Policy Transfer with a Relational Learning Classifier System

Drew Mellor
School of Electrical Engineering and Computer Science
The University of Newcastle, Callaghan, 2308, Australia
Telephone: (+612) 4921 6034, Facsimile: (+612) 4921 6929
dmellor@cs.newcastle.edu.au

ABSTRACT

Policy transfer occurs when a system transfers a policy learnt for one task to another task with little or no retraining, and allows a system to perform robustly and learn efficiently, especially when the new task is more complex than the original task. In this paper we report on work in progress into policy transfer using a relational learning classifier system. The system, FOX-CS, uses a high level relational language (a subset first order logic) in combination with a P-learning technique adapted for XCS and its derivatives. FOX-CS achieved successful policy transfer in two blocks world tasks, *stacking* and *onab*, by learning a policy that was independent of the number of blocks, thus avoiding the prohibitive training times that would normally arise due to the exponential explosion in the number of states as the number of blocks increases.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Predicate logic; Representations (procedural and rule-based)*

General Terms

Algorithms, Languages

Keywords

Policy transfer, relational learning, first order logic, learning classifier system, XCS, blocks world

1. INTRODUCTION

When a task or task environment is changed in some way, very often much of the regularity that forms the basis for generalisation remains the same, thus systems that are capable of expressing generalisations at a sufficiently abstract level may be able to transfer them to other related tasks. For online learning systems a capacity for transferring generalisations is of interest because generalisations reduce training time (as well as memory requirements). In the reinforcement learning setting, Dzeroski et al. [2] and Cole et al. [1] have shown that using high level relational languages in conjunction with policy learning, or P-learning, successfully allowed policies to be transferred between related tasks. The systems were applied to blocks world tasks with a given number of blocks, and after the number of blocks was increased the systems demonstrated the same level of performance with little or no retraining. The experiments demonstrate two advantages which arise from successful transfer: *robustness*, because the systems could still perform after the environment was changed; and *efficiency*, because the time cost of training was reduced by learning in the blocks worlds with less blocks.

In this paper we show how the above approach can be applied to learning classifier systems. First we give a method for doing P-learning with the XCS system [6], and then we demonstrate that an XCS derivative system with a P-learning extension can successfully transfer policies between blocks worlds with different numbers of blocks. In order to represent generalisations about blocks world in a relational way the system we employed was the FOX-CS system [4], which supports the use of first order logic languages for representing inputs and classifiers.

2. APPROACH

In this section we show how to do P-learning with an XCS system. In P-learning a function $P : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$, where $\mathcal{S}$ is the set of states for the task and $\mathcal{A}$ is the set of actions available to the system, indicates the optimality of a state and action pair with respect to the Q function for the task. Thus, if it is optimal in state s to execute action a then $P(s, a) \mapsto 1$, otherwise $P(s, a) \mapsto 0$ (for a formal definition of P-learning see [2]). Note that the P function does not return payoff values, thus it remains accurate when a task's Q function changes without affecting the optimal policy, which can be the case for related tasks. The P function can be stored in a table, although similar to Q-learning, P-learning is most beneficial when combined with generalisation, in which case the P function is represented by an inductive architecture, such as a decision tree or a list of decision rules.

To calculate P, we follow the approach adopted in [2, 1] of estimating it from the Q value estimates. Here we make the common assumption that the prediction array in XCS

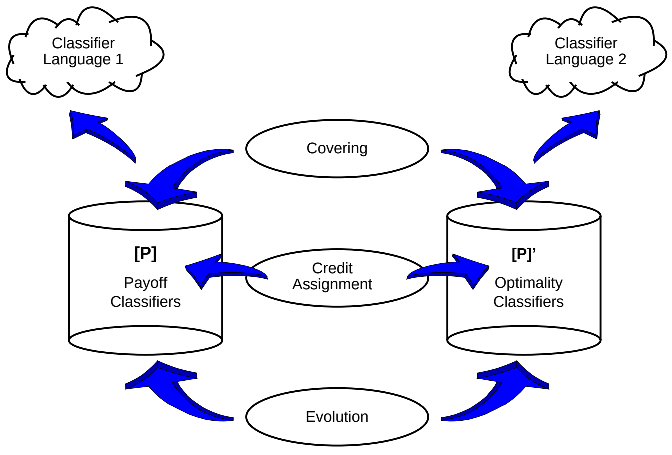


Figure 1: Payoff classifiers and optimality classifiers are separated into two populations, $[P]$ and $[P]'$. Using two classifier languages, generalisations about optimal behaviour can be described using a separate language from generalisations about payoffs.

