Appeared in:
International Conference on Artificial Neural Network,
pages 1133-1138, Skövde, Sweden, September 2-4, 1998.

Embedding Knowledge in Reinforcement Learning

G. Hailu, G. Sommer

Christian Albrechts University, Department of Cognitive Systems
Preusserstrasse 1-9, D-24105 Kiel, Germany
gha@informatik.uni-kiel.de

Abstract. In almost all real systems where reinforcement learning is applied, it is found that a knowledge free approach doesn't work. The basic RL algorithms must sufficiently be biased to achieve a satisfactory performance within a bounded time. This bias takes different forms. In this paper, in addition to reflex rules [6], environment (domain) knowledge is embedded into the learner. Environment knowledge gives leverage to the adaptive state space construction algorithm by splitting *key* states quickly. The learner is tested on a B21 robot for a goal reaching task. Experimental results show that after few trials the robot has indeed learned the right situation action rules that unfold its path.

1 Introduction

For more than a decade reinforcement learning (RL) has been studied extensively and its properties are well understood. One of its nice property is that it allows agents to be programmed by reward and punishment, without the need to specify how the task is achieved. Unfortunately, it has an inherent problem - its learning time increases exponentially with the size of the state space. Consequently, RL has remained difficult to implement in realistic domains that are characterized by large state and action spaces - typically robot domains. Yet, despite this inherent problem, there is still a surge of interest in putting RL onto a real robot.

Researchers have tried to overcome the inability of RL to scale well to learning tasks with large state and action spaces. Mahadevan et al. [4] have decomposed the task into sets of simple sub-tasks each with its own prewired applicability predicate. Matarić [5] has minimized the state space by transforming state-action pairs to condition-behavior pairs and maximized learning by designing reward rich heterogeneous reinforcement. Recently, Millán [6] has tremendously accelerated RL by integrating it with reflex rules that focus exploration where it is mostly needed. The common characteristic of the above examples is that the basic RL algorithm has been endowed with some *built-in* *knowledge*. In each case, however, the built-in knowledge has different forms and is used for different purposes: in [4] to break down and to arbiter tasks, in[5] to design rich reward, and in [6] to focus exploration.

This paper is concerned with using environment knowledge to pre-structure the state space. There is a conflict between the required number of states and the actual states constructed by the *adaptive state space construction* algorithm [6]. This is not because of the algorithm, but because of the particular platform we are working with. To resolve this conflict, the controller is shaped to accommodate implicit environment knowledge that enables the algorithm to construct appropriate state space during the course of learning.

2 The robot task

The B21 robot from RWI has been used as our experimental platform. The robot is a four-wheeled cylindrical synchro-drive with two parts: a base and an enclosure. The base carries 32 infra-red (IR) and 32 tactile sensors. Whereas the enclosure carries 24 tactile, 24 IR, and 24 sonar sensors.

The task of the robot is to reach a specified goal $\boldsymbol{p}_g$ through a (sub)-optimal path. Optimality is defined on a certain payoff function. For every action the robot has chosen, it receives an immediate reinforcement r_t that has two components. The first component penalizes the robot when it either collides with or approaches an obstacle. Whereas the second component penalizes the robot in proportion to the angle between the robot heading and the vector connecting the current robot and goal locations. The immediate reinforcement value is the sum of these two components and the payoff function is defined as the sum of immediate reinforcements the robot receives until it reaches the goal, i.e., $R = \sum_t r_t$.

In any mobile robot control, determining the robot position is one of the crucial issue. In the presented learning system the robot position has been used in two ways. First to decode the relative distance between the robot and the goal (section 3) and second to provide a part of the reinforce function from which the robot learns. From the two, the latter one is more sensitive to the inaccuracy of robot position. Because it leads to inconsistent reinforce function that makes learning difficult or even impossible[†]. In this work, *dead reckoning* method has been used to obtain the robot position, $\boldsymbol{p}_r(t)$. However, to get a satisfactory reading, we have exploited the crucial property of the robots' dead reckoning system. Dead reckoning system performs satisfactory provided that the robot does not move for prolonged periods of time with out reaching the goal. This characteristic puts directly a limit on how far and how hidden the goal should be placed from the robot.

[†]Noting this, Millán [7] has eliminated the dependence of the reinforcement value on the odometry reading by building other types of sensors that are capable of detecting the goal *directly*.

