq V^{\pi'}(s)\ \forall s \in S$ and $\forall \pi'$. The optimal value function is denoted V^*. One of the standard techniques for exactly solving MDPs is *value iteration* (VI). The VI algorithm assumes that the state space is represented as a table and can be stated as follows: starting with a value function V_0 over all states, we iteratively update the value of each state according to

$$V_{t+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_t(s')] \tag{1}$$

to get the next ($t = 1,2,3,\ldots$). VI is guaranteed to converge in the limit towards V^*.

Traditional MDPs and VI as expressed by Equation (1) are essentially propositional in that each state must be represented using a separate proposition. Therefore, they are severely limited in expressiveness and do not really capture the structure of the underlying class of problems. As a consequence, it is hard to generalize policies across domains with similar properties. For instance, propositional policies for the hotel in Figure 1 cannot directly be applied in other hotels.

III. Learning Relational Navigation Policies

Relational MDPs (RMDPs, see below) combine relational logic with MDPs. Using RMDPs, it becomes possible to generalize such policies even for those cases where the hotels may possess a varying number of objects (rooms) and relations (connections) among them.

A. Relational Logic

The hotel world can elegantly be represented using relational logic. Reconsider hotel h in Figure 1: there are rooms $\mathtt{ro}(\mathtt{r}_1)$, $\mathtt{ro}(\mathtt{r}_2), \ldots, \mathtt{ro}(\mathtt{r}_5)$ which are connected: $\mathtt{con}(\mathtt{r}_1,\mathtt{r}_2)$, $\mathtt{con}(\mathtt{r}_1,\mathtt{r}_6)$, $\mathtt{con}(\mathtt{r}_2,\mathtt{r}_3)$, $\mathtt{con}(\mathtt{r}_2,\mathtt{r}_4)$, $\mathtt{con}(\mathtt{r}_4,\mathtt{r}_5)$, $\mathtt{con}(\mathtt{r}_4,\mathtt{r}_6)$ together with the symmetric facts. There are several types of rooms such as horizontal, $\mathtt{hPass}(\mathtt{r4})$, and vertical passages, $\mathtt{vPass}(\mathtt{r}_2)$ and $\mathtt{vPass}(\mathtt{r}_6)$. Furthermore, there are emergency exits, $\mathtt{eExit}(\mathtt{r}_3)$, and entrance halls, $\mathtt{main}(\mathtt{r}_1)$. At each time the robot is in a room, $\mathtt{in}(\mathtt{r}_5)$, there are several actions the robot can take. One typical such action is going from a room $\mathtt{R}$ into a horizontal passage $\mathtt{H}$, $\mathtt{goto\_RHP}(\mathtt{R},\mathtt{H})$. With some probability actions might fail, i.e., the robot stays in the current room. The task for the robot is to find its way out of a hotel, i.e., to "enter" $\mathtt{main}(\mathtt{r}_1)$.

In *relational* logic, expressions of the form $\mathtt{p}(\mathtt{t}_1, \ldots \mathtt{t}_\mathtt{m})$, where a relation symbol $\mathtt{p}$ followed by a bracketed m-tuple of terms $\mathtt{t}_\mathtt{i}$, are called *atoms*. A *term* is either a variable $\mathtt{R}$ or a constant $\mathtt{r}_1$. We follow the convention that variables start with an upper case and constants with a lower case character.

The main idea underlying RMDPs is to replace the propositional symbols used in MDPs by abstract states. An *abstract state* is a conjunction Z of logical atoms, i.e., a logical query and represents a set of states. Consider, e.g., the abstract state $Z \equiv \mathtt{in}(\mathtt{R}), \mathtt{con}(\mathtt{R},\mathtt{R}'), \mathtt{ro}(\mathtt{R}), \mathtt{ro}(\mathtt{R}')$. It summarizes situations in which a robot is in a room connected to another room. An instance z of Z is for example $z \equiv \mathtt{in}(\mathtt{r}_5), \mathtt{con}(\mathtt{r}_4,\mathtt{r}_5), \mathtt{hPass}(\mathtt{r}_4), \mathtt{ro}(\mathtt{r}_4), \mathtt{ro}(\mathtt{r}_5)$; there exists a substitution θ such that $Z\theta \subseteq z$. A substitution θ is an assignment of terms to variables $\{\mathtt{X}_1/\mathtt{t}_1, \ldots \mathtt{X}_\mathtt{n}/\mathtt{t}_\mathtt{n}\}$ where $\mathtt{X}_\mathtt{i}$ are variables and all $\mathtt{t}_\mathtt{i}$ are terms. A term, atom or conjunction is called *ground* if it contains no variables. Conjunctions are implicitly assumed to be *existentially quantified*. A conjunction A is said to be subsumed by a conjunction B, denoted by $A \preceq_\theta B$, if there exists a substitution θ such that $B\theta \subseteq A$.

