Advanced Robotics, Vol. , No. , pp. 1–18 (2007)
© VSP and Robotics Society of Japan 2007.
Also available online - www.brill.nl/ar

Full paper

Learning to transfer optimal navigation policies

KRISTIAN KERSTING [1], CHRISTIAN PLAGEMANN [2,*],
ALEXANDRU COCORA [1], WOLFRAM BURGARD [2] and LUC DE RAEDT [1]

[1] *Machine Learning Laboratory and*
[2] *Autonomous Intelligent Systems Group, Institute for Computer Science, University of Freiburg, Georges-Koehler-Allee, Building 079, 79110 Freiburg, Germany*

Received 22 December 2006; accepted 11 April 2007

Abstract—Autonomous agents that act in the real world utilizing sensory input greatly rely on the ability to plan their actions and to transfer these skills across tasks. The majority of path-planning approaches for mobile robots, however, solve the current navigation problem from scratch given the current and goal configuration of the robot. Consequently, these approaches yield highly efficient plans for the specific situation, but the computed policies typically do not transfer to other, similar tasks. In this paper, we propose to apply techniques from statistical relational learning to the path-planning problem. More precisely, 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 imitate navigation behavior shown by users or by optimal policies. Second, it yields comprehensible models of behavior. Finally, a navigation policy learned in one environment naturally transfers to unknown environments. In several experiments with real robots and in simulated runs, we demonstrate that our approach yields efficient navigation plans. We show that our system is robust against observation noise and can outperform hand-crafted policies.

Keywords: Statistical relational learning; mobile robotics; navigation; imitation; path planning.

1. INTRODUCTION

For various tasks such as delivery, guidance, rescue, etc., mobile service robots need to plan their actions. In the past, the vast majority of approaches for computing navigation paths have dealt with solving the given navigation problem that the robot faces during operation from scratch. Whereas such approaches yield highly efficient paths for a given scenario [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 for users to understand and validate the navigation behavior. In this paper, we consider the problem of learning abstract

*To whom correspondence should be addressed. E-mail: plagemann@informatik.uni-freiburg.de navigation plans for mobile robots based on a set of trajectories in the configuration space of the robot. 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 broadly 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 large hotel without knowing the map of the building (Fig. 1). Different algorithms including depth-first search and uninformed LRTA* (see Ref. [6] for a comprehensive comparison) have been proposed for such problems. 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

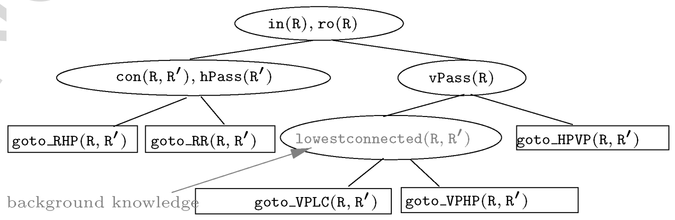


Figure 1. A relational decision tree representing a relational navigation policy for the hotel world.

can be assigned to each position in the configuration space of the robot. Such labels can be obtained during operation by analyzing sensor measurements and their temporal evolution, see [11, 12]. The observed labels are used to form (relational) state descriptions of a relational MDP (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.

The paper is organized as follows. After discussing the research context of our approach, we 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 mobile robot. Section 5 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.

2. THE RELATIONAL APPROACH

In the past few years, relational representations in machine learning and artificial intelligence have received a lot of attention (e.g. Ref. [13]) and is generally known under the name statistical relational learning (SRL). Relational reinforcement learning (RRL) is the sub-area of SRL concerned with learning what to do, i.e. how to map situations to actions so as to maximize a numerical reward signal. In contrast to most forms of S(R)L, the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them [14]. Given the success of SRL in general, it now seems to be an appropriate time to apply SRL, respectively RRL, techniques within robotics. The advantages of the relational approach are 3-fold. First, the learned navigation preferences can be directly transfered to alternative instances of the same 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 even in 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 (*cf.* Ref. [15]). Our experiments show that also in such cases our approach can be beneficial.

3. MDPs 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 Fig. 2. 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 $\mathtt{r}_4$.

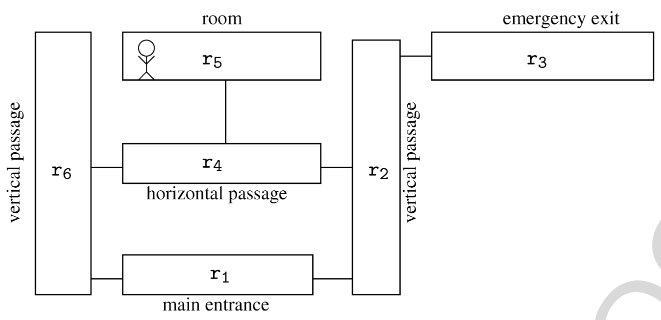


Figure 2. A map of hotel h. The robot in room $\mathtt{r}_5$ is supposed to find its way to the main entrance $\mathtt{r}_1$ of the hotel.

