Learning to Fly: An Application of Hierarchical Reinforcement Learning

Malcolm Ryan MALCOLMR@CSE.UNSW.EDU.AU
Mark Reid MREID@CSE.UNSW.EDU.AU
School of Computer Science and Engineering, University of New South Wales, Sydney 2052 Australia

Abstract

Hierarchical reinforcement learning promises to be the key to scaling reinforcement learning methods to large, complex, real-world problems. Many theoretical models have been proposed but so far there has been little in the way of empirical work published to demonstrate these claims.

In this paper we begin to fill this void by demonstrating the application of the RL-TOPs hierarchical reinforcement learning system to the problem of learning to control an aircraft in a flight simulator. We explain the steps needed to encode the background knowledge for this domain and present experimental data to show the success of this technique.

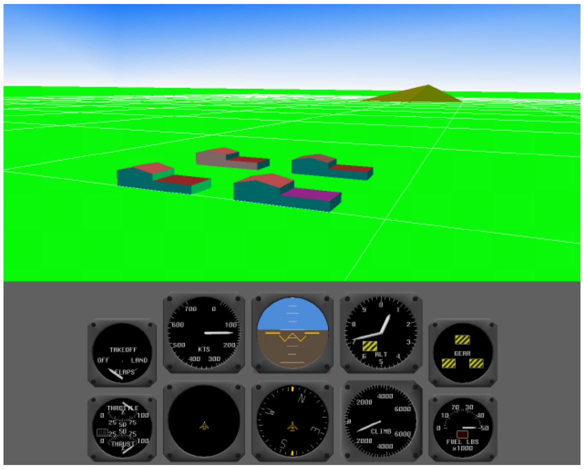

Figure 1. A sample screen-shot from the flight simulator.

1. Introduction

A significant amount of work has recently been published on the topic of hierarchical reinforcement learning (eg. Dietterich, 1998, Hauskrecht et al., 1998, Parr & Russell, 1998, Sutton et al., 1999). Hierarchy is seen as a way to scale traditional reinforcement learning algorithms up to handle complex, multi-dimensional problems. By using appropriately formulated background knowledge a complex learning task can be broken down into several smaller subtasks. These, it is asserted, can be learnt more quickly and easily and can be recombined in some fashion to solve the greater problem. Various systems of decomposition and recombination have been proposed but this broad characterisation holds.

Yet in spite of the proliferation of such methods and the promise they show, there has so far been little empirical work published demonstrating their application to truly complex domains (with the notable exception of Kalmár et al., 1998). In this paper we shall begin to fill that void by demonstrating the application of one such system, the RL-TOPs architecture (Ryan & Pendrith, 1998), to a realistic problem domain — that of learning to fly.

This paper is arranged as follows: Section 2 contains a description the flight simulator that is our problem domain. Section 3 briefly outlines the motivation and operation of the RL-TOPs architecture. Section 4 explains in detail the steps required engineering background knowledge of the flight domain into a form that the RL-TOPs agent can use. Experimental results are provided in Section 5 and final conclusions are drawn in Section 6.

2. The Flight Simulator

Controlling an aircraft in a flight simulator was seen as an ideal demonstration of hierarchical reinforcement learning. It involves fine-grained control based on a large number of input variables which have different degrees of relevance at different stages of the flight.

Table 1. The symbolic description of the scene in Figure 1.

```
time(203.2).
visible(centre of build1).
bearing(plane,centre of build1,
        1235.267212, -8.236785, -2.103162).
visible(centre of build2).
bearing(plane,centre of build2,
        1109.746460, 5.127558, -3.112595).
visible(centre of build3).
bearing(plane,centre of build3,
        991.044739, -17.268343, -4.294295).
visible(centre of build4).
bearing(plane,centre of build4,
        829.350098, -1.246226, -6.422787).
visible(top of mountain4).
bearing(plane,top of mountain4,
        9454.479492, 18.013281, 7.788969).
position(plane, -1878.2, 219.8, 5718.7).
roll_pitch_yaw(plane, 0, 7, 337).
stick(-0.037500, 0.328125).
thrust(1.0).
airspeed(138.8).
climb(-15.8).
flaps(0).
gear(0).
onGround(no).
crashed(no).
autopilot(off).
```

Thus the state space of the problem is huge and the goal can be thousands of actions away from the starting state. This makes for a very difficult reinforcement learning task. Years of training human pilots, however, has left us with a large body of background knowledge on the topic and a natural decomposition of a flight into sequences of manoeuvres which are themselves composed of simpler behaviours.

