Incremental Acquisition of Complex Behaviour using Structured Evolution

Simon Perkins
s.perkins@ed.ac.uk

Gillian Hayes
gmh@dai.ed.ac.uk

Department of Artificial Intelligence, University of Edinburgh, 5 Forrest Hill, Edinburgh, Scotland

Abstract

In practice, general-purpose learning algorithms are not sufficient by themselves to allow robots to acquire complex skills — domain knowledge from a human designer is needed to bias the learning in order to achieve success. In this paper we argue that there are good ways and bad ways of supplying this bias and we present a novel evolutionary architecture that supports our particular approach. Results from preliminary experiments are presented in which we attempt to evolve a simple tracking behaviour in simulation.

1 Engineering vs. Evolution

In recent years search/optimization based methods have been widely touted as a way to design robot controllers without all that tedious mucking about with analysing the complex interaction between a robot and its environment.

Unfortunately, despite success in automatically designing controllers for a few simple tasks, pure learning methods do not scale to the complex tasks we would like our robots to perform, *e.g.* tasks involving visual sensing.

Increasingly, people are suggesting that what we need is a hybrid of the two approaches (*e.g.* [3]). Specifically: how can we use domain knowledge supplied by humans to speed up or bootstrap search-based methods? We explicitly recognize the tradeoff between engineering and search, and ask the questions: 'What bits of robot design are suited to human engineering?', 'What bits are suited to automated search?' and 'How do we combine them?'. Our proposed solution to this problem is called *structured evolution*.

2 Structured Evolution

The major principles of SE[1] are:

1. The job of the human designer is to determine the high-level structure of a task, and to devise appropriate environmental constraints to make training tractable.

2. The job of the evolutionary algorithm is to find low-level solutions to simple problems within this structure.

3. The designer shouldn't have to fiddle with the internal details of the evolutionary algorithm.

The first two points recognize that, while humans are generally quite good at decomposing complex tasks into simpler ones at a coarse scale, they are usually rather bad at imagining what a real robot is going to have to do in detail. Similarly, while learning/evolutionary algorithms can develop successful controllers for simple tasks, they are bad at determining the coarse scale structure of complex tasks. These complementary qualities suggest the above division of labour.

The last point makes the claim that, given this division, it is unnecessary for the designer to know about whatever internal representations, connections, weights, sub-symbolic rules etc. the low level learning algorithm is using. This frees the learning algorithm from the constraint of having to produce humanly intelligible solutions, and frees the designer from having to worry about what is going on at the lowest level. Instead, the designer is forced to think and analyze the task at hand at a level more suited to the human imagination.

We feel that these criteria can be met by a robot that acquires complex behaviour in an *incremental*

[1] We use the word evolution, but in fact these principles could be applied to many sorts of learning/search/optimization techniques where the only training signal available is a scalar reinforcement.

fashion. The role of the designer is to specify a path of increasing competence from simple behaviour to complex, and a suitable training scheme to go with it. The role of the learning algorithm is to actually move the controller from one specified point on this path to the next. Note that the human trainer is only concerned with the external behaviour of the robot, and not with the internal workings of the learning mechanism.

3 Methods

There are many methods that a designer can use to provide external constraints to make incremental learning of complex tasks possible, including:

Task Decomposition The robot is initially trained on sub-tasks of the complex task. Hopefully once it has these then the full task can be learned much more easily. Task decomposition has been used by a number of researchers to make learning of complex tasks tractable (*e.g.* [1, 2, 8]). However, we do *not* specify a specific controller hierarchy — that is left to emerge in response to the hierarchical training.

Training Explicit Representations The robot is encouraged to develop internal representations for particular situations that are deemed likely to be useful in later learning. Since we are not 'allowed' to examine or specify internal values, this is done by training external responses *e.g.* 'stick up your right hand if there's a light in front of you'.[2]

Good Reinforcement Policy Design The evolutionary algorithm's job is made much easier if the robot receives frequent evaluations on its performance. [9] provides some discussion of these terms in terms of 'progress estimators' and 'heterogeneous reinforcement'. We are also looking at ways of using non-scalar reinforcement where it is available.

Enriched Learning Environments The rate at which a robot receives rewards can be increased by simplifying or enriching the environment in the early stages of training so that the robot is more likely to achieve goals by accident. This should help speed up learning.

Simulator Training We are keen to produce controllers for real robots. However, several researchers have shown that it is possible to train controllers in simulation and transfer them to a robot later *e.g.* [11]. Simulators also allow more informed evaluation functions.

Prohibiting Irrelevant Sensors/Actuators Sometimes we can say for sure that a robot will not need a particular sensor/actuator for a task, so we can discourage the use of it.

[2] Or equivalently: 'flash LED 1...'

4 An Evolutionary Architecture for Structured Evolution

The incremental approach to robot design required by SE puts constraints on the learning architecture we can use. In particular we would like to be able to separately learn different sub-skills without forgetting old ones, and we would like new sub-skills to be able to take advantage of existing skills and internal representations.