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 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 $\mathtt{goto}(\mathtt{r}_5, \mathtt{r}_4)$, $T : S \times A \times S \to [0, 1]$ is a transition model and $R : S \times A \times S \to [0, 1]$ is 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 $a \in A(s)$ is to be 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 $\gamma \in [0, 1]$, the state value function $V^\pi : S \to \mathbb{R}$ represents the value of being in a state following policy π, with respect to expected rewards. A policy π^* is optimal if $V^{\pi^*}(s) \geqslant V^{\pi'}(s)$ $\forall s \in S$ 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 $V_{t+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_t(s')]$ to get the next $(t = 1, 2, 3, \ldots)$. VI is guaranteed to converge in the limit towards V^*. From any value function V, we can compute a policy $\pi(V)$ that selects at any state s the greedy look-ahead action $\arg\max_{a\in A(s)} \sum_s T(s, a, s')[R(s, a, s') + \gamma V(s)]$. The policy $\pi(V^*)$ is an optimal policy.

Traditional MDPs and VI 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. For instance, propositional policies for the hotel in Fig. 2 cannot directly be applied in other hotels as each room is treated as a different proposition. As a consequence, it is hard to generalize policies across domains with similar properties.

4. LEARNING RELATIONAL NAVIGATION POLICIES

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

4.1. Relational logic

The hotel world can elegantly be represented using relational logic. Reconsider hotel h in Fig. 2: 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, $\mathtt{hPass}(\mathtt{r}_4)$, 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 $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_m)$, where a relation symbol $\mathtt{p}$ followed by a bracketed m-tuple of terms $\mathtt{t}_i$, 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, for example, 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 $\{\mathtt{X}_1/\mathtt{t}_1, \ldots, \mathtt{X}_n/\mathtt{t}_n\}$ where $\mathtt{X}_i$ are variables and all $\mathtt{t}_i$ 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 $B\theta \subseteq A$. For instance, $\mathtt{in}(\mathtt{R})$ subsumes $\mathtt{in}(\mathtt{R}), \mathtt{con}(\mathtt{R}, \mathtt{R}')$.

4.2. RMDPs

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 Ref. [16].

An abstract action 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. (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.) 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:

$$\begin{gathered}\texttt{in}(\mathtt{R}'), \texttt{con}(\mathtt{R}, \mathtt{R}'), \texttt{hPass}(\mathtt{R}'), \texttt{ro}(\mathtt{R}), \texttt{ro}(\mathtt{R}') \\ \xleftarrow{0.9:\texttt{goto\_RHP}(\mathtt{R},\mathtt{R}')} \\ \texttt{in}(\mathtt{R}), \texttt{con}(\mathtt{R}, \mathtt{R}'), \texttt{hPass1}(\mathtt{R}'), \texttt{ro}(\mathtt{R}), \texttt{ro}(\mathtt{R}'),\end{gathered}$$

where we assume $\mathtt{R} \neq \mathtt{R}'$. The action 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 $\texttt{goto\_RHP}(\mathtt{r}, \mathtt{r}')$ will yield $z' \equiv \texttt{in}(\mathtt{r}_4), \texttt{con}(\mathtt{r}_4, \mathtt{r}_5), \texttt{hPass}(\mathtt{r}_4), \texttt{ro}(\mathtt{r}_4), \texttt{ro}(\mathtt{r}_5)$ 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, for example, $10 \leftarrow \texttt{in}(\mathtt{r}')$ and $0 \leftarrow \texttt{true}$. It assigns 0 to z but 10 to z'. Using $\texttt{true}$ 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 $\texttt{main}$. When $\texttt{main}$ 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. $\texttt{main}(\mathtt{R})$. In our example, z is not absorbing but z' is. In addition, integrity constraints can be employed to exclude impossible states (*cf.* Ref. [16]).

4.3. 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 [16], 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:

$$\begin{aligned}&\texttt{goto\_RHP}(\mathtt{r}_5, \mathtt{r}_4) \\ &\quad \leftarrow \texttt{in}(\mathtt{r}_4), \texttt{hPass}(\mathtt{r}_4), \texttt{ro}(\mathtt{r}_4), \texttt{con}(\mathtt{r}_4, \mathtt{r}_5), \texttt{ro}(\mathtt{r}_5), \mathtt{r}_4 \neq \mathtt{r}_5, \qquad (1)\end{aligned}$$

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:

$$\begin{aligned}&\mathtt{goto\_RHP}(\mathtt{R}, \mathtt{R}')\\ &\quad\leftarrow \mathtt{in}(\mathtt{R}), \mathtt{con}(\mathtt{R}, \mathtt{R}'), \mathtt{hPass}(\mathtt{R}'), \mathtt{ro}(\mathtt{R}), \mathtt{ro}(\mathtt{R}'), \mathtt{R} \neq \mathtt{R}',\end{aligned} \tag{2}$$

generalizes (1) over arbitrary rooms $\mathtt{R}$ and $\mathtt{R}'$.