The flight simulator used in these experiments, simply called "Fsim", simulates a PC-9 aircraft. It was written as part of a joint research effort between the Universities of Melbourne and New South Wales, Curtin University of Technology and the DSTO Aeronautical Research Laboratory. It's design was particularly chosen to include a "Symbolic Description Generator" (Dillon et al., 1993) which outputs a description of the pilot's view and the instruments in a symbolic form appropriate for manipulation by an AI system. Figure 1 and Table 1 show an example screen shot from the simulator and the corresponding output from the description generator as a set of Prolog facts.

The simulator has a socket interface allowing it to be controlled by an arbitrary autopilot program. Our learning agent runs as a separate process, controlling the simulator by commands setting the stick position and the thrust, flaps and gear settings and reading back the symbolic output. Control is synchronised so that each action takes 0.5s of simulated time.

3. The RL-TOPs Architecture

The RL-TOPs system is a synthesis of symbolic and statistical AI methods. Symbolic methods are ideal for imparting background knowledge to the learning agent and reasoning about it, whereas statistical methods are more suited to the problem of fine grained control. The challenge is combine these advantages into a single system.

Central to this synthesis is the idea of a *Reinforcement-Learnt Teleo-operator* (or *RL-TOP*) based on the teleo-operators (*TOPs*) of Benson and Nilsson (1994). These operators are high-level behaviours defined in terms of their preconditions and effects, but unlike standard TOPs their behaviour is not hard-coded rather it is learnt by reinforcement learning.

For example, in the flight domain the Ascend(Target) behaviour, which ascends to within 100ft of the given target altitude, might be described as:

```
Ascend(Target)
  Post:
    within_range(altitude, Target, 100).
  Pre:
    less_than(altitude, Target),
    flaps(off),
    gear(up).
```

This description serves two purposes. Firstly it advises the agent how to use this behaviour in the decomposition of a more complex task, and secondly it is used as part of the learning process itself, as a description of the reward function for this behaviour: +1 if the post-condition is achieved, -1 if the precondition is violated prematurely.

Given many behaviour descriptions like these the RL-TOPs system uses a symbolic planner to combine them into reactive plans to solve more complex goals. So, for example, the task of flying over a given landmark in the flight simulator can be decomposed into a plan involving taking off, ascending to the right altitude, turning towards the landmark, and so on. Each of these sub-tasks is described by a separate RL-TOP and learnt as a separate behaviour. Combined they form a solution to the more complex problem.

3.1 Levels of Granularity

Task decomposition need not be a single-level affair. The Teleo-Reactive formalism allows for hierarchial program in which the behaviours are themselves de-