B. Relational Markov Decision Processes

Using these notions from relational logic, we now briefly review the key ingredients of RMDPs: *abstract actions* and *abstract rewards*. For more details, we refer to [15].

An *abstract action*[1] is a rule $H \xleftarrow{p:A} B$ where A is an atom representing the name and the arguments of the action and B is an abstract state denoting the preconditions of A. H is an abstract state and represents the successful outcome of A. The value p is the probability that the action succeeds. The semantics of an abstract action are: *If the current state b is subsumed by B, i.e., $b \preceq_\theta B$, then taking action A will result in $[b \setminus B\theta] \cup H\theta$ with probability p. With probability $1-p$ the action fails, i.e., we stay in b.* As an illustration, consider

$$\begin{array}{l}\mathtt{in(R'),con(R,R')},\\ \mathtt{hPass(R'),ro(R)}\\ \mathtt{ro(R'),R\neq R'}\end{array} \xleftarrow{0.9:\mathtt{goto\_RHP(R,R')}} \begin{array}{l}\mathtt{in(R),con(R,R')},\\ \mathtt{hPassl(R'),ro(R)}\\ \mathtt{ro(R'),R\neq R'}\end{array}$$

which describes that a robot is going from room $\mathtt{R}$ into a horizontal passage $\mathtt{R'}$ with probability 0.9. Applied to the above state z the action $\mathtt{goto\_RHP(r,r')}$ will yield $z' \equiv \mathtt{in(r_4),con(r_4,r_5),hPass(r_4),ro(r_4),ro(r_5)}$ with probability 0.9 and z with probability 0.1.

The *abstract reward* model specifies the rewards generated by entering abstract states. It is specified as a finite list of *value rules* of the form $c \leftarrow B$ were B is an abstract state and $c \in \mathbb{R}$. To any abstract state Z, V assigns the maximal value c of all matching value rules $c \leftarrow B$ to Z as value. A rule matches if $Z \preceq_\theta B$. Consider e.g. $10 \leftarrow \mathtt{in(r')}$ and $0 \leftarrow \mathtt{true}$. It assigns 0 to z but 10 to z'. Using $\mathtt{true}$ in the last value rule assures that all states are assigned a value. This simple reward model is expressive enough for the hotel world, which is basically a *shortest-path problem*: the goal to reach is $\mathtt{main}$. When $\mathtt{main}$ is entered, the process ends. Such episodic tasks are encoded using so-called *absorbing states*, which can be specified by a set of queries, e.g., $\mathtt{main(R)}$. In our example, z is not absorbing but z' is. In addition, integrity constraints can be employed to exclude impossible states, cf. [15].

C. Relational Navigation Policies

Let us now discuss how to compute (navigation) policies from RMDPs. The key observation is that each RMDP induces a traditional MDP [15], which can be obtained by starting in some initial ground state and then applying each abstract transition until no more new ground states can be computed. Thus, the existence of an optimal policy π for each (resulting) ground MDP is guaranteed. In the hotel world, a navigation pattern might be

$$\mathtt{goto\_RHP(r_5,r_4)} \leftarrow \begin{array}{l}\mathtt{in(r_5),hPass(r_4),ro(r_4)},\\ \mathtt{con(r_4,r_5),ro(r_5),r_4\neq r_5}\end{array} \qquad (2)$$

[1]For the sake of simplicity, we will consider only actions which succeed or fail and which do not cause any costs. The more general cases are straightforward.