3 Embedding

The input $\boldsymbol{x}$=[$\boldsymbol{s}$,$\boldsymbol{d}$] to the controller is a vector of 32 elements, each between $[0, 1]$. The first 24 elements are normalized depth readings of the sonar sensors[‡], while the remaining eight inputs are codified distance between the robot and the goal. In the work of [6], where sensor values are made independent of the robot heading, the input to the controller turns out to be a function of the robot position (if sensors noise is neglected), i.e., $\boldsymbol{x} = [\boldsymbol{s}(\boldsymbol{p}_r(t)), \boldsymbol{d}(\boldsymbol{p}_r(t), \boldsymbol{p}_g)]$. In this case, key states that require different actions are easily split.

For most platforms, however, the sensors can not be aligned independently of the base. Consequently, the perceived sensory data would be different every time the robot visits a given location at different headings, i.e., $\boldsymbol{x} = [\boldsymbol{s}(\boldsymbol{p}_r(t), \theta_r(t)), \boldsymbol{d}(\boldsymbol{p}_r(t), \boldsymbol{p}_g)]$. This results in huge states which the adaptive state space constructor could not cope with identifying and splitting key states quickly. In order to overcome this problem, we have *embedded* environment knowledge [3] [8] into the learning architecture. The environment in which the robot operates is first partitioned into four regions that are considered to be the same for the purpose of learning and generating actions. These are: a `concave` region that misleads and fold the robot path, a `door` region through which the robot has to carefully pass, a `corridor` where the goal is located, and a vast space inside the `room` from where the robot starts off. These partitions together with their corresponding metric data are supplied to the controller as built-in knowledge. From the metric data and the robot position $\boldsymbol{p}_r(t)$, disjointed rules have been written to single out a particular partition where the robot is found. This early splitting of the state space based on prior environment knowledge can be viewed as one way of giving leverage to the adaptive state space constructor so that during the course of learning it can construct appropriate states for each partitions.

Apart from environment knowledge, two fuzzy behaviors [9] *obstacle avoidance* and *goal following* are used as reflex that enable the controller to act initially in some reasonable way. The reflex delivers the next robot heading α, whenever it is requested. The fuzzy reflex works as follows. First, the range of possible robot heading has been fuzzified into three fuzzy sets: `left`, `straight` and, `right`. The obstacle avoidance behavior receives the range data of the sonars and outputs a vector $\boldsymbol{\alpha}_a$ - whose elements indicate the activation levels of the above fuzzy sets. Likewise, the goal following behavior inputs the acute angle θ between the robot heading and the vector connecting the current robot and goal locations and outputs a similar vector $\boldsymbol{\alpha}_g$. A simple behavior blender with constant *desirability functions* d_a and d_g $(d_g \leq d_a)$ is used to combine the output of the two behaviors, i.e., $\boldsymbol{\alpha}_f = d_g\boldsymbol{\alpha}_g + d_a\boldsymbol{\alpha}_a$. Subsequently, a defuzzifier decodes the fused vector $\boldsymbol{\alpha}_f$ to a crisp value α using centroid technique.

[‡]Since IRs are short range ($\approx 0.3m$) proximity sensors, they are used here in emergency routine only.

4 Controller

The architecture of the controller is an actor critic type that is proposed by [6]. In most actor critic systems, two networks are adapted over time - an action and a critic network. In the proposed architecture, however, these networks are integrated into one network. Besides, unlike the former ones where the training rules adjust certain weights of the action or critic networks, the latter one adapts directly the critic and action values. The controller consists of a gradually growing RBF neurons in the input layer and a stochastic neuron in the output layer. Whenever a new situation is perceived, the controller uses the built in knowledge to associate the situation to one of the partitions discussed in section 3. Within the partition existing neurons (if any) compete to win the situation. If a winning neuron exists, it will be connected to the output layer to generate action. Action is generated by exploring a restricted area around a prototypical action. To enforce exploration a *Gaussian stochastic unit* with parameters (μ, σ) is introduced at the output layer [2]. The extent of the exploration is determined by the critic (utility) value u_j and the temperature factor $T(n)$, i.e., $\sigma = T(n)f(u)$. At the end of every trial the temperature is cooled down so that the stochastic unit produces a progressively deterministic output [1].