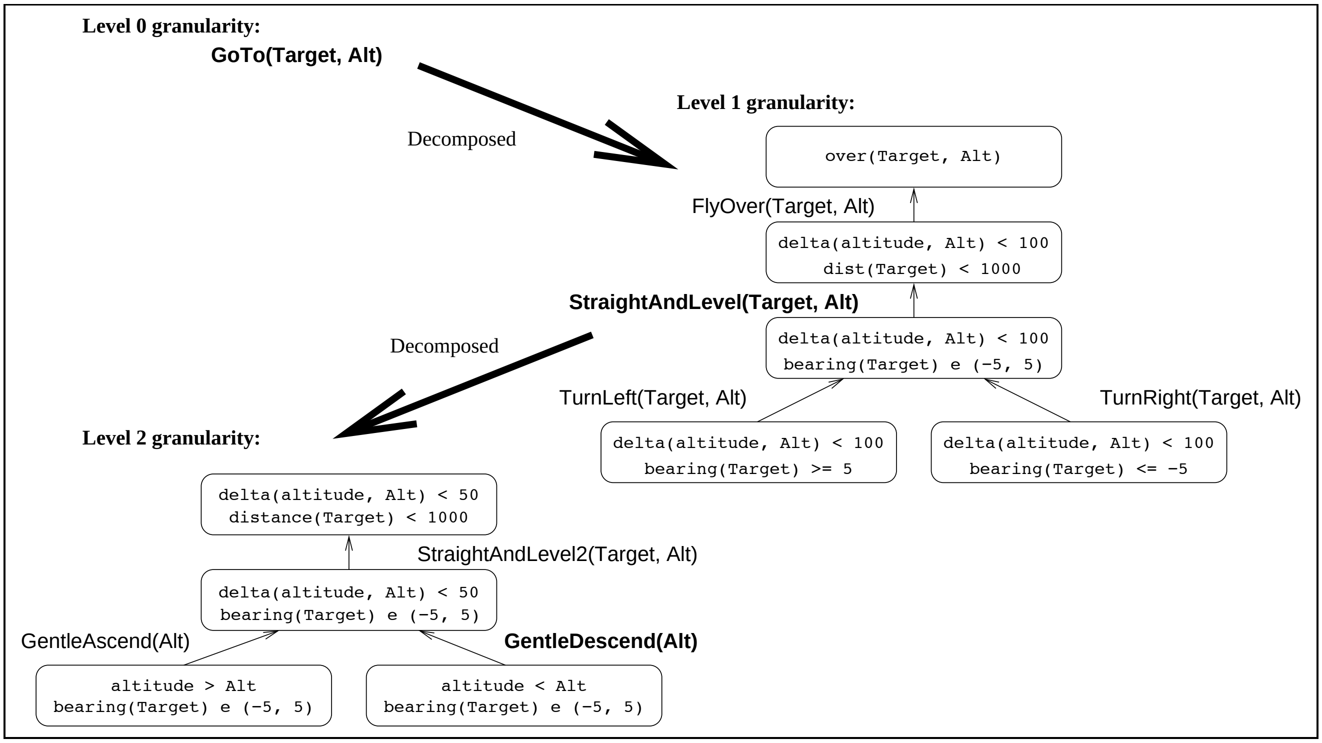

Figure 2. An example of behaviour decomposition in the RL-TOPs architecture. The top-level behaviour, GoTo(Target, Alt), has been decomposed into a plan consisting of behaviours of granularity one. One of these behaviours, StraightAndLevel(Target, Alt), has been further decomposed using behaviours of even finer granularity.

composed into plans (Nilsson, 1994). To achieve this we define behaviours at different *levels of granularity.*

At the coarsest level (granularity zero) there are the most general behaviours which cover the entire state-space and require the most complex goals. At finer levels (granularity 1, 2, 3 etc.) there are progressively simpler behaviours with more specialised goals and more limited preconditions. Figure 2 demonstrates the process of successive decomposition of behaviours of one granularity into plans of the next finest level. Behaviours of granularity zero are decomposed into plans of behaviours of granularity one which can in turn be individually decomposed into plans of behaviours of granularity two, and so on.

3.2 Execution

Given a behaviour to execute we now have two options: (1) we can follow the action dictated by its own reinforcement-learnt policy, or (2) we can decompose the behaviour into a plan of finer granularity behaviours and recursively execute the behaviour dictated by the plan. We use a simple heuristic to choose between these two options: if the Q-value for the behaviour's current policy action is non-zero then the behaviour is confident in this action and follows it; if the Q-value is zero (the initial value) the it assumes that this action hasn't been well explored and it defers to its decomposition. An exception to this rule is when the behaviour cannot be decomposed or the decomposition does not cover the current situation in which case the policy action is executed regardless of its Q-value. Pseudo-code for this operation is illustrated in Table 2.