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 (e.g. if they were obtained by computing the optimal policy for a fully known map of the environment) or not (e.g. 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 in which the model is known, the second one corresponds to a model-free case and also allows the robot to learn from imitation or to perform what is called behavioral cloning. The key idea is that each trace t_i describes a situation–action sequence, e.g. for leaving a hotel. More precisely, each t_i consists of ground navigation rules such as (1). Each rule describes an interpretation, $\mathtt{in}(\mathtt{r}_5), \mathtt{con}(\mathtt{r}_4, \mathtt{r}_5), \mathtt{hPass}(\mathtt{r}_4), \mathtt{ro}(\mathtt{r}_4), \mathtt{ro}(\mathtt{r}_5), \mathtt{r}_4 \neq \mathtt{r}_5$, 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. office, horizontal passage, vertical passage, elevator, etc.—in order to take the associated optimal actions $\mathtt{goto\_RHP}(\mathtt{r}_5, \mathtt{r}_4)$. 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 [17] where relational programs are induced from interpretation–class pairs. One standard approach to employ during generalization are relational decision trees.

A relational decision tree [18] (see Fig. 1) is a binary decision tree in which each node contains a conjunction such as $\mathtt{in}(\mathtt{R}), \mathtt{ro}(\mathtt{R})$. 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 $\mathtt{con}(\mathtt{R}, \mathtt{R}'), \mathtt{hPass}(\mathtt{R}')$. 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, e.g. $\mathtt{in}(\mathtt{R}), \mathtt{ro}(\mathtt{R}), \mathtt{con}(\mathtt{R}, \mathtt{R}')$, $\mathtt{hPass}(\mathtt{R}')$. 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 (2) in its left-most path and also suggests to take action $\mathtt{goto\_RHP}(\mathtt{r}_5, \mathtt{r}_4)$ in state z. To induce the tree, we essentially employ Quinlan's well-known C4.5 [19] scheme with the information gain as splitting criterion (for more details, see Ref. [18]).

Indeed, our overall approach of inducing a relational decision tree from situation-action pairs is akin to explanation-based learning (EBL) [20, 21], 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 [22] to learn general control rules from specific examples of problem solving episodes. Later, Dietterich and Flann [23] combined this idea with reinforcement learning by associating these generalized state descriptions with values obtained from value iteration. Subsequently, Boutilier *et al.* [24] and Kersting *et al.* [16] 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 further and further 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 in early EBL systems and has been called the utility problem [23]. To avoid this problem, our approach works directly with state–action pairs and inductively generalizes them into relational policy trees. This has the additional advantage that, in contrast to EBL, no model is required, which allows us to apply our techniques onto behavioral cloning. Finally, we would like to stress that EBL is a deductive generalization technique whereas our approach performs the generalization inductively.

To summarize, our approach works as follows:

(i) Observe a number of successful ground state–action sequences.

(ii) Induce a relational navigation policy in the form of a relational decision tree from this experience.

The resulting abstract navigation policy typically uses local information only, i.e. the environment does not need to be completely known. One particularly interesting case, on which we will focus in the experiments, is concerned with learning from optimal state–action sequences. These can 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 an MDP solver. We then generalize over several of such optimal state–action sequences computed for different problem instances using the TILDE algorithm to construct one relational decision tree. Note that we do not need the abstract formulation of the MDP in this learning step, but just the concrete states–actions pairs by the solver. Note also that the algorithmic properties of this approach are directly inherited from the specific MDP solver [25] and (relational) decision tree learner [19, 18] used. In the experiments reported on in the following section, we took the freedom to assume the raw sensor output to be relationally interpreted in terms of a set of predefined symbols and relations. In this way we avoided possibly overly rich representations that are computationally too expensive to be handled, and at the same time ensured enough flexibility and expressiveness to be able to solve the individual problems.

5. 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 in previously unknown environments.

5.1. Implementation details

In our current system, we use the system SPUDD [25] to solve the MDPs and to generate example navigation plans. To execute the navigation plans on the robot and in simulation runs, we used the Robot Navigation Toolkit CARMEN [26]. 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].

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 necessarily deterministic, the system may need to choose among several equally likely actions. Consider being in room $\mathtt{r}_4$ such as in state z'. Both neighboring rooms $\mathtt{r}_6$ and $\mathtt{r}_2$ are vertical passages leading to $\mathtt{r}_5$. Thus, the policy equally likely evaluates to $\mathtt{goto\_VPLC}(\mathtt{r}_4, \mathtt{r}_6)$ and $\mathtt{goto\_VPLC}(\mathtt{r}_4, \mathtt{r}_2)$. 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 chooses that state in *AltS* 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*.