Most evolutionary robotic systems attempt to evolve the whole controller in one go. They work with populations of monolithic controllers and attempt to evolve a single all-knowing controller. We feel this approach to be incompatible with the above aims — it is difficult to retain previously learned skills that are not immediately useful, and difficult to learn a single coordination system that can switch between them all.

We prefer a *multiple expert* approach where the total behaviour of the robot is a result of the interaction between a collection of experts. Ideally each expert is a specialist which is only active in a sub-portion of the total state space of the robot. Experts correspond quite naturally to our sub-tasks in SE.

In our architecture, each expert is itself a population of individuals (called *agents*) and each population is (co-)evolved separately using a fairly conventional EA. The hope is that each population evolves a different specialization so that once converged, each population can be represented by its 'best' agent *i.e.* competition within populations, cooperation between populations.

Each agent takes input from sensors calculates both an output action and a *validity*. This value says how confident the agent is that it is in a part of the state space where it is saying the right thing. If more than one agents end up trying to influence the same actuator, then the one with the highest validity is chosen. In this way the state space is divided up into overlapping regions where different agents have priority.

Agents themselves are simple tree-structured GP-like programs. Their function set is inspired by typical ANN transfer functions (in earlier implementations agents were perceptrons) and terminal nodes are constants or sensory inputs.

5 Preliminary Experiments

We are interested in evolving complex visual behaviours such as tracking moving targets using a real 'pan/tilt head'. This task is particularly suitable for investigating SE since the huge number of raw sensory inputs and difficulty of the task make it very difficult for pure reinforcement learning, while we do have a good idea how to design suitable controllers by hand.

Experiments are still at an early stage and currently we are working with the simpler task of trying to track a simulated bright target against a dark background. The target is initially positioned randomly at the edge of the robot's visual field. Evaluation is then given each cycle proportional to how much the centre of the image moved closer to the target. If the robot 'hits' the target or it goes out of sight, the target is randomly repositioned.

Visual sense inputs for agents sample varying sized regions of the input image in a natural neuron-inspired fashion. Separate actuator outputs are provided for the pan and tilt head velocity.

Evaluating agents within individual populations in our architecture poses problems. The only evaluation that can be given is to the robot as a whole. How do we evaluate the contribution of individual agents? Moreover, how do we cope with the fact that an agent may be evaluated in a part of the state space where it is not valid (and hence can't be blamed for things going wrong)? Our solution is as follows: every second or so, an agent is picked from each population to represent that population. The evaluation given to the robot over the next second is then given equally to all the chosen agents, with an extra weighting if they were particularly confident. We evaluate every agent in each population 20 times in different random parts of the state space, and with different random other agents active, and the total fitness is simply the weighted average of all these evaluations. Breeding then occurs by selecting single parents using rank-based selection and applying point mutation or random sub-tree replacement to generate new agents. The top 20% of the population is retained each generation.

In theory, it is possible to co-evolve several populations at the same time, but in practice interference between non-converged populations seems to make this difficult. Therefore we currently only evolve one new population each time we try to learn some new sub-skill or increase in task complexity, and freeze the others.

6 Results and Further Work

We present here some preliminary results using a simplified version of the above experiment where the robot is deemed to have 'hit' the target if the horizontal offset between the camera direction and the target reaches zero. Thus the population only has to worry about the pan velocity and can merely set the tilt velocity to zero. Figure 1 shows a typical run of 200 generations with a single population containing 50 agents.

These results demonstrate that the basic learning mechanism can at least learn something. The next stage is to learn the full task and we hope to be able to do this by freezing the converged 'pan' population above and then training an additional population to take care of the tilt component. We also wish to compare this with evolving the whole task in one go using one or two populations.

Eventually we hope to be able to use various SE techniques to tackle the considerably harder problem of tracking moving targets (as opposed to bright ones).

Another thing we want to be able to do is to allow agents to influence each other by both allowing agents to take input from other populations, and allowing them to send inhibition or excitation to other populations. This will make the credit assignment even more complex and we hope to use a bucket-brigade technique [5] to allow fitness to flow between populations.

7 Related Work

There has been quite a lot of work on hierarchical training of skills in order to speed up learning in robots, *e.g.* [2, 1, 8]. Almost all of it involves an explicit decomposition of the controller itself however, which is something we hope to avoid.

The idea of emergently co-evolving different specialists within a controller population is quite old. Much of the work attempts to evolve such cooperation within a single population, *e.g.* classifier systems [6] and 'symbiotic neuro-evolution' [10]. Potter *et al.* [12] presents quite a similar architecture to our own in which multiple populations are used to evolve different specializations in a simple simulated robot foraging task, although they see the potential more for *avoiding* human design input rather than supporting it.

The idea of using a 'validity' to decompose the state space into areas of different priority for different agents is similar to Mark Humphry's 'W-Learning' [7].