3.3 Learning

The RL-TOPs system incorporates a limited version *all goals updating* (Kaelbling, 1993). An experience gained by executing a behvaiour is not only used to update that behaviour but also any activated behaviours above or below it in the hierarchy. Each behaviour evaluates the experience based on the reward function dictated by its own pre- and post-conditions and adds the resulting $(state, action, reward)$ triplet to a *lesson.*

When the behaviour terminates or the lesson grows past its maximum length the lesson is replayed using Lin's TD(0) experience replay algorithm (Lin, 1993). (It is important to note that this is an off-policy learning algorithm. This allows behaviours to learn from

Table 2. The RL-TOPs algorithm

```
function RL-TOPs(Goal goal)
    let state := initial state after resetting the flight simulator
    repeat
        let activeTops := getActiveTops(state)
        let action := selectAction(activeTops, state)
        let state' := the result of executing action in the flight simulator
        updateTops(activeTops, state, action, state')
        state := state'
    until $state \in goal$ or state represent a critical failure
end RL-TOPs

function getActiveTops(Goal g, State s)
    let $k := 0$
    let $\mathsf{T}_0$ := a top of granularity zero that achieves g.
    while $\mathsf{T}_k$ can be decomposed do
        let $\mathsf{T}_{k+1}$ := the top dictated by the decomposition of $\mathsf{T}_k$ for state s
        $k := k + 1$
    end while
    return $\mathsf{T}_0, ..., \mathsf{T}_k$
end getActiveTops

function selectAction(Tops $\mathsf{T}_0, ..., \mathsf{T}_n$, State s)
    for each $\mathsf{T}_k \in \mathsf{T}_0, ..., \mathsf{T}_n$ do
        let $a = \arg\max_{a'} Q(\mathsf{T}_k, \mathsf{state}, \mathsf{a}')$
        if $Q(\mathsf{T}_k, \mathsf{s}, \mathsf{a}) \neq 0$ then
            return a
        end if
    end for
    return a
end selectAction

function updateTops(Tops $\mathsf{T}_0, ..., \mathsf{T}_n$, State s, Action a, State s')
    for each $\mathsf{T} \in \mathsf{T}_0, ..., \mathsf{T}_n$ do
        if $s' \in \mathsf{T.post}$ then
            let $r = 1$
        else if $s' \notin \mathsf{T.pre}$ then
            let $r = -1$
        else
            let $r = 0$
        end if
        add $(s, a, r)$ to T.lesson
        if terminated(T) or length(T.lesson) = MaxLessonLength then
            replay T.lesson
            T.lesson := {}
        end if
    end for
end updateTops
```

experiences which do not come from executing their own policies.) Table 2 shows pseudo-code for this operation.

4. Adding Background Knowledge

The knowledge engineering for a task like flying the simulator is not trivial. A human trainer possesses both declarative and procedural knowledge he wishes to impart to the computer. We make use of both types of knowledge by a combination of direct encoding and Behavioural Cloning methods (Sammut et al., 1992).

The RL-TOPs architecture requires the trainer to supply four kinds of background knowledge: a symbolic state description language, a set of high-level behaviour descriptions, a primitive state description and a set of primitive actions. In addition to these required forms of knowledge the learning agent was also assisted by replaying a flight recorded by a human pilot. For the flight simulator experiments each of these parts required careful design, as outlined below.

4.1 Symbolic State Descriptions

The symbolic description of the high-level state began with the Prolog facts output by the Symbolic Description Generator. These described the objects in the world and their location relative to the aeroplane, and the settings of the controls and instruments. On every iteration of the main control loop the learning agent read this information from the flight simulator. This was augmented by a set of Prolog predicates written by the trainer which extract various parts of this information such as `roll(R)` or `altitude(A)`, or compare a given instrument value with a fixed threshold, as `less_than(Instrument, Value)`.