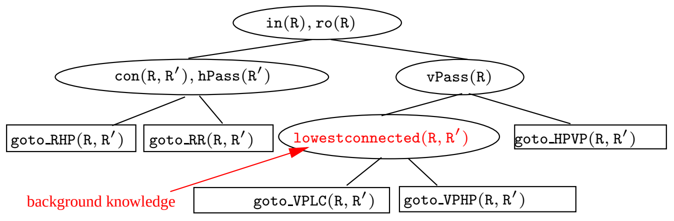


Fig. 2. A relational decision tree representing a relational navigation policy for the hotel world.

which states that the robot will go to the horizontal passage $\mathtt{r_4}$ when it is in room $\mathtt{r_5}$. Of course, such policies are extensional or propositional in the sense that they specify for each ground state separately which action to execute. Instead, we would like to learn an *abstract policy*, which intentionally specifies the action to take for an abstract state, i.e., for the set of ground states it makes abstraction from. More formally, an *abstract*, i.e., *relational navigation policy*, is a finite set of *relational navigation rules* of the form $A \leftarrow Z$ where A is an abstract action and Z is an abstract state. For instance, the relational navigation rule

$$\mathtt{goto\_RHP(R,R')} \leftarrow \begin{array}{l}\mathtt{in(R),con(R,R'),hPass(R')}\\ \mathtt{ro(R),ro(R'),R\neq R'}\end{array} \qquad (3)$$

abstracts from the specific rooms used in (2).

To learn a relational navigation policy, we start from a set of traces t_i, i.e., ground situation-action sequences that lead to a goal state. These specific situation-action sequences can be optimal (for instance, if they were obtained by computing the optimal policy for a fully known map of the environment) or not (for instance, if they are provided by a human that shows the robot how to proceed from a particular initial goal to a goal state). Whereas the first case could correspond to the situation where the model is known, the second one corresponds to a model-free case, and also allows to learn from imitation or perform what is called *behavioral cloning*. The key idea is that each trace t_i describes a situation-action sequence, for instance for leaving a hotel. More precisely, each t_i consists of ground navigation rules such as (2). Each rule describes an interpretation, $\mathtt{in(r_5),con(r_4,r_5),hPass(r_4),ro(r_4),ro(r_5),r_4\neq r_5}$, i.e, a simple enumeration of all ground facts the robot needs to know – the rooms, the connections among the rooms, the types of the rooms, i.e., room, horizontal passage, vertical passage, elevator, etc. – in order to take the associated optimal actions $\mathtt{goto\_RHP(r_5,r_4)}$. The task then is to induce a relational navigation policy based on these situation-action pairs that makes abstraction of the experience provided to the agent. This can be realized using the *learning from interpretations* settings well studied in the field of *inductive logic programming* [16] where relational programs are induced from interpretation-class pairs. One standard approach to employ during generalization are relational decision trees.

A relational decision tree [17] (see Figure 2) is a binary decision tree in which each node contains a conjunction such as $\mathtt{in(R),ro(R)}$. Each node captures a logical test, which either succeeds or fails when applied to a particular state. If it succeeds, the left subtree is considered; otherwise the right one. Moreover, nodes may share variables with their ancestor nodes such $\texttt{con}(\texttt{R},\texttt{R}'), \texttt{hPass}(\texttt{R}')$. The test to be performed at each node consists of its conjunction together with the conjunctions on the succeeding path from the root to the node for instance $\texttt{in}(\texttt{R}), \texttt{ro}(\texttt{R}), \texttt{con}(\texttt{R},\texttt{R}'), \texttt{hPass}(\texttt{R}')$. Leafs represent the action to be taken in the abstract state consisting of the conjunctions along the path to the leaf. For instance, the tree essentially encodes the relational navigation rule (3) in its leftmost path and also suggests to take action $\texttt{goto\_RHP}(\texttt{r}_5, \texttt{r}_4)$ in state z. To induce the tree, we essentially employ Quinlan's well-known C4.5 [18] scheme with the information gain as splitting criterion, for more details see [17]. To summarize, our approach works as follows:

1) Observe a number of successful ground state-action sequences
2) Induce a relational navigation policy in the form of a relational decision tree from this experience.

The resulting abstract navigation policy typically – as in our experiments – uses local information only, i.e., the environment does not need to be completely known.

