Scaling Up Reinforcement Learning with a Relational Representation

Eduardo F. Morales

Computer Science and Engineering**
University of New South Wales, UNSW, Sydney 2051, Australia
emorales@cse.unsw.edu.au

Abstract. Reinforcement learning has been repeatedly suggested as good candidate for learning in robotics. However, the large search spaces normally occurring robotics and expensive training experiences required by reinforcement learning algorithms has hampered its applicability. This paper introduces a new approach for reinforcement learning based on a relational representation which: (i) can be applied over large search spaces, (ii) can incorporate domain knowledge, and (iii) can use previously learned policies on different, although similar, problems. In the proposed framework states are represented as sets of first order relations, actions in terms of those relations, and policies are learned over such *generalized* representation. It is shown how this representation can capture large search spaces with a relatively small set of actions and states, and that policies learned over this *generalized* representation can be directly apply to other problems which can be characterized by the same set of relations.

1 Introduction

Robots are becoming part of our every day life. Their full incorporation requires an increased flexibility to learn and adapt from their interaction with their environment. One of the most active research areas in artificial intelligence devoted to learning through interaction with the environment is reinforcement learning [4, 11]. Reinforcement learning (RL) is about learning how to map situations to actions so as to maximize a numerical reward. The learner is not told what actions to take and must discover, by trial and error, the actions that produce the greatest reward.

Despite recent advances, there are still not completely satisfactory solutions for dealing with large search spaces, easily incorporating domain knowledge, and transferring previously learned policies to other related problems. This is particularly relevant to robotics where large search spaces and expensive training experiences are normally needed.

In this paper, it is argued that a richer and more general representation can advance in the solutions of these problems and is indeed needed to tackle more challenging problems as those encountered in robotics.

** On sabbatical leave from ITESM-Cuernavaca, Temixco, Morelos, 62589, México.

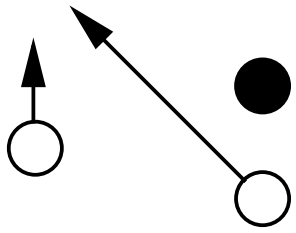

Fig. 1. Passing a ball forward.

Suppose we want to learn a policy to play soccer with robots. If the space is discretized into 10×10 cm., even for a field of 5×10 mt. and two player a side, there are roughly 6.25×10^{10} possible states (assuming each robot can occupy only a single square). On top of that, we need to consider all the possible actions for each robot, thus learning directly is this representation is just too slow. In this domain, however, there are many states which are essentially the same, in the sense that they all share the same set of relations. For a robot, it is not so much important its exact location but the relations that hold between the robot and its environment. For instance, whenever an opponent robot is in front of the robot with the ball, a robot of the same team is slightly forward to one side, and there are no other robots around, a good action is to pass the ball to a position in front of that robot (see figure 1). Assuming this action can be accurately perform, it is applicable to all positions where the previously mentioned relations hold between the robots regardless of their exact location. In fact, it is applicable to fields of different sizes. It should be clear that a relational representation can produce useful state abstractions. In this paper it is shown how it can be used to produce useful policies when several possible actions are applicable to a particular state, and that the learned policies can be directly transfer to other, although similar, problems where the same relations apply.

This paper is organized as follows. Section 2 describes in more detail the proposed relational representation. Section 3 describes how to perform reinforcement learning over this representation. Section 4 provides experimental results, while section 5 analyses the perspectives of the approach for robotics. Finally, section 6 concludes and suggests future research directions.

2 Relational Representation

The main idea behind this research is to represent states as sets of properties that can be used to characterize a particular state and which may be common to other states. *An* r-state *is a state described by a set of first-order relations.* Each state is an instance of one and only one *r-state*. With robots these relations could be *goal_in_front*, *team_robot_to_the_left*, *opponent_robot_with_ball*, etc. An *r-state* can cover a large number of states (i.e., all the states where the relations described in the *r-state* hold). In this paper it is assumed that the relations are

previously defined by the user (other approaches to learn such relations can be seen in [6, 7]).

Once a set of relations has been defined, the search space in the relational space is completely defined. Considering that relations can also occur in their negated form, for N relations there are potentially 2^N *r-states*. Fortunately, not all the combinations of the relations are possible, and in general huge reductions in the number of states and actions can be achieved.