4.2 Symbolic Behaviour Descriptions

Given the symbolic state description language defined above the trainer then provided a set of behaviour descriptions. These were based on the standard manoeuvres taught to pilots from a flight training manual (Thom, 1992). Behaviours were defined at three levels of granularity. At the coarsest level GoTo(Target, Altitude) was a single monolithic behaviour the goal of which was to fly the aeroplane over a given landmark at a certain altitude. At level one there was a set of standard manoeuvres such as TakeOff, Ascend(Altitude) and TurnLeft(Target, Altitude). At level two – the finest level – the behaviours were more specialised, with simpler goals and small application spaces. These corresponded to the various sub-parts of the standard manoeuvres, such as the MaintainLeftTurn(Target) behaviour which holds the aeroplane in a turn until it is close to facing the Target.

4.3 Primitive State Representation

An important advantage offered by hierarchical reinforcement learning is the ability to provide different state abstractions for different behaviours (Dietterich, in press). This allows each behaviour to view only those parts of the state space that are important to it and to represent them in a way that is relevant to that behaviour. The RL-TOPs architecture allows the trainer to specify for each behaviour a set of "functions" that form its state-space. So for example the Ascend(Altitude) behaviour had the following state variables: `roll`, `pitch`, `airspeed`, `climb`, and `delta(altitude, Altitude)`. Each of these referred to one of the instruments read from the flight simulator, except for the last which was a call to a Prolog predicate which calculates the difference between the `altitude` instrument, and the `Altitude` argument to the behaviour.

Most of the state variables for the behaviours were continuously valued and so were discretised for the purpose of representing the Q-function. The appropriate discretisation varied from one behaviour to another and was not obvious to the trainer, so they were instead computed from actual flight data in the following manner.

A human pilot flew five training flights each about 800 actions (6-7 minutes) long, which involved flying over three different landmarks and included examples of all the described behaviours. The state information was recorded and replayed through the learning agent which computed which of its behaviours it deemed appropriate at each point, according to its plan. Each state was recorded as a set of typical state values for the corresponding behaviour.

Then, treating each state variable for each behaviour individually, the values were sorted and partitioned into five sets by *equal frequency binning* (Dougherty et al., 1995). The boundaries between these partitions formed the discretisation of that variable. Thus portions of the state space that were visited more frequently by the human pilot were discretised more finely.

4.4 Primitive Action Selection

As well as having a multidimensional continuous state space the flight simulator control problem has five dimensions in its action space: the x and y positions of the stick, the thrust, the flaps and the gear settings.

The first three of these can take on a large number of values. As with the primitive state representation, an abstraction was needed to bring this space to a manageable size. Again this abstraction varied between behaviours, as different behaviours required the use of different controls. So, as with the primitive state, the RL-TOPs system allows the trainer to specify a separate set of actions for each behaviour.

For these experiments the appropriate action values were computed in much the same way as the state representations. Each behaviour was given a set of relevant action variables, examples of which were recorded from the human flight data. These data were sorted and divided into four bins by equal-frequency binning and the five partition boundaries (including the two extremes) were taken as the five possible values of that action variable. The primitive actions for each behaviour were all the combinations of possible values for all the action-variables assigned to that behaviour. Thus, for example, the Ascend behaviour had 125 actions assigned to it corresponding to all combinations of five x-positions for the stick, five y-positions, and five different thrust settings.

4.5 Learning by Observation

Even with the provision of this background knowledge learning even the simplest of the defined behaviours by random exploration alone was extremely time-consuming. In order to overcome this one further kind of background knowledge was used — the observation of a more experienced pilot.

A single flight flown by the human trainer was replayed through the learning system and the pilot's actions and experiences were used to update the agent's behaviours. In order to maximise the use of this information any behaviour which had its precondition satisfied by a certain experience was updated using that experience.

For example, the state described in Table 1 satisfies the precondition of the Ascend(1000) behaviour as the altitude is less than 1000ft. For the same reason it also satisfies the precondition of the Ascend(1200) behaviour, and since the altitude is greater than 500ft, it also satisfies the preconditions of Descend(500). All of these behaviours are therefore able to evaluate the pilot's action in this state and use this experience to update their own policies regardless of what manoeuvre the pilot is actually performing.