Indeed, this is akin to explanation-based learning (EBL) [19], [20], where subsequent to a successful problem solving session a proof is constructed that explains the success. The proof is then generalized to a description of states which can be solved in the same way. In state-space problems – as we are investigating – proofs correspond to showing that a sequence of actions achieves a goal and EBL corresponds to goal regression over an state-action sequence. Therefore, it is not surprising that EBL has been used as generalization algorithm within the Prodigy system [21] to learn general control rules from specific examples of problem solving episodes. Later, Dietterich and Flann [22] combined this idea with reinforcement learning by associating these generalized state descriptions with values obtained from value iteration. Subsequently, Boutilier *et al.* [23] and Kersting *et al.* [15] generalized Dietterich and Flann's approach to relational domains, i.e., RMDPs. Recently, Mausam and Weld [10] suggested to approximate the value function by inducing a relational regression tree from observed traces. Unfortunately, the relational description of states that share a value becomes increasingly complex as these states get farther and farther from the goal while the number of states covered by an abstract state reduces dramatically. This results in a large number of value rules. Indeed this has been observed to be the case in early EBL systems and has been called the *utility problem* [22]. To avoid this problem, our approach works directly with state-action sequences and inductively generalizes them into relational policy trees. At the same time, this has the advantage that – in contrast to EBL – no model is required, which allows to apply our techniques onto behavioral cloning.

One particularly interesting case, on which we will focus in the experiments, is concerned with learning from optimal state-action sequences. These can actually be generated if the model, i.e., the RMDP R is fully known. To obtain an optimal state-action sequence one has to ground the RMDP and then compute an optimal navigation policy for the resulting MDP using any MDP solver. Thus, our approach does not face the utility problem and, thus, typically learn more compact policies than approaches approximating the value function.

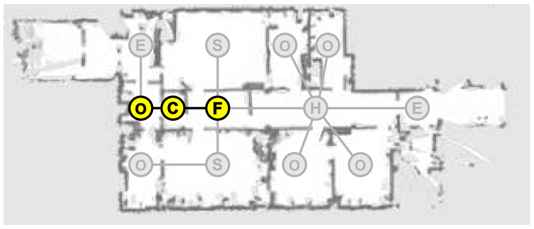


Fig. 3. Map of the real office environment in which the experiments with our robot were carried out. When the robot was in room "C", only three node labels were observable (black). All other labels as well as the overall topology of the environment were unknown (gray). The used navigation policies were learned from different real buildings.

IV. Experiments

Our algorithm has been evaluated in experiments carried out with a real ActivMedia Pioneer 2-DX8 robot equipped with two SICK laser range finders as well as in simulation. The goal of the experiments is to demonstrate that the abstract navigation plans can be used to effectively control a mobile robot to reach its target location even in unknown environments.

A. Implementation Details

In our current system, we use the system SPUDD [24] to solve the MDPs and to generate example navigation plans. We assume that the robot can identify the type of place at its current location as well as the type of place a door leads to [11], [12]. We do not require this information to be free of noise as one of our experiments demonstrates. Under these assumptions, we perform a forward search guided by the learned navigation policy. That is, we start in some state and then determine which action to perform next by evaluating the relational decision tree on the current state. We perform the action, observe the next state, and repeat the overall process. Since relational (navigation) policies are not deterministic, the system needs to choose among several equally likely actions. We choose uniformally among all possibilities and put the ones not chosen in a list *AltS*. Whenever the robot encounters a loop or a dead end, it calculates the shortest path from its current state to every state in *AltS* and chooses the one with the shortest distance to the current location. In case *AltS* is empty, we put every state connected to an already visited state into *AltS*.