5.2. Navigation in an office environment

The first experiments were 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 top of Fig. 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 floor plans of two other buildings. The actual map of the environment was unknown to the robot and only the labels of neighboring rooms could be observed.

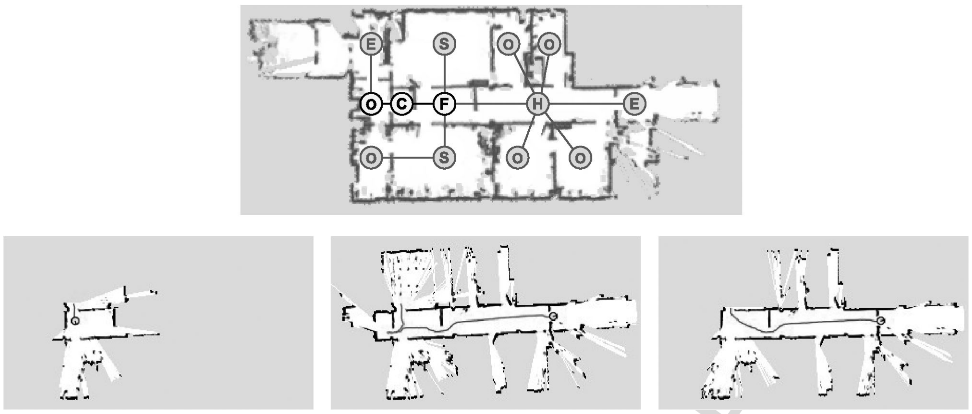


Figure 3. (Top) Map of the real office environment in which the experiments with our robot were carried out. When the robot was in room ‘C’, only three node labels were observable (black). All other labels as well as the overall topology of the environment were unknown (gray). (Bottom) Application of an abstract policy for finding the entrance hall of a building. The abstract policy was learned from different real buildings. (Left) The robot first leaves the room and enters the corridor. (Middle) Then it has to sample randomly and moves to the corridor to the left. (Right) At the same step in a different experiment, it randomly chose the right-hand side corridor directly.

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 shown in the leftmost image on the bottom of Fig. 3. At this point, the navigation policy outputs two equally likely alternatives: corridor C and hallway H. In the first experiment, it chose (randomly) 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 on the bottom of Fig. 3. The rightmost image shows the resulting trajectory in case the optimal corridor H is sampled directly when the robot is in F.

5.3. Simulation experiments

To quantitatively evaluate the performance of our approach, we compared it with the optimal paths as well as with uninformed LRTA* with maximum lookahead [5], which is a popular real-time search method for robot navigation in unknown terrain. In all experiments, the robot received a reward of 1.0 when reaching the goal, otherwise it received 0.0 as reward.

5.3.1. Learning to find an unknown target location. We ran several simulation experiments on maps of real buildings such as the ones depicted in Fig. 4. The

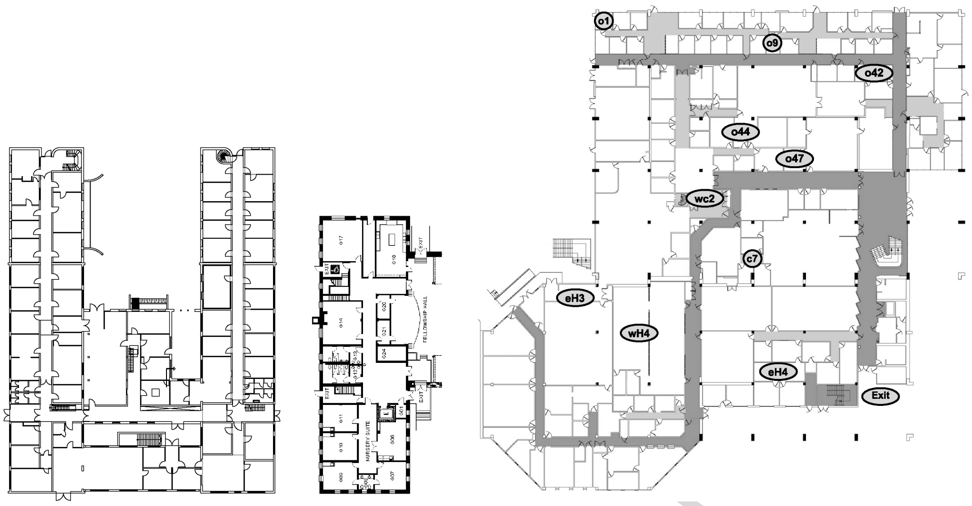


Figure 4. Maps used for the second set of experiments. The two left maps were used for learning and the right one for testing.

task for the robot was to find the previously unknown location of the exit. From the outlines of these buildings we manually generated an annotated topological map which then was used for calculating paths. To evaluate the performance, we randomly chose the starting locations. These starting locations are indicated by yellow/gray labels in Fig. 4. 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 LRTA* 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. Moreover, 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.