The set of actions also use a first-order relational representation. Rather than trying to apply all the available actions, we want to apply only *relevant* actions, i.e., those which apply when particular relations in the state description hold and which, possibly, enforce other relations to hold. *An* r-action *is defined by a set of pre-conditions, a* generalized *action, and possibly a set of post-conditions.*

The pre-conditions are conjunctions of relations that need to hold for the *r-action* to be applicable, and the post-conditions are conjunctions of relations that need to hold after a particular primitive action. The *generalized* action represents all the instantiations of primitive actions which satisfy the conditions. When several primitive actions satisfy the conditions of an *r-action*, one of them is chosen randomly. An *r-action* can be applied to many states, and as expected, not all the *r-actions* apply to all the *r-states*.

For an *r-action* to be properly defined, it must satisfied the following condition: *If an* r-action *is applicable to a particular instance of an* r-state*, then it should be applicable to all the instances of that* r-state.

3 Reinforcement Learning on *R-Space*

For any Markov decision process, the objective it to find the optimal policy, i.e., one which achieves the highest cumulative reward among all policies. The main purpose to learn in an *r-space* is to reduce the size of the search space, and take all the advantages of a richer representation language. However, in the *r-space* there is no guarantee that the defined *r-actions* are adequate to find an optimal sequence of primitive actions and sub-optimal policies can be produced. We can however, defined optimality in terms of an *r-space*.

A policy consistent with our representation, which we will refer to as an *r-space* policy (π_R), is a scheme for deciding which *r-action* to select when entering an *r-state*. An *r-space* optimal policy (π_R^*) is a policy that achieves the highest cumulative reward among all *r-space* policies.

The expected reward, in this case, is the expected *average* reward over all the instances of the *r-state*. When several *r-actions* are applicable to a particular *r-state*, the best policy will prefer the *r-actions* which lead to an *r-state* with the best expected average reward. This process is in general *non* Markovian, as the same *r-action* in the same *r-state* may take the agent to different *r-states*. Nevertheless, we can show convergence to an optimal *r-space* policy.

3.1 The rQ-learning Algorithm

This paper focuses on applying Q-learning [12] in *r-space*, although a similar argument can be applied to other reinforcement learning algorithms. Table 1 gives the pseudo-code for the rQ-learning algorithm. This is very similar to the Q-learning algorithm, but the states and actions are characterized by relations. The algorithm still takes primitive actions (a's) and moves over primitive states (s's), but learns over *r-state*-*r-action* pairs.

Table 1. The rQ-learning algorithm.

Initialize $Q(S, A)$ arbitrarily
(where S is an *r-state* and A is an *r-action*)
Repeat (for each episode):
  Initialize s
  $S \leftarrow rels(s)$ % evaluates the set of relations over state s
  Repeat (for each step of episode):
    Choose A from S using a *persistently exciting* policy (e.g., ϵ-greedy)
    Randomly choose action a applicable in A
    Take action a, observe r, s'
    $S' \leftarrow rels(s')$
    $Q(S, A) \leftarrow Q(S, A) + \alpha(r + \gamma max_{A'} Q(S', A') - Q(S, A))$
    $S \leftarrow S'$
  until s is terminal

It can be shown, following [10], that rQ-learning converges to an optimal *r-state* policy. In order to guarantee convergence to the *r-space* policy, we need the agent to follow a stationary stochastic policy that assigns to each state a non-zero probability of executing every action in every state. This can be achieved with a *persistently exciting* policy, for instance, ϵ-greedy, and if the *r-actions* are defined in such a way that every possible primitive action can be considered in each state (this can easily be achieved by adding, if necessary, an *r-action* with no conditions which can perform any primitive action). We also need to satisfy $lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t = \infty$ and $lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t^2 < \infty$.

It will be shown that obtaining an optimal *r-space* policy is good enough to solve complex problems and that in many cases, it also corresponds to the optimal policy. Also, since we are learning over generalized actions and states, the same policy is applicable to other problems where the same set of relations hold, as long as the *r-actions* continue to apply to all the instances of the *r-states* where they are applicable and the problem can be characterized by the same set of relations.