The adaptive state construction algorithm introduces a new neuron into the selected subspace when existing neurons can not generalize the current situation or if a selected neuron has performed poorly for the previous situation. When a new neuron is created four learning parameters $(p_j, u_j, w_j$ and, $\boldsymbol{c}_j)$ are attached to it [6]. Each of the parameters are adapted by different adaptation algorithm and error sources. The utility value of the winning neuron $u_j(t)$ is updated by *temporal difference* (TD) method [10]. Williams' REINFORCE algorithm [11] is employed to adapt the weight w_j. Depending on the performance of the winning neuron, its center position $\boldsymbol{c}_j$ is either shifted toward the previous sensation or left untouched. The prototypical action p_j is overridden by a more accurate learned action when the robot reaches the goal through a trajectory whose total payoff is greater than the maximum payoff so far obtained.

5 Experimental results

Figure 1 depicts the trajectories of the robot in the first and the last trials and figure 2 shows the learning curves of the controller against the number of trials. Ten sets of experiments, each consisting of 20 trials were carried out. The vertical error bars in figure 2 indicate the variations of the learning curves. During the first few trials, the robot has taken many steps, figure 2 top-right, to reach the goal, there by incurring a high payoff figure 2 bottom-left, and the number of neurons added to the network has grown sharply figure 2 top-left. As trials goes on, however, the robot has started to unfold its path and neurons are added to the network at a reduced slope than earlier trials. On the sixth trial and afterwards the robot has straighten its path, except at the eighth trial where the robot left the optimum path in search for a better one. In subsequent trials, however, the robot has returned to its previous performance and followed the same path without significant divergence through out the remaining trial. A similar phenomena is also observed in the work of [6].

Comparing the final network performance, figure 2 bottom-right, with that of [6] the following observation can be made. First, since neurons are not shared across partitions, the total number of neurons in this small environment is almost equal to that obtained in the large environment of [6]. Second, due to sensory alignment scheme of [6], the network size has already ceased to grow during the last few trials and hence, the variances of the final network performance are smaller than the one reported here. To obtain a similar performance on B21, we are emulating the turret motor in our subsequent work. Instead of using the sensory sequence that point in the direction of the robot heading, the controller can mentally rotate (in the reverse direction of the base rotation) the sensors in such a way that the new sensory sequence points always towards the goal.