5.3.2. Learning to find an unknown person. In a further experiment, we applied our approach to the problem of finding persons. We modeled the situation for three research groups of the University of Freiburg including the maps of two buildings, (36 rooms) and 24 persons with their functions (professor, researcher, student and secretary). In the learning step, we computed the optimal paths from a set of starting locations to all persons working in the modeled department by solving the MDPs respectively. From these traces we learned a decision tree using the algorithm described in the previous sections. The induced decision trees (an example for a complete decision tree induced is shown in Fig. 5) encoded navigation rules such

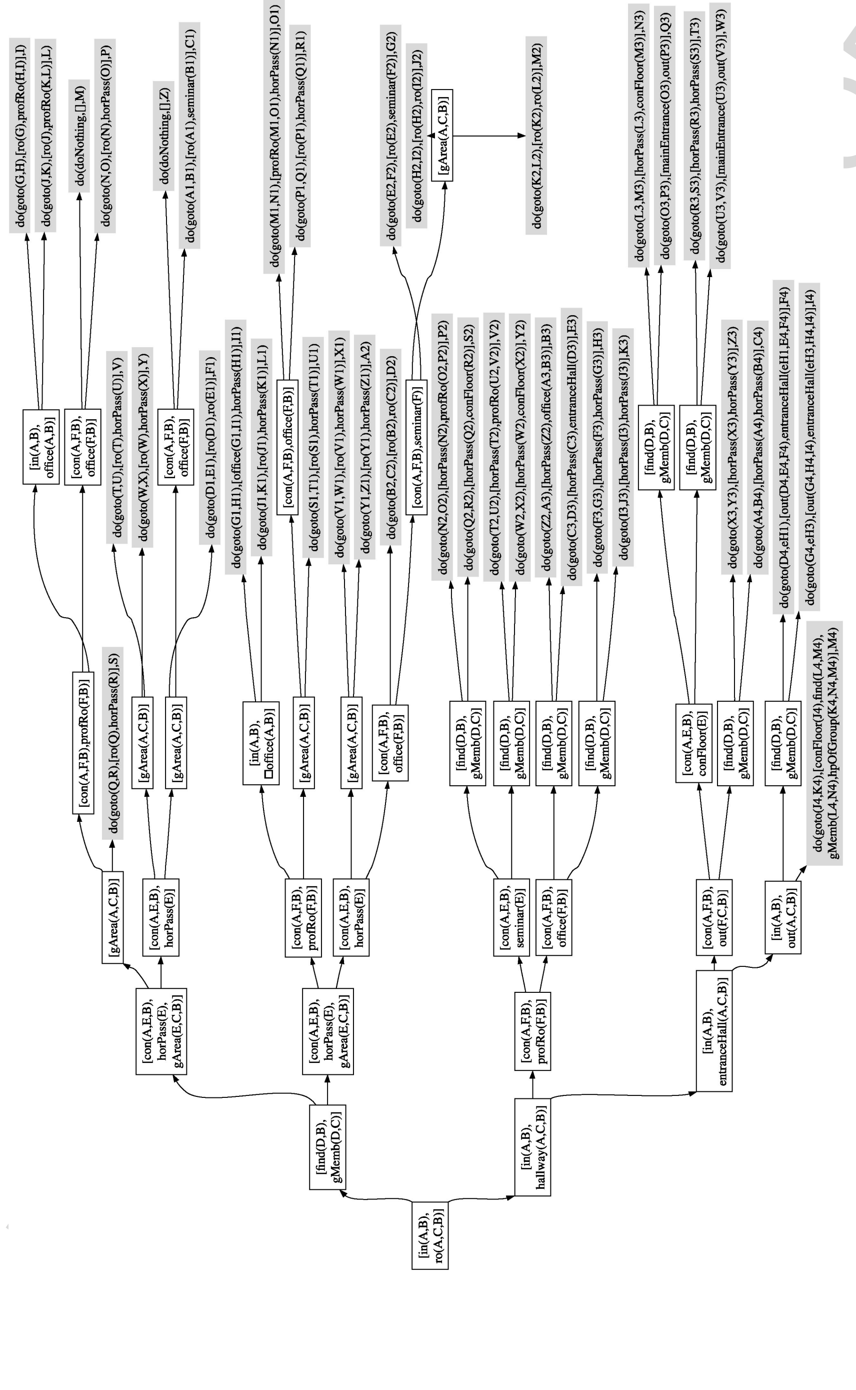


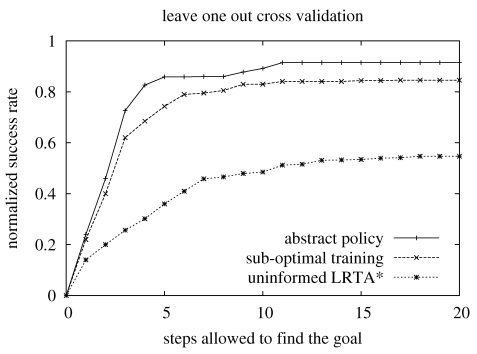
Figure 5. A relational policy induced over the ‘find an unknown person’ domain. White nodes are decisions and grey nodes are actions to be carried out. An action node `do(A, Pre)` means ‘perform `A` with the precondition `Pre` to bound variables’. Note that symbols are abbreviated due to size restrictions.

as:

```
goto(Room, Pass) ← find(Person), groupMember(Person, Group),
                   room(Room), in(Room),
                   not(areaOfGroup(Room, Group))
                   horizontalPass(Pass),
                   connected(Room, Pass),
                   areaOfGroup(Pass, Group).
```

This can be readily interpreted. For example, the above navigation rule can be written in English as follows: *If the robot is looking for a member of a research group and is in a room, which does not belong to the research group, but there is a horizontal passage in the area of the research group, which is connected to the current room, then the robot should enter that horizontal passage.* This is indeed a general and useful navigation rule, which particularly takes the research group of the sought-after person into account. This information is crucial in the search for persons as it limits the search space significantly. To test the generalization performance of the induced abstract policies, we introduced so far unknown visiting researchers labeled with a research group and function and evaluated the decision tree for the same set of starting locations searching for the new visiting researchers. The averaged path length in the evaluation step was 5.93 ± 2.56, only 1.05 ± 1.04 steps longer than the optimal paths (on average 4.85 ± 2.33). Finally, the abstract policy reached the goal in 0.83 ± 0.18 of the runs.

5.3.3. 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 experiment described here has been carried out to analyze the degradation of the performance in 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 then evaluated on the one left out. We performed five restarts and started randomly in one of the areas. Figure 6 shows 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 ith 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