The most closely related work is what has been called Relational Reinforcement Learning [3]. They also use a relational representation for states and actions, however their main focus has been on approximating value functions with

a relational representation. This extends previous work on incremental regression trees to a relational representation. There are two main differences with our approach: (i) we are not trying to approximate a value function with a relational representation, we believe that there are alternative methods better suited for that, and (ii) our representation of states is at a higher abstraction level (in terms of properties of states rather than atoms describing states). This means that we have less number of states, and state-action pairs, it is easier to learn a value function, and it is easier to re-use previously learned policies. On the other hand, it is more difficult for the user to adequately represent states and actions.

4 Experiments

In all the experiments, the Q values were initialized to zero, $\epsilon = 0.1, \lambda = 0.9$, and $\alpha = 0.1$. Let us start with a simple grid problem, where the idea is to go from a starting point to a particular destination. In this simple domain, we can think of two general types of movements, where the agent either moves closer to or further away from the goal. Knowing the current position of the agent and its intended destination, we can evaluate the Manhattan distance between these two points and construct two *r-actions* that will decrease or increase, respectively, this distance. The Manhattan distance can be part of the state description or can be a property obtained from the state description. A clear advantage of using a richer representation formalism is that it is easy to express and include spatial and temporal relations into the states and actions. It is always possible to decrease the Manhattan distance to the goal, unless the agent has already reached it, and to increase the distance to the goal, unless the agent is in a corner and the goal is not in the same row or column. We can then characterize this problem with two relations: (1) the agent is in its goal position (*in_goal_position*) and (2) the agent is in one corner and the goal is not in the same row or column (*in_corner*), and define the following two *r-actions* using Prolog notation:

```
r_action(1,State1,Move,State2) :-
    not in_goal_position(State1),
    distance_to_goal(State1,Dist1),
    move(State1,Move,State2),
    distance_to_goal(State2,Dist2),
    Dist1 > Dist2.
r_action(2,State1,Move,State2) :-
    not in_corner(State1),
    distance_to_goal(State1,Dist1),
    move(State1,Move,State2),
    distance_to_goal(State2,Dist2),
    Dist1 < Dist2.
```

In this case, rather than the usual four primitive actions (north, south, east, west), we have two generalized actions, which can get instantiated, for instance, to north–east and south–west, respectively, when the goal is in the upper right

side of the agent (see figure 2). They however, may be instantiated to different primitive actions depending on the current position of the agent and the goal.

With 2 relations, there are 4 possible *r-states*, however, the goal position is an absorbing state, so we will considered only 2 possible *r-states*: (i) *not in_goal_position* and *not in_corner* (where both *r-actions* are applicable) and (ii) *not in_goal_position* and *in_corner* (where only the first *r-action* is applicable). In this simple case, we have only 3 *r-state–r-action* pairs, and the only thing for the rQ-learning algorithm to decide is whether to prefer *r-action* 1 or 2 in one *r-state*, which is achieved after a very small set of interactions (see figure 3).

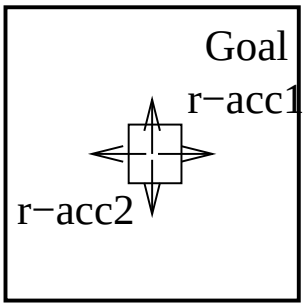

Fig. 2. *R-actions* in a Grid-World.

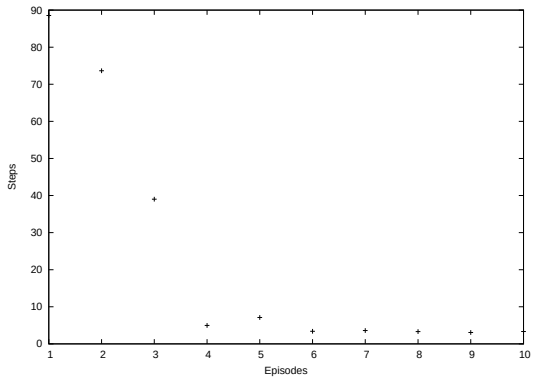

Fig. 3. Average number of steps (primitive actions) per trial for a simple 5×5 grid problem (over 35 trials).

Since the *r-states* and *r-actions* are expressed in terms of first order relations, on each trial the starting and goal position were randomly chosen, so rather than learning 25 policies for a 5×5 grid, we only need to learn one. Furthermore,

the learned policy can be directly applied to any rectangular shape grid problem (without internal walls) and from any starting and goal positions. So a policy learned on a 5×5 grid and can be used to navigate between any pair of points on any size grid problem. Figure 4 shows six paths followed on a 25×20 grid from randomly chosen starting and ending position using the learned 5×5 grid policy. In this simple case, the *r-space* policy is also an optimal policy.