B. Navigation in an Office Environment

The first experiments are designed to demonstrate that our approach results in effective navigation behaviors in real-world scenarios. The experiments have been carried out with a real mobile robot in a typical office environment (see Figure 3). The task of the robot was to find the entrance hall of the building using a navigation policy that was learned by abstracting from optimal trajectories calculated given the

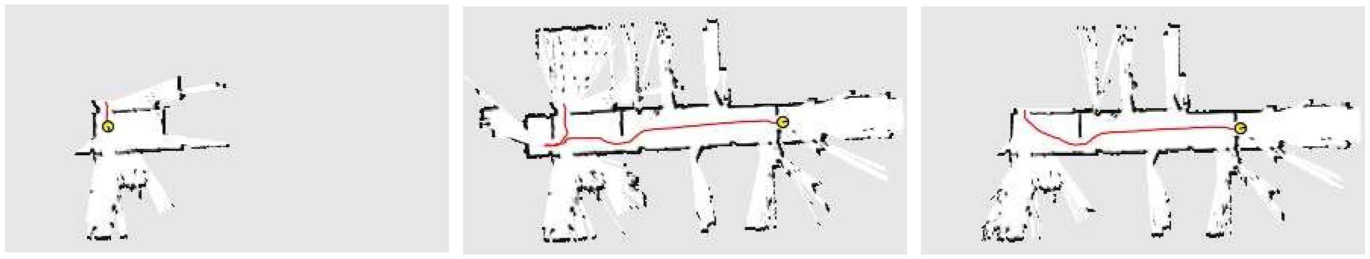


Fig. 4. Application of an abstract policy for finding the entrance hall of a building. The robot first leaves the room and enters the corridor (left). Then it samples randomly and turns right (middle). At the same step in a different experiment, it chose the corridor to its left directly (right).

floor plans of two other buildings. The actual map of the environment was unknown to the robot and just the labels of neighboring rooms could be observed. In this experiment, we provide the labels incrementally when requested by the robot. In principle, such labels can be obtained by analyzing sensor measurements and their temporal evolution, see [11], [12].

In the two experiments described here, the robot started in the upper seminar room, labeled S. According to the navigation policy, the first action of the robot was to leave the room and to enter the corridor labeled F. The situation after carrying out this action together with the part of the environment observed by the robot thus far is depicted in the left image of Figure 4. At this point, the navigation policy outputs two equally likely alternatives: corridor C and hallway H. In the first experiment, it chose to turn right and enter the corridor labeled C. Since the place labeled O is not a corridor, the robot decided to return to F and to choose the alternative corridor adjacent to F, which was corridor H. From there it proceeded to the area labeled E, which corresponds to the entrance hall. The resulting trajectory of the robot is depicted in the middle image of Figure 4. The rightmost image of Figure 4 shows the resulting trajectory in case the optimal corridor H is sampled directly when the robot is in F.

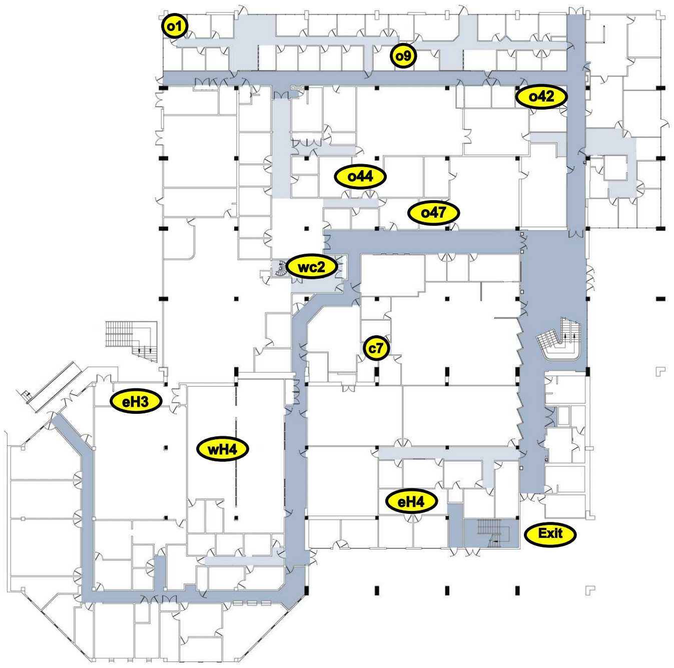


Fig. 5. Map used for the second set of experiments.