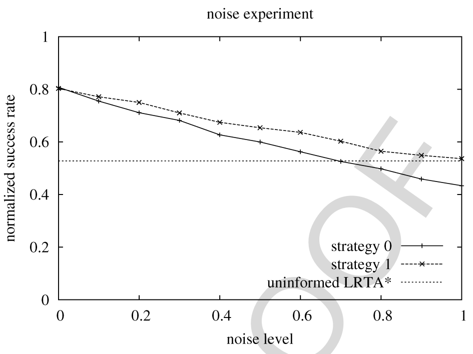


Figure 6. Normalized success rates (see text) for different maximal steps allowed to reach the goal (left). Normalized success rates for a varying level of noise in the observed labels (right).

almost 1.8 times longer than the optimal ones, whereas those generated by our algorithm showed only 25% overhead.

5.3.4. Comparison to manually crafted policies. Furthermore, we compared abstract navigation policies induced from optimal traces using our method to human generated policies. More precisely, we asked four students not related to the project to devise abstract navigation policies for the ‘find an unknown person’ task. We then encoded the manually crafted policies as relational decision trees. Overall, the manually crafted navigation policies were indeed much smaller and hence more comprehensible than our induced ones. To evaluated the performance of the policies, we searched for a new person starting in each state 10 times and allowing at most 50 steps.

The manually crafted policies achieved normalized success rates of 0.56, 0.61, 0.57 and 0.71 whereas the automatically induced one, denoted as A, achieved 0.82. To gain more insights, we analyzed the best manually crafted policy, denoted as B, in detail. Whereas 52% of the paths generated by A were optimal, B yielded optimal paths in only 27% of the cases. The median of additional steps needed with respect to the optimal paths are 0 for A and 6 for B, whereas the averages are almost identical, i.e. 9.34 for A and 9.89 for B. Thus, manually crafted policies achieve lower performances. The reason seems to be that humans tend to ignore special situations a robot can encounter and rather concentrate on more obvious, standard problem instances.

5.3.5. Observation noise. A robot’s perception of the world is never perfect. We implemented two strategies for dealing with situations in which the belief about place labels (office, horizontal passage, etc.) has to be revised. Strategy 0 always returns to the previous place when an inconsistent place label was detected, whereas Strategy 1 stays in the new room, updates the faulty label information, and continues navigating from there. The right diagram of Fig. 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 faulty place label is observed. It can be seen 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.

5.3.6. Manipulation task. Finally, we investigate our approach in a simulated manipulation task, which are often fetch-and-carry tasks. As an example, imagine a robot laying a table. At formal occasions like weddings or conference banquets, place names might be used. In contrast, a family meal such as lunch or dinner represents a daily occurrence and the table is obviously set in a simpler way. There are, however, some rules that are common to all. For instance, plates are usually equidistant and also the distances between the cutlery and the plate are identical for every person.