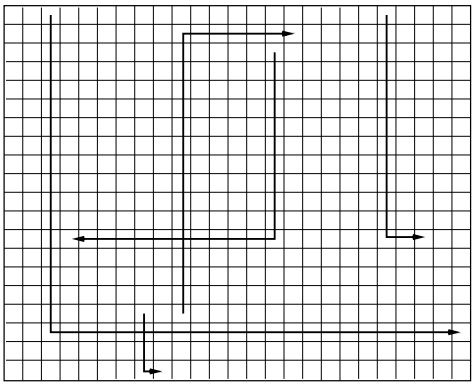

Fig. 4. Paths followed on a 25×20 grid using the policy learned on a 5×5 grid

4.1 The Taxi Domain

rQ-learning was tried in the taxi problem [2], where a taxi, starting at a random location navigates in a 5×5 grid world to pick up and put down a passenger (see figure 5). There are four possible source and destination locations called taxi ranks (R, Y, G, and B). The objective is to pick up a passenger in one taxi rank and put down the passenger at the destination rank. The source and destination ranks are chosen at random on each new trial.

We define 8 relations to characterize this domain, that express, whether the taxi is in a pick-up position, in a delivery position, with a passenger, next to a border, in a corner, in front of a goal, inside an area blocked by a border, and in the border of that area. We also defined 14 *r-actions* for staying in or moving out of blocking areas and for moving closer or further away from a particular goal rank, for picking-up and dropping a passenger, etc.

There are, however, only 32 *r-states* and only 80 *r-state-r-action* pairs (as opposed to 3,000 in the original formulation). They were used to learn an *r-state* optimal policy for the Taxi domain. Figure 6 shows the average number of steps per episode over 50 trials.

We tried to used the learned policy on a different Taxi problem, however, not all the possible *r-state-action* pairs were visited, since there are no blocking borders of height greater than 2, which was used on a particular *r-action*. To

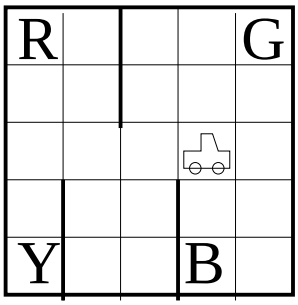

Fig. 5. The Taxi Domain.

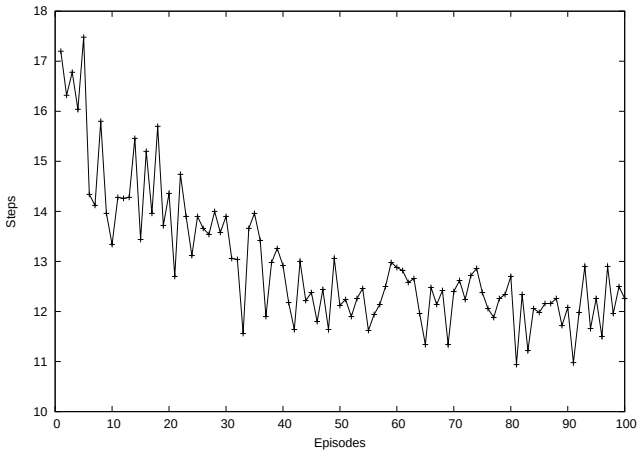

Fig. 6. Average number of steps (primitive actions) per trial (over 50 trials).

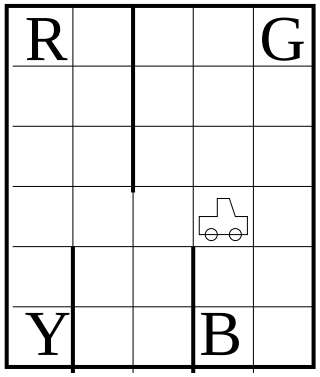

Fig. 7. Augmented Taxi Domain.

fix this, we add an extra row at the top of the taxi domain, and learn over this domain (see figure 7). After 500 episodes, we used this learned policy over a modified Taxi problem shown in figure 8, where vertical barriers were also introduced along with an extra taxi rank. As in the previous case, no extra learning is needed as this new problem is characterized by the same *r-states* and *r-actions*. Figure 8 shows the learned policy from figure 7 applied over two random instances of this new domain.