5. Experimental Results

5.1 Experiment 1: Hierarchical vs. Non-Hierarchical RL

Our first experiment compared the performance of the following three approaches: (1) non-hierarchical learning, (2) learning with one level of decomposition, and (3) learning with two levels of decomposition. Each approach was run ten times [1] with all thirty runs seeded with the Q-functions learnt from replaying the training flight, as explained above.

Each run consisted of 6 hours of simulated flying time (equal to 43200 action steps). Figures 3(a) and (b) show the results of these runs, averaged over each batch of ten. The first of these graphs shows the number of successful flights flown over the duration of the trial. There is a clear sequence of improvement from the non-hierarchical learner (which did not complete a single flight in any of the trials) through the single-level hierarchy to the two-level hierarchy. A t-test assuming unequal variances verifies this improvement with 82.5% confidence. This is arguably quite low but not unreasonable considering the small number of runs.

A more striking difference is found by comparing the percentage of flights that ended in the learner crashing the aeroplane. The non-hierarchical learner crashed 100% of the time, an average of 393 times per run. The single-level learner crashed 95% of the time (255 times per run) whereas the two-level learner crashed on only 46% of the flights (26 times per run). Again a t-test assuming unequal variance shows this difference is significant with 98% confidence.

Figure 3(b) shows a closer comparison of the learning performance of the two hierarchical agents. As the agents improve, the average length of the flight should become shorter. As the graph shows both agents were unable to fly reliably when they began their online learning, in spite of the training they had already received. The two-level learner began to succeed more consistently after 1.5 hours and continued to improve its performance as time went on. The single-level learner began succeeding later, at around 2 hours of learning time, and while it improved quickly at first, its performance appears to have degraded slowly as time progressed. The reason for this is not apparent. Longer experiments are required to test whether this trend continues.

[1] The extremely small number of runs is due to the fact that each run took upwards of 9 hours to complete. Work is underway to port the flight simulator to Linux, allowing us to run the learning algorithm on a much larger number of faster machines.

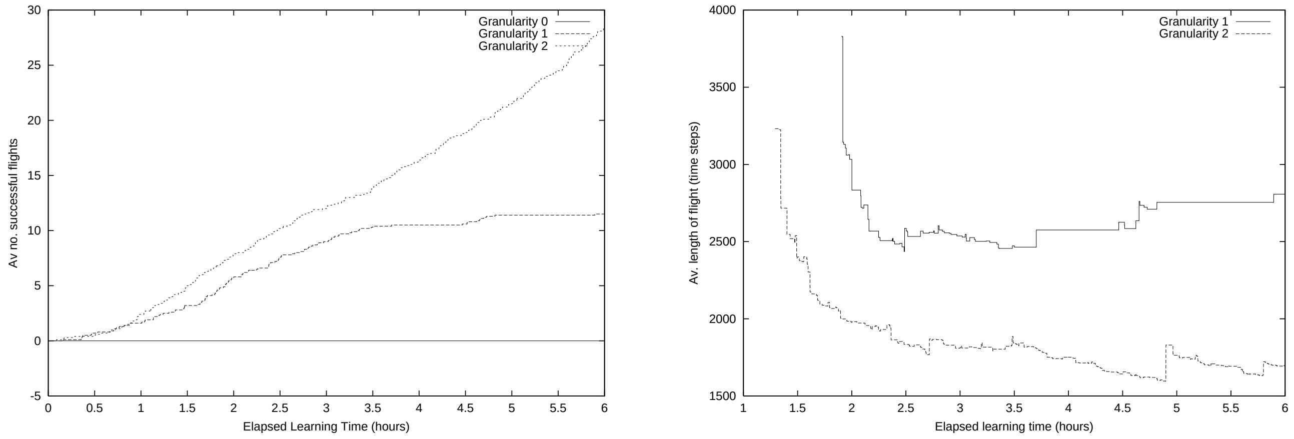

Figure 3. Performance on the learning to fly task using behaviours of different granularities, in terms of (a) the number of successful flights, and (b) the length of the flights.