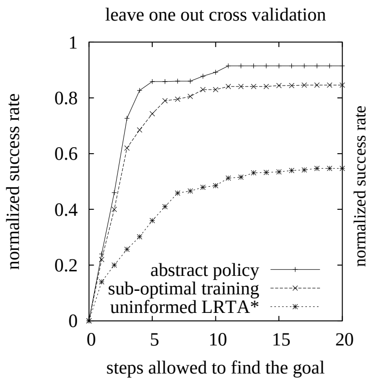
C. Simulation Experiments

To quantitatively evaluate the performance of our approach, we compared it with the optimal paths as well as real-time search methods, which interleave planning (via local searches) and plan execution. A popular real-time search method for robot navigation in unknown terrain is *uninformed* LRTA* with maximum lookahead [5].

We ran several simulation experiments on maps of real buildings such as the one depicted in Figure 5. From the outlines of these buildings we manually generated an annotated topological map which then was used for calculating paths. The robot observes the topological map only incrementally. To evaluate the performance, we randomly chose the starting locations. These starting locations are indicated by yellow/gray labels in Figure 5. The goal of the robot in all tasks was to find the exit of the building as indicated in the figure. On average, the optimal plan length was 3.1 ± 0.99 (mean $\pm$ standard deviation). Our method achieved 4.9 ± 2.18, whereas uninformed LRT* performed 30.7 ± 18.25 steps to reach the exit. Thus, our approach required substantially fewer steps than uninformed LRTA*. Note that in these experiments we count each room visited as a step. More over, LRTA* performed in no case superior to our approach. This illustrates, that our approach substantially increases the efficiency of the resulting navigation plans. At the same time the plans are only 1.8 ± 1.55 steps longer than the optimal plans. In additional experiments not reported here, a two-sampled t-test revealed that the improvement obtained by the abstract policy search is significant on the $\alpha = 0.05$ level.

D. Behavioral Cloning

One important aspect of our approach is that the training instances do not need to be the optimal paths. Rather, they can also be generated by manually sketching possible trajectories. The final experiment described here has been carried out to analyze the degradation of the performance in the case the system has to learn from sub-optimal training instances. To evaluate this, we performed an experiment in which we used 20 maps of hotels where each hotel had 15-20 different areas. In a leave-one-out cross-validation we tested how the performance of our approach compares to that of the optimal policy and the real-time search algorithm. The general policy was learned on 19 maps and than evaluated on the one left out.



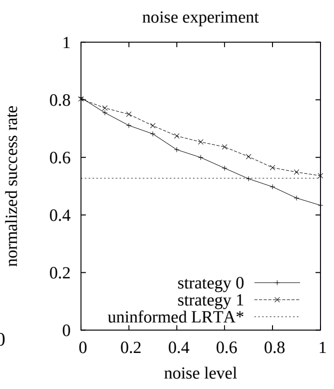


Fig. 6. Normalized success rates (see text) for different maximal steps allowed to reach the goal (left). Normalized success rates for a varying level of noise in the observed labels (right).

We performed 5 restarts and started randomly in one of the areas. Figure 6 shows as performance measure the *normalized success rate* of the different approaches which is defined as $1/N \sum_{i=1}^{N} l^*/l_i \cdot r_i^j$ where $N = 20 \cdot 5$ is the number of runs, l^* is the length of the optimal path, l_i the length of the path in the i-th run, and r_i^j indicates whether the goal has been reached within j steps. All differences are significant (two-sampled t-test, $\alpha = 0.05$). Again, our approach is substantially better than uninformed LRTA*. Additionally, the policy learned from sub-optimal and hand-drawn trajectories is only 10% worse than the policy learned from optimal trajectories. Note that we also found that the policy abstracted from hand-drawn trajectories still yields better paths than manually generated paths, which where almost 1.8 times longer than the optimal ones whereas those generated by our algorithm showed only 25% overhead.

E. Observation noise

A robots perception of the world is never perfect. Our algorithm is able to cope with noisy label observations and gracefully degrades to uninformed LRTA* when the noise level increases. We implemented two strategies for dealing with situations where the belief about place labels has to be revised. Whereas Strategy 0 always returns to the previous place when an inconsistent place label was detected, Strategy 1 stays in the new room, updates the faulty label information, and continues navigating from there. The right diagram of Figure 6 shows how the navigation performance changes with a varying level of observation noise. The results were obtained from 1470 simulated runs in five hotels. The noise level specifies the probability with which a label is observed as a different one. It can be seen in the diagram that Strategy 1 outperforms Strategy 0 for all noise levels and that its performance smoothly degrades to the one of uninformed LRTA* when the noise level approaches 1.