contains estimates of Q values. Accordingly, $\hat{P}(s,a) = 1$ if the prediction value for a is equal to the maximum value in the prediction array, and 0 otherwise:

$$\hat{P}(s,a) = \begin{cases} 1 & \text{if } p(a) = \max_{u \in \mathcal{A}(s)} p(u), \\ 0 & \text{otherwise,} \end{cases}$$

where $p(a)$ is the system's prediction for action a when the current state is s, and $\mathcal{A}(s)$ is the set of actions available in s. We assign $\hat{P}$ to individual classifiers, but rather than creating another parameter to store $\hat{P}$, we use separate classifiers to estimate the payoff (Q value) and optimality (P value), thus reusing the prediction parameter to store $\hat{P}$. Having separate types of classifiers — let us call them *payoff classifiers* and *optimality classifiers* — allows generalisations that predict optimality to differ from generalisations that predict payoff, in fact it allows two different classifier languages to be used if necessary. Although standard XCS systems may not be able to use an additional language for the optimality classifiers as there is a one-to-one correspondence between the classifier language (for the conditions at least) and the input language, nevertheless some XCS systems that employ higher level languages, such as XCSL [3] and FOX-CS [4], can make use of the capability because in those systems more than one classifier language can be used for a given input language. We introduce a second population $[P]'$ to contain the optimality classifiers. Note that the introduction of a second population to the system, $[P]'$, also brings with it an additional match set, $[M]'$, and an additional action set, $[A]'$, that are composed from classifiers belonging to $[P]'$.

Now we describe how to update the optimality classifiers (the payoff classifiers are updated normally according to the XCS specification). For optimality classifier j, the prediction parameter, p_j, and error parameter, ε_j, is updated in an analogous fashion to the corresponding updates for payoff classifiers; the main difference is that the optimality estimate, $\hat{P}$, is used in the update instead of the Q-learning-like payoff. Thus, the prediction parameter, p_j, is updated according to the following Widrow-Hoff delta rule with learning rate parameter $\beta \in (0,1]$:

$$p_j \leftarrow p_j + \beta(\hat{P}(s, \mathit{action}(j)) - p_j),$$

where $\mathit{action}(j)$ returns the action specified by j. Similarly, the error parameter, ε_j, is updated with:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta(|\hat{P}(s, \mathit{action}(j)) - p_j| - \varepsilon_j).$$

The update of the fitness parameter, including the accuracy calculation, remains unchanged.

3. EXPERIMENTS

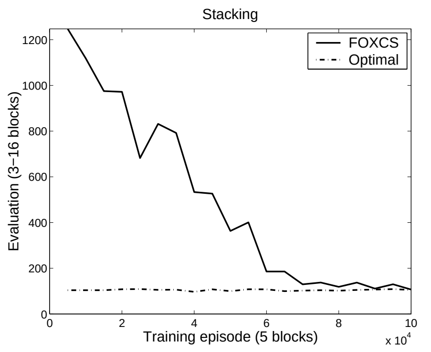
In this section our aim is to reveal whether an XCS system using P-learning can transfer its generalisations between blocks worlds with different numbers of blocks. In order to achieve the level of abstraction required for successful transfer we used the FOX-CS system [4], with the above P-learning extension, because it supports the use of first order logic languages for representing generalisations. The system was trained in separate experiments at the *stack* and *onab* tasks [2] for a fixed number of episodes (an episode starts with a randomly generated state and finishes when the goal of the task is achieved).[1] For both tasks the blocks worlds contained 5 blocks while training, however at regular intervals the system performed 25 evaluation episodes where at the beginning of each episode the number of blocks was set randomly from the range 3 to 16 inclusive. During evaluation all learning was switched off (all parameter updates and evolution of the payoff and optimality classifiers was ceased) and action selection was based on the optimality classifiers in an analogous procedure to action selection over payoff classifiers. The performance measure was the number of steps taken to reach the goal, which was summed over the 25 evaluation episodes to give the final evaluation at that point during training. The experiments were run 15 times for both tasks and the average performance graphed (see figure 2).