5.2 Experiment 2: Learning at Multiple Levels

The second experiment was designed to test the concurrent learning of behaviours at different levels of granularity. We compare the mid-level (granularity one) behaviours learnt by the two hierarchical learners in the first experiment. It would be expected that the behaviours learnt by the second learner, which included behaviours of granularity one and two should perform as well as or better than those learnt by the first. This turned out to not be the case.

The behaviours of granularity one were taken from each of the learners in batches 2 and 3 in experiment one. These behaviours were used to control the plane for a further half an hour of flying. In that time, the behaviours from batch two (the single-level learners) flew an average of 4.2 successful flights, whereas the the behaviours learn in batch three (the two-level learner) averaged only 1.5. On the other hand, the number of crashes is better for the behaviours from batch three, which crashed the plane an average of 6.4 times per half hour, compared to 8.0 times for the behaviours from batch two. Time constraints have prevented us from analysing in depth why the expected improvement did not occur, but preliminary investigation suggests that it is due to the differences in the state and action abstractions at different levels of the hierarchy. Further investigation is necessary.

6. Conclusions and Future Work

This work serves to demonstrate empirically the viability of the RL-TOPs system and of hierarchical reinforcement learning in general. It highlights the various kinds of background knowledge, both declarative and procedural, that need to be exploited for a task of this complexity and provides impetus for the design of learning systems of greater flexibility, so that we can indeed make use of this knowledge.

The experimental results, while affirming the usefulness of the technique, leave many questions to be asked. Further experiments shall be conducted to examine the longer term learning effects, and the effects of simultaneously learning behaviours at multiple levels of granularity.

Development is underway on "closing the loop" between background knowledge and learnt behaviours in the system. The symbolic description provided by the trainer is to be augmented based on the agent's own experience, using Inductive Logic Programming. This should further enable the automatic invention of new behaviours, the Holy Grail of hierarchical reinforcement learning.

Acknowledgements

The authors would like to thank Megan Corson, Andrew Piper, Nick Begbie and Trish Delaney-Brown for their assistance in writing this paper, and Virginia Wheway who provided invaluable help with the statistical analysis of results.

References

Benson, S., & Nilsson, N. J. (1994). Reacting, planning and learning in an autonomous agent. In K. Furukawa, D. Michie & S. Muggleton (Eds.), *Machine intelligence 14*. Oxford: the Calrendon Press.

Dietterich, T. G. (1998). The maxq method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning.* San Franciso, CA: Morgan Kaufmann.

Dietterich, T. G. (in press). State abstraction in maxq hierarchical reinforcement learning. *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference.* San Franciso, CA: Morgan Kaufmann.

Dillon, C., Caelli, T., Goss, S., & Murray, G. (1993). *Symbolic description generator: Final report* (Technical Report). DSTO Aeronautical Reseach Laboratory.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Proceedings of the Twelfth International Conference on Machine Learning.* San Franciso, CA: Morgan Kaufmann.

Hauskrecht, M., Meuleau, N., Boutilier, C., Kaelbling, L. P., & Dean, T. (1998). Hierarchical solution of markov decision processes using macro-actions. *Proceedings of the Fourteenth International Conference on Uncertainty In Artificial Intelligence.*

Kaelbling, L. P. (1993). Learning to achieve goals. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.* San Franciso, CA: Morgan Kaufmann.

Kalmár, Z., Svepesvári, C., & Lőrincz, A. (1998). Module-based reinforcement learning experiments with a real robot. *Machine Learning*, *31*, 55–85.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks.* Doctoral dissertation, School of Computer Science, Carnegie Mellon University.

Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference.* San Franciso, CA: Morgan Kaufmann.

Ryan, M. R. K., & Pendrith, M. D. (1998). Rl-tops: An architecture for modularity and re-use in reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning.* San Franciso, CA: Morgan Kaufmann.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. *Proceedings of the Ninth International Conference on Machine Learning.* San Franciso, CA: Morgan Kaufmann.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Thom, T. (1992). *The flying training manual.* Williamstown, Vic. 3016 Australia: Aviation Theory Centre Pty. Ltd.