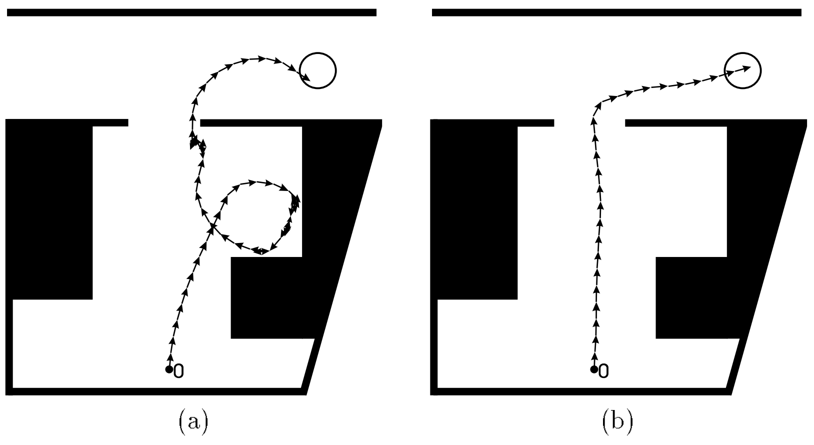
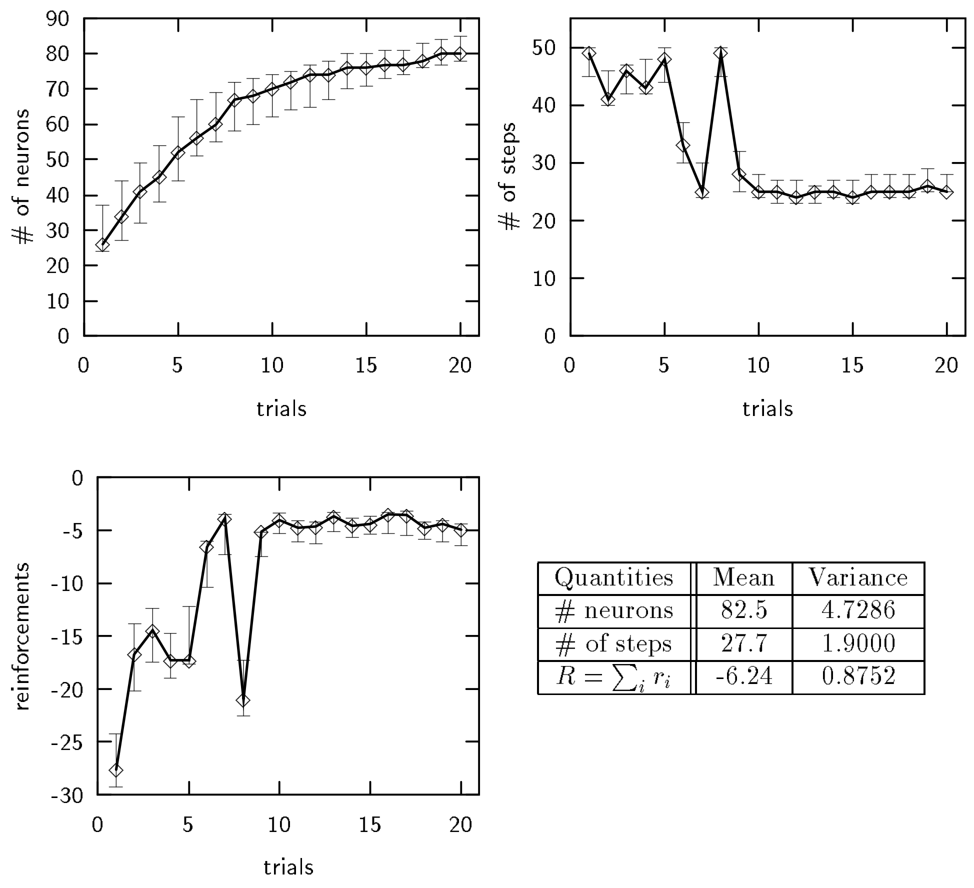


Figure 1: Trajectories of the robot during the first (a) and final (b) trials. The robot has learned 1) to skip the concave region that causes the robot to fold its path, 2) to pass in the middle of the door, and 3) to head directly to the goal after it has passed the door.

6 Conclusions

Two kinds of built-in knowledge have been used to support RL on B21 robot. The first one is *a priori* environment knowledge to pre-structure the state space rapidly. Whereas the second one is two fuzzy behaviors combined with fixed desirability values to focus exploration. Experimental results have shown that



Quantities	Mean	Variance
# neurons	82.5	4.7286
# of steps	27.7	1.9000
$R = \sum_i r_i$	-6.24	0.8752

Figure 2: Learning curves of the robot. Top-Left: The size of the network vs. the number of trials. Top-Right: Number of actions the robot has required at each trial to reach the goal. Bottom-Left: The total reinforcement (penalty) the robot has incurred at each trial. Bottom-right: The final network performance.

the robot has indeed learned to unfold its path and to consistently follow a trajectory that has a minimum payoff value.

7 Acknowledgment

We would like to extend our thanks to J. R. Millán for a useful discussion on TESEO's architecture that helped us greatly in this work. The support given to the first author by DAAD under grant code 413/ETH-4-BOA is also acknowledged.

References

[1] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[2] Vijaykumar Gullapalli. A stochastic reinforcement learning algorithm for learning real valued function. *Neural Networks*, 3:671–692, 1990.

[3] Leslie P. Kaelbling. *Learning in Embedded Systems.* The MIT Press, Cambridge, 1993.

[4] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[5] Maja J. Matáric. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learining*, 1994.

[6] José R. Millán. Rapid, safe and incremental learning of navigation stratagies. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):408–420, 1996.

[7] José R. Millán. Incremental acquisition of local networks for the control of autonomous robots. In *7th International Conference on Artificial Neural Networks*, pages 739–744, Lausanne, Switzerland, 1997.

[8] Ulrich Nehmzow, Tim Smithers, and John Hallam. Steps towards intelligent robots. Technical Report 502, Universty of Edinburgh, 1990.

[9] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Transaction on Systems, Man, and Cybernetics*, 20(6):1370–1382, 1990.

[10] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[11] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.