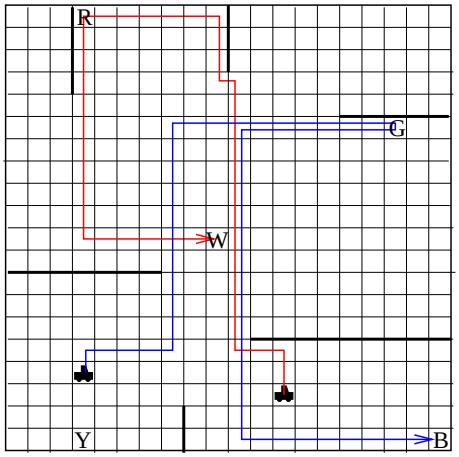

Fig. 8. Two paths followed on two instances of a different Taxi domain.

4.2 King-Rook vs. King

Now lets turn our attention to a more challenging problem where a relational representation is fundamental to efficiently learn a good policy. In the KRK domain, the goal is to check mate the opponent king from any legal starting position. First, we decided to manually construct a playing strategy for this endgame. To simplify things, the opponent king moves were randomly selected from its available legal moves. Our manually designed strategy, involves 27 *r-actions*, and triggers the first applicable *r-action*. This required a careful ordering of the *r-actions* in order to produce reasonable results. The average number of the winning-side moves of this strategy repeated three times over 100 random initial positions was 14.646. It plays reasonably well and in all our tests was always able to check-mate the opponent, however, it is not optimal and can get trapped into loops against a human player.

Upon examination, the 27 *r-actions* use 26 relations in their definition, and we decided to try them in our framework aiming for a better strategy. Although we have 26 relations and 27 *r-actions* there are only 1,318 *r-states* and only

2.67 *r-actions* on average per *r-state* (as opposed to ≈ 150,000 states and ≈ 22 primitive actions per state).

rQ-learning was given the same *r-actions* to try to find a better policy. After 5,000 games, the average number of the winning-side moves of the learned strategy over the same 100 random initial positions, repeating it three times was 12.07, which represents an improvement of over 2.5 moves in average.

Figure 9 shows the average number of winning-side moves per game over three runs. Points are plotted every 500 games and show the minimum, maximum and average number of winning-side moves per game.

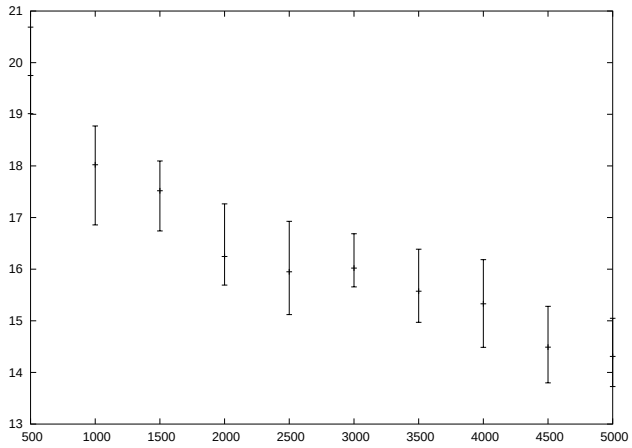

Fig. 9. Number of games vs. average number of moves per game.

5 Perspectives for Robotics

There are at least three reasons why the above formalism may be useful to robotics: (i) policies learned by one robot can, in principle, be transfered to another robot and/or to a similar domain, (ii) in many cases it is not so much important the exact position of the robot, but its relative position with respect to its environment and possibly other robots, and (iii) an abstraction based on relations can substantially reduce the training time.

There are, however, several research issues that need to be addressed. In particular: (i) handling noisy information, (ii) dealing with a continuous domain, and (iii) defining an adequate set of relations and *r-actions*.

The continuous space is not much of an issue, in general, as an action can be executed repeatedly as long as a relation holds. In the grid problems this was illustrated by moving always the agent closer to the goal (many discrete states were considered as the same *r-state*).

The main problem comes with noisy information and recognising when a particular relation is not longer valid. For instance, a relation may be used to define when an object is in front of the robot. This requires the definition of a particular threshold to specify what it means to be in front (e.g., facing towards the object plus/minus 5^o). This branches into two problems: (i) how to define adequate thresholds, and (ii) how to avoid switching continuously between actions when the robot is around a threshold value. Unfortunately, there are no easy solutions here, and most of the time it is handled by trial-and-error.