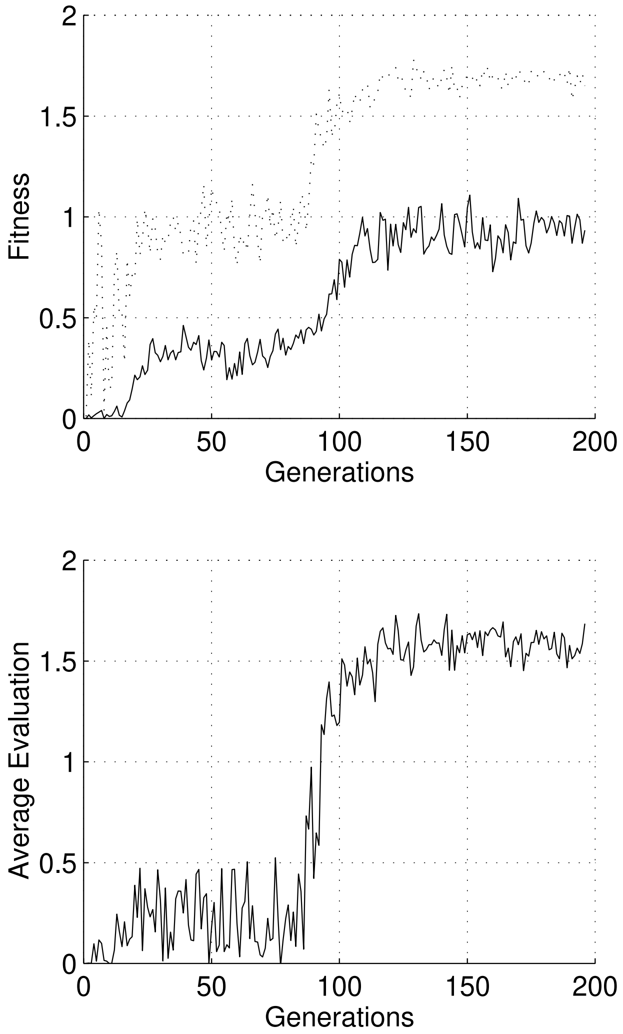

Figure 1: Results: The top graph shows the maximum and mean fitness of the population during a typical run. The bottom graph shows the average evaluation received during the test phase at the end of each generation during which an agent is picked randomly from the top 10% of the population each robot cycle for 100 cycles. The optimum average evaluation is about 1.7.

One of our main aims is to train robots to perform complex visual behaviours. The evolutionary robotics group at Sussex University has similar aims *e.g.* [4] and has had some success evolving neural network controllers, although again they use a population of monolithic controllers.

References

[1] Jonathan H. Connell and Sridhar Mahadevan. Rapid task learning for real robots. In Jonathan H. Connell and Sridhar Mahadevan, editors, *Robot Learning*, chapter 5, pages 105–139. Kluwer Academic Press, 1993.

[2] Marco Dorigo and Marco Colombetti. Robot shaping: Developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkley, CA 94704, April 1993.

[3] J.J. Grefenstette and A.C. Schultz. An evolutionary approach to learning in robots. In *Proc. Machine Learning Workshop on Robot Learning*, New Brunswick, NJ, 1994.

[4] Inman Harvey, Phil Husbands, and Dave Cliff. Seeing the light: Artificial evolution, real vision. In D. Cliff, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proc. 3rd Int. Conf. Simulation of Adaptive Behavior*. MIT Press, 1994.

[5] J.H. Holland. Adaptive algorithms for discovering and using general patterns in growing knowledge bases. *Int. Journal of Policy Analysis and Information*, 4(2):217–240, 1980.

[6] J.H. Holland and J.S. Reitman. Cognitive systems based on adaptive algorithms. In D.A. Waterman and F Hayes-Roth, editors, *Pattern Directed Inference Systems*, pages 313–329. Academic Press, New York, 1978.

[7] Mark Humphrys. Action selection methods using reinforcement learning. In *From Animals to Animats 4: Proc. 4th Int. Conf. Simulation of Adaptive Behavior*, Cape Cod, Massachusetts, USA, September 1996.

[8] Long-Ji Lin. Hierarchical learning of robot skills by reinforcement. In *International Conference on Neural Networks*, 1993.

[9] Maja J Mataric. Reward functions for accelerated learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference*, San Fransisco CA, 1994. Morgan Kaufmann Publishers.

[10] David E. Moriarty and Risto Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, (22), 1996.

[11] Stefano Nolfi and D Parisi. Evolving non-trivial behaviours on real robots: An autonomous robot that picks up objects. In M. Gori and G. Soda, editors, *Proceedings of the fourth congress of the Italian Association of Artificial Intelligence*, pages 243–254, 15 Viale Marx, 00137 - Rome - Italy, 1995. Springer-Verlag.

[12] M.A. Potter, K.A. De Jong, and J.J. Grefenstette. A coevolutionary approach to learning in sequential decision rules. In *Proc. 6th Int. Conf. on Genetic Algorithms*, Pitsburgh, July 1995. Morgan Kaufmann.