V. Conclusions

This paper presents a new approach for generating abstract navigation policies using relational learning. The key idea is to learn a relational decision tree from sequences of places traversed by a robot while it carries out its task. The resulting tree can then be used to guide the search of the robot for the same and similar tasks. The advantages of our approach are that relational abstraction allows to generalize from previously planned paths and to transfer policies across tasks in even previously unseen environments. This should also be useful for high level planning tasks such as manipulation or delivery.

Our algorithm has been evaluated in experiments with real robots as well as in simulation runs. The results demonstrate that the learned policies are highly efficient and outperform uninformed LRTA* with maximum lookahead. In experiments not reported here, the learned policies were even better than policies provided by humans. Additionally, we have presented an experiment in which we learn trajectories from sketched examples provided by users.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.
[2] S. LaValle, *Planning Algorithms*. Cambridge Univ. Press, 2006.
[3] J. Latombe, *Robot Motion Planning*. Kluwer Acad. Publ., 1991.
[4] A. Stentz, "The focused D* algorithm for real-time replanning," in *Proc. of IJCAI-95*, 1995.
[5] R. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189–211, 1990.
[6] S. Koenig, "Agent-centered search." *AI Magazine*, vol. 22, no. 4, pp. 109–132, 2001.
[7] D. Bellman, *Dynamic Programming*. Princeton Univ. Press, 1957.
[8] S. Koenig and R. Simmons, "Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models," in *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*. MIT Press, 1998, pp. 91 – 122.
[9] S. Džeroski, L. De Raedt, and K. Driessens, "Relational reinforcement learning," *Machine Learning*, vol. 43, pp. 7–52, 2001.
[10] Mausam and D. Weld, "Solving Relational MDPs with First-Order Machine Learning," in *Proc. ICAPS Workshop on Planning under Uncertainty and Incomplete Information*, 2003.
[11] A. Rottmann, O. Martínez Mozos, C. Stachniss, and W. Burgard, "Semantic place classification of indoor environments with mobile robots using boosting," in *Proc. of AAAI-05*, 2005, pp. 1306–11.
[12] O. M. Mozos, C. Stachniss, and W. Burgard, "Supervised learning of places from range data using Adaboost," in *Proc. of ICRA-05*, 2005.
[13] L. De Raedt, T. Dietterich, L. Getoor, and S. Muggleton, Eds., *Dagstuhl-Seminar* 5051 *on Probabilistic, Logical and Rel. Learning*, 2005.
[14] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[15] K. Kersting, M. Van Otterlo, and L. De Raedt, "Bellman goes Relational," in *Proc. of ICML-04*, 2004, pp. 465 – 472.
[16] S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *J. of Logic Progr.*, vol. 19, no. 20, pp. 629–679, 1994.
[17] H. Blockeel and L. De Raedt, "Top-down induction of first order logical decision trees," *Artificial Intelligence*, vol. 101, pp. 285–297, 1998.
[18] J. Quinlan, "C4.5: Programs for machine learning," Morgan Kaufmann series in machine learning, 1993.
[19] T. Ellman, "Explanation-based Learning: A Survey of Programs and Perspectives," *ACM Comp. Surveys*, vol. 21, no. 2, pp. 163–221, 1989.
[20] M. Lent and J. Laird, "Learning procedural knowledge through observation," in *Proc. of the First International Conference on Knowledge Capture (K-CAP), Canada*, 2001, pp. 179–186.
[21] S. Minton, J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni, and Y. Gil, "Explanation-Based Learning: A Problem Solving Perspective," *Artificial Intelligence*, vol. 40, pp. 63–118, 1989.
[22] T. G. Dietterich and N. S. Flann, "Explanation-based learning and reinforcement learning: a unified view," *Machine Learning*, vol. 28, pp. 169–210, 1997.
[23] C. Boutilier, R. Reiter, and B. Price, "Symbolic dynamic programming for first-order MDP's," in *Proc. of IJCAI'01*, 2001, pp. 690–697.
[24] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier, "Spudd: Stochastic planning using decision diagrams," in *Proc. UAI-99*, 1999.