The other issue that needs to be addressed is the definition of adequate relations. Although some may be intuitive it is not necessarily an easy task.

The definition of adequate *r-action* can be done either by the user or by a behavioural cloning approach (e.g., [9]), where a human performs desirable actions from which *r-actions* are learned. This could be achieved by constructing rules considering the relations that hold before and after each action and generalizing constants consistently (e.g., see [6]).

There are other issues that also need to be addressed when multiple agents are involved. We will consider only two main approaches: one based on zero-sum games, where two agents have diametrically opposed goals, and the reward function is inverted. Under this setting, there is in effect a single reward function and a single Q value function $Q_1 = -Q_2$, which one agent tries to minimize, while the other tries to maximize [5]. The other approach is when we have n agents all searching for their maximum possible payoff. What is best for one agent is also best for the others, this is also called coordination equilibrium. In this case, we can define teams of agents with precisely the same goal. All agents have the same reward function which all agents try to maximize together. They are also called cooperative games [1]. Acting together, every reward received by one agent in a team is received by all agents, $Q_1 = Q_2 = \ldots = Q_n$, therefore only one Q-function needs to be learned.

Zero-sum strategies are useful in games where two agents have opposite goals, while team strategies are useful where a set of agents is seeking the same goal. In a domain like RoboCup we faced both situations. We believe that a relational representation allows to transfer more effectively policies.

6 Conclusions and future work

This work introduces a relational representation for reinforcement learning. It is based on properties of states, which allows to perform state aggregation. On the negative side, the resulting process is no longer Markovian and can produce sub-optimal policies. On the bright side, it is easy to incorporate domain knowledge, can be used in large application domains, and can re-use previously learn policies on other related problems which can be described with the same set of relations. In robotics, it is not always necessary to know an exact location of a robot but rather its relative position with other objects in the environment. This can be easily expressed with a relational representation.

As part of our future work, we would like to learn *r-actions* and relations from samples or traces (e.g., see [8]). We would also like to know whether a particular relational representation is well suited for the task or needs to be corrected, either when learning in a particular domain or when re-using previously learned policies in a related domain. Finally, we would like to apply our relational approach to robotics.

Acknowledgemente

I am grateful to Bernhard Herst, Claude Sammut and Will Uther for useful discussions and comments to this work. This research was supported by CONACyT grant 020121 and Tec de Monterrey - Cuernavaca, Mexico.

References

1. Boutilier, C.: Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge* (TARK-96), (1996).
2. Dietterich, T.: Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* **13** (2000) 227-303.
3. Džeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. *Machine Learning* **43** (2000) 7-52.
4. Kaelbling, L., Littman, M.L., and Moore, A.W.: Reinforcement learning: a survey. *Journal of Artificial Intelligence Research* **4** (1996) 237-285.
5. Littman, M.: Markov Games as a Framework for Multiagent Reinforcement Learning, In *Proc. 11th Int. Conf. on Machine Learning*, (1994) 157-163, New Brunswick, NJ, Morgan Kaufmann.
6. Morales, E.: Learning Playing Strategies in Chess. *Computational Intelligence* **12** (1996) 65–87.
7. Morales, E.: PAL: A pattern–based first–order inductive system. *Machine Learning* **26** (1997) 227–252.
8. Morales, E.: On learning how to play. *Advances in Computer Chess 8*, (1997) H.J. van den Herik y J.W.H.M. Uiterwijk (eds), Universiteit Maastricht, The Netherlands, pp. 235–250.
9. Sammut, C., Hurst, S., Kedzier, D., Michie D.: Learning to fly. In D. Sleeman and P. Edwards (eds.), *Proceedings of the Ninth International Workshop on Machine Learning*, (1992) 385-393, Morgan-Kaufmann.
10. Singh, S., Jaakkola, T., Jordan, M.: Reinforcement learning with soft state aggregation. In *Neural Information Processing Systems* 7 (1996). G. Tesauro, D. Touretzky and T. Leen (eds.) Cambridge, MA, MIT Press.
11. Sutton, R., Barto, A.: *Reinforcement Learning an Introduction*, MIT Press, Cambridge, MA, (1998).
12. Watkins, C.: *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, MA. (1989).