The results show that on average the system had learnt an optimal or near-optimal policy that transferred to environments with an arbitrary number of blocks by about 70,000 episodes for *stacking* and about 11,000 episodes for *onab*. As the number of blocks increases the number of states in blocks world grows exponentially and the amount of training required becomes prohibitive, thus the transfer of generalisations can be an efficient way of scaling up to environments with large state spaces when the underlying regularity remains unchanged.

One of the advantages of the rule based approach adopted by learning classifier systems is that the system's hypotheses are relatively comprehensible compared to other inductive architectures such as neural networks. Inspection of $[P]'$ revealed rules that were readily interpretable as useful knowledge.

When we compare the performance of FOX-CS to RRL [2] we find that the most notable difference is in the learning rate. Learning for FOX-CS proceeded at a couple of orders of magnitude slower than for RRL, thus RRL trained over tens of episodes whereas FOX-CS trained over tens of thousands of episodes. The discrepancy in training time may be

[1]Random blocks world states are generated with a uniform distribution using the method given by Slaney and Thiébaux [5]



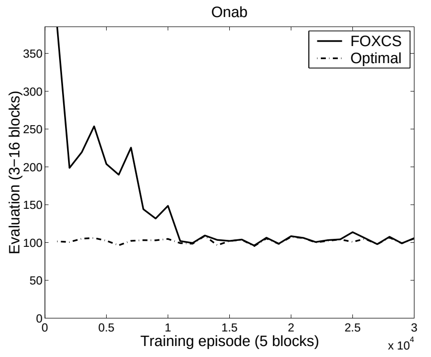


Figure 2: Performance of Fox-cs with P-learning at the *stack* and *onab* tasks. Throughout training the system learnt with 5 blocks, but during evaluation the number of blocks was set randomly at the start of each episode from the range 3–16. The optimal performance is shown for comparison. Results are averaged over 15 runs.

accounted for by two factors: firstly learning classifier systems learn slower than other reinforcement learning systems due to the stochastic nature of the evolutionary component; and secondly there were differences in how the systems were trained. The Rrl system trained with 3 blocks initially before proceeding to 4 and then 5 blocks, whereas the Fox-cs system trained with five blocks right from the beginning, thus Rrl made use of transfer effects whilst in training. On the positive side for Fox-cs it does appear to have learnt the *onab* task to a better performance level than Rrl, with Fox-cs achieving essentially optimal performance whereas Rrl achieves optimality in approximately 90% of its evaluation episodes, although with an equivalent amount of training time Rrl may have achieved a better result.

4. CONCLUSIONS

In this paper we reported on work in progress into policy transfer using a relational learning classifier system. The system, Fox-cs, used a high level relational language (a subset first order logic) in combination with a P-learning technique (generic to Xcs and its derivatives) and achieved successful transfer in two blocks world tasks, *stacking* and *onab*, by learning a policy that was independent of the number of blocks. An immediate advantage that arises from policy transfer is that it provides a solution to the exponential explosion in the number of states as the dimension of the task increases. Using policy transfer the system can efficiently "scale up" from small state spaces to larger ones, thus avoiding the prohibitive training times associated with higher dimensions.

We are currently conducting further experiments in blocks world where the task can vary from episode to episode. In this scenario there are a random number of subtasks to perform, each of which is an *onab* task in itself, hence the order in which subtasks are solved is important. We hope to show that Fox-cs can learn a transferrable policy, i.e. one that is independent of the number of subtasks.

5. REFERENCES

[1] Joshua Cole, John Lloyd, and Kee Siong Ng. Symbolic learning for adaptive agents. In *Proceedings of the Annual Partner Conference, Smart Internet Technology Cooperative Research Centre*, 2003. `www.smartinternet.com.au/SITWEB/publication/publications.jsp`.

[2] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1–2):7–52, 2001.

[3] Pier Luca Lanzi. Mining interesting knowledge from data with the XCS classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 958–965, San Francisco, California, USA, 2001. Morgan Kaufmann.

[4] Drew Mellor. A first order logic classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005), To Appear*, 2005.

[5] John Slaney and Sylvie Thiébaux. Blocks World revisited. *Artificial Intelligence*, 125:119–153, 2001.

[6] Stewart W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.