The 'lay the table' task is easy for humans, but rather difficult for robots. This is mainly due to the explosion in the number of possible states. To see this, consider a simplified version of the 'lay the table' domain, i.e. the block world [27], which we considered in our experiments. The block world has been a standard example in the literature on planning over the last decades and has extensively been used as a benchmark domain. The robot must learn to use its hand to remove blocks from blocks `a` and `b` so that block `a` may be put on `b`. Only when `a` is on top of `b`, the robot receives a positive reward. Such tasks are challenging for propositional approaches because the number of possible states can easily explode: for four blocks there are 73 states, for seven blocks there 37, are 663 states and for 10 blocks there are already 58 941 091 states. In propositional approaches such as traditional MDPs and their solution methods each state must, in principle, be represented using a separated proposition. Consequently, they cannot naturally capture the structure of the underlying class of problems and the learned policies do not easily generalize across domains with similar properties.

The goal was to solve $\texttt{on}(\texttt{A},\texttt{B}), \texttt{A} \neq \texttt{B}$, i.e. to put a block `A` on a block `B` for any blocks `A` and `B`. The `move` action was encode as in Ref. [16] with two possible outcomes: success with probability 0.9 and failure with probability 0.1. We computed the optimal policies for three and four blocks, and induced an abstract policy from them, which is shown in Fig. 7. The policy is indeed optimal as it removes all blocks on top of `A` and `B`, and then moves `A` on top of `B`. More importantly, it neither specifies the number of blocks nor the particular goal blocks. Thus, it is an optimal policy for any number of blocks and, hence, all blocks world situations.

This is an interesting result. So far, many RRL approaches have had problems with this task. For instance, Dzeroski *et al.* [9] report on experiments where the policy induced succeeds in about 90% of the cases. Kersting *et al.* [16] report on an automatically computed relational value function optimal for up to 10 blocks.

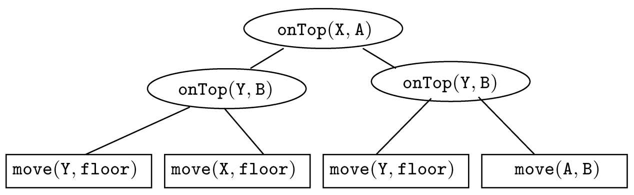


Figure 7. The relational policy induced to solve the blocks world problem on(A, B), A $\neq$ B. The predicate onTop(U, V) denotes that there is some block U on top of V. Its definition was provided in the background knowledge. The constant floor denotes the floor.

6. CONCLUSIONS

We presented 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 advantage of our approach is that relational abstraction allows one to generalize from previously planned paths and to transfer policies across tasks in even previously unseen environments.

Our algorithm has been evaluated in experiments with real robots as well as in simulation runs. To the best of our knowledge, this is the first application of R(R)L within robotics and on a real robot. The results demonstrate that the learned abstract policies are highly efficient and outperform uninformed LRTA* with maximum lookahead. Furthermore, they show that our approach can even imitate trajectories from sketched examples provided by users and solve manipulation tasks.

An interesting question for future research will be how many training instances—either optimally solved MDPs or observed state-action sequences—are required for successfully learning policies. As our experiments indicate, the relational approach seems to be able to learn from relatively few examples.

Acknowledgments

This work was partially supported by the European Union IST programme, contract no FP6-508861, Applications of Probabilistic Logic Programming 2, by the EC under contract number FP6-004250-CoSy, and by the German Federal Ministry of Education and Research (BMBF) under contract number 01IMEO1F (project DESIRE).

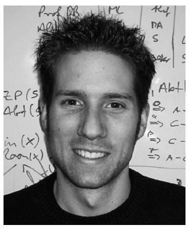
REFERENCES

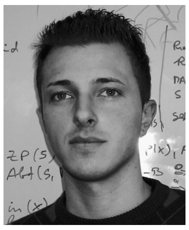
1. H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA (2005).
2. S. LaValle, *Planning Algorithms*. Cambridge University Press, Cambridge (2006).
3. J. Latombe, *Robot Motion Planning*. Kluwer, Dordrecht (1991).

4. A. Stentz, The focused D* algorithm for real-time replanning, in: *Proc. IJCAI*, Montreal, pp. **00–00** (1995).
5. R. Korf, Real-time heuristic search, *Artificial Intell.* **42**, 189–211 (1990).
6. S. Koenig, Agent-centered search, *AI Mag.* **22**, 109–132 (2001).
7. D. Bellman, *Dynamic Programming*. Princeton University Press, Princeton, NJ (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*, pp. 91–122. MIT Press, Cambridge, MA (1998).
9. S. Džeroski, L. De Raedt and K. Driessens, Relational reinforcement learning, *Machine Learn.* **43**, 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*, Trento, pp. **00–00** (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. AAAI*, Pittsburgh, PA, pp. 1306–1311 (2005).
12. O. M. Mozos, C. Stachniss and W. Burgard, Supervised learning of places from range data using Adaboost, in: *Proc. ICRA*, Barcelona, pp. **00–00** (2005).
13. L. De Raedt, T. Dietterich, L. Getoor and S. Muggleton (Eds), *Working Notes of the Dagstuhl-Seminar* 5051 *on Probabilistic, Logical and Relational Learning—Towards a Synthesis* (2005).
14. R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA (1998).
15. C. Sammut, S. Hurst, D. Kedzier and D. Michie, Learning to fly, in: *Proc. ICML*, Aberdeen, pp. **00–00** (1992).
16. K. Kersting, M. Van Otterlo and L. De Raedt, Bellman goes relational, in: *Proc. ICML*, Banff, pp. 465–472 (2004).
17. S. Muggleton and L. De Raedt, Inductive logic programming: Theory and methods, *J. Logic Progr.* **19**, 629–679 (1994).
18. H. Blockeel and L. De Raedt, Top-down induction of first order logical decision trees, *Artificial Intell.* **101**, 285–297 (1998).
19. J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA (1993).
20. T. Ellman, Explanation-based learning: a survey of programs and perspectives, *ACM Comp. Surv.* **21**, 163–221 (1989).
21. M. Lent and J. Laird, Learning procedural knowledge through observation, in: *Proc. 1st Int. Conf. on Knowledge Capture*, Victoria, BC, pp. 179–186 (2001).
22. S. Minton, J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni and Y. Gil, Explanation-based learning: a problem solving perspective, *Artificial Intell.* **40**, 63–118 (1989).
23. T. G. Dietterich and N. S. Flann, Explanation-based learning and reinforcement learning: a unified view, *Machine Learn.* **28**, 169–210 (1997).
24. C. Boutilier, R. Reiter and B. Price, Symbolic dynamic programming for first-order MDP's, in: *Proc. IJCAI'01*, San Francisco, CA, pp. 690–697 (2001).
25. J. Hoey, R. St-Aubin, A. Hu and C. Boutilier, Spudd: stochastic planning using decision diagrams, in: *Proc. UAI*, San Francisco, CA, pp. **00–00** (1999).
26. M. Montemerlo, N. Roy and S. Thrun, Perspectives on standardization in mobile robot programming, in: *Proc. IROS*, Las Vegas, NV, pp. **00–00** (2003).
27. J. Slaney and S. Thiébaux, Blocks world revisited, *Artificial Intell. J.* **125**, 119–153 (2001).

ABOUT THE AUTHORS

Kristian Kersting is now a Postdoctoral Associate at MIT CSAIL, Cambridge, USA. Before joining MIT he was a member of the Machine Learning group at the Albert-Ludwigs University Freiburg, Germany, where he also received his PhD (Dr rer. nat.) in Computer Science in 2006. He has done considerable work in developing Bayesian logic programs and logical hidden Markov models, and on the theory of relational reinforcement learning. In 2006, he has received the ECML-06 Best Student Paper Award.


Christian Plagemann has studied Computer Science at the University of Karlsruhe, Germany. Since 2005 he has been a Research Assistant at the Department of Computer Science of the University of Freiburg, Germany. His work within the research group for Autonomous Intelligent Systems is focused on artificial intelligence, non-parametric learning methods and relational learning in robotics.


Alexandru Cocora received his diploma in Computer Science in 2005 at the University of Freiburg, Germany. Until 2006 he worked as a Research Assistant within the research group for Autonomous Intelligent Systems, where his research was focused on relational robotics. Since 2006 he has been working as a Software Engineer at Avaloq Evolution in Zurich.

Wolfram Burgard received his PhD degree from the Department of Computer Science of the University of Bonn, Germany, in 1991. Since 1999, he has been is Associate Professor at the University of Freiburg where he heads the research group for Autonomous Intelligent Systems. His areas of interest lie in artificial intelligence and mobile robots. He has published two books and over 100 articles in outstanding journals and conference proceedings. He is an active member of the IEEE Robotics and Automation Society as well as the American Association of Artificial Intelligence.

Luc De Raedt received his PhD in Computer Science (Dr Informatica) from the Katholieke Universiteit Leuven, Belgium in 1991, where is now a Research Professor in the Department of Computer Science. From 1999 till 2006, he was a Professor at the University of Freiburg and Head of the Machine Learning and Natural Language Processing Lab research group. His research interests are in inductive logic programming, machine learning and data mining, as well as their applications. In 2005, he was elected as an ECCAI fellow.