Robot Shaping — Principles, Methods and Architectures *

Simon Perkins Gillian Hayes

March 8th, 1996

Abstract

In this paper, we contrast two seemingly opposing views on robot design: traditional engineering methods, and automated methods using learning and evolutionary techniques. We argue that while each has its advantages, it is likely that significant progress in robotics could be made using a suitable hybrid of the two philosophies. Of course many successful systems already do this, but they do so in rather ad hoc ways. In contrast, we are attempting to propose a principled way in which engineering design can be combined with evolution, which we call *shaping*.[1] We present some general principles that we believe should underlie shaping, and follow this up with a set of methods that might be used to put those principles into practice. We then discuss and justify a novel neuro-evolutionary architecture that we believe to be particularly suitable for use in a shaping context. Finally we set out our goals for the future of this research.

1 Introduction — Engineering vs. Evolution

In the beginning, people designed robots using traditional engineering methods. Very approximately the engineering approach to robot design can be broken down into the following steps:

1. Specify the task to be carried out in a declarative fashion (*i.e. what* has to be done rather than *how* to do it).
2. Analyze the system (the robot plus its environment) in some detail.
3. Based on this analysis, use creativity and intuition to design a robot and controller capable of meeting the specification.
4. Test, and repeat cycle until satisfied.

Unfortunately robots turn out to be very difficult to design by hand. Engineering works well where the interaction between the thing you design and its external environment is well known and can be described fairly simply. In contrast, the interaction between a sophisticated robot and an unknown, or at least uncertain and varying environment is very complex.

More recently then, researchers have suggested that we should get robots to 'design themselves', using automated search techniques. There are a huge number of different implementations, using neural networks, evolutionary algorithms, dynamic programming etc. as well as various combination of these, but many fit the following general pattern:

1. Specify the task to be carried out in a declarative fashion. Often, the task is specified indirectly using an evaluation function.
2. Design some flexible adaptive control architecture that is believed to be capable of performing the specified task (with a suitable choice of parameters, initially unknown).

*Presented at the Workshop on Learning in Robots and Animals at AISB'96, University of Sussex, April 1–2 '96

[1] Note that this term has been used in a similar context before by M. Dorigo and others.

3. Heuristically search the parameter space of this architecture until a satisfactory solution is found.[2]

The variety of methods falling into this category is enormous, but what they all have in common is that they produce systems that contain some elements which are not supplied by the designer, and which are produced as the result of some search or optimization process. This includes most methods that involve learning or evolution.

The obvious advantage of having these 'automatically-designed' elements is that the designer doesn't have to know how they work, only how they can be trained, which avoids traditional engineering problems of having to model the world and then calibrate an implementation to fit the model.

Unfortunately, virtually all pure search methods scale very badly with the complexity of the task to be learned. Complex tasks usually require complex controllers, and these in turn require lots of parameters to specify them. In addition, most search methods rely on 'heuristics' such as hill-climbing or GA-style recombination, and these usually work badly where the parameter space is rugged and where evaluation is sparse. Both these factors mean that trying to learn complex skills from scratch in a reasonable time using general learning methods is next to impossible.

Increasingly then, people are suggesting that what we need is a hybrid of the two approaches. Specifically: how can we use domain knowledge supplied by humans to speed up or bootstrap search-based methods? Of course, in practice this is what people have always done — most learning systems that actually work are heavily tweaked so as to bias them towards finding solutions in the particular domain in which they are applied. However, at the moment this is done on a very ad hoc basis. We would like to explicitly recognize the tradeoff between engineering and search, and apply ourselves to 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 *shaping*.

2 Shaping Principles

Shaping is a term borrowed from behavioural psychology that has been used in a robotics context by a few researchers (notably Marco Dorigo [6]) to refer to the process of humans actively structuring learning in robots. We use the term to refer both to a set of principles that define our views on what sorts of knowledge humans should supply to learning robots, and also to a set of methods we have proposed to put the principles into practice. Other people use the term to mean slightly different things so our use isn't definitive. The words 'search', 'learning' and 'evolution' are more or less interchangeable in the following descriptions.

The major principles of our shaping approach are:

1. The job of the human designer is to determine the high-level structure of a task in terms of simpler tasks and skills.
2. The job of the search algorithm is to implement those simple tasks in the robot architecture.
3. The designer shouldn't have to know what the search algorithm is doing.

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

[2] The term 'heuristic search' is commonly associated with symbolic systems, but here I use it in a wider sense to include such 'search' algorithms as Backpropagation and genetic algorithms, which respectively use the 'heuristics' of gradient-descent and genetic recombination.

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.

As an analogy, think of training an animal to carry out a complex task. The human trainer has no idea how the animal's brain works, yet by devising a suitable training programme based on high-level analysis of the task to be carried out, and observations of the animal's behaviour, the animal can be taught how to carry out the task. It is our hope that this will prove to be a good way of getting robots to do complex tasks as well.

In addition to these major principles, there are some other features which we think are important:

- **Behaviour should be built up in an incremental fashion.** A search algorithm generally works best when its starting point is not to far from the intended goal. Therefore we can only expect to use learning to make a relatively small change in the parameters of our controller. In order to go from a controller that knows nothing to a controller capable of performing a complex task, we must follow a path of gradually increasing competence where each step on the path is of tractable size.
- **The training process should be interactive and iterative.** Designing such an incremental path is likely to be quite hard, and the chances of getting it right first time for a complicated task are small. Therefore we need a system that allows the designer to easily adjust this path, as and when the shortcomings of the originally proposed route become apparent.
- **The tradeoff between evolution and engineering should be easily adjustable.** In most cases, we would like the evolution to do as much of the work as possible. Not only is this less work for the designer, it also helps ensure that the designer's prejudices about how a task should be carried don't prevent the robot finding a simpler solution if one exists. It makes sense then to be slightly optimistic at first about how much evolution will achieve by itself, and then gradually introduce more and more engineering assistance at points where the robot is observed to have problems producing a particular increment in competence.
- **Real robots provide the only real test (but simulation is a useful tool).** Ultimately we are interested in developing methods that can be used to produce useful real robots. Therefore whatever system we use must eventually produce real robot controllers. So at least some learning must happen on the real robot. However, in many cases an approximation to a real solution can be obtained using simulation, which may be easier to work with — in particular more useful evaluation functions can often be derived in simulated worlds. Additionally, several researchers (*e.g.* [14]) have also demonstrated that robot controllers can be trained in simulated environments until they do well there, and then transferred to the real robots for final training. The initial simulation training means that the real world training phase takes much less time than if all the training was carried out in the real world.

3 Shaping Methods

Based on the above principles we propose the following general 'shaping methods'.

3.1 Task Decomposition

Probably the most important aspect of shaping is the process of breaking a complex task down into simpler sub-tasks (and the sub-tasks into sub-sub-tasks and so on). It is this decomposition that largely defines the incremental path that the robot will follow in learning a complex task — each lowest-level sub-skill represents one step along the path of increasing competence.

Task decomposition has been used by a number of researchers to make learning of complex tasks tractable (*e.g.* [4, 6, 11]). Like them, the decomposition we use has much in common with behaviour-based approaches as advocated by Brooks [2] and others. However, our approach differs in that we not specify an explicit behaviour hierarchy (*e.g.* we do not specify that this behaviour coordinates that behaviour using this interface). Instead the designer designs an incremental training programme under the influence of which it is hoped the learning algorithm will develop a suitable (hierarchical if necessary) controller.

3.2 Training Explicit External Representations

Our controller will almost certainly need to use some sort of internal representations, and there will almost certainly be occasions, when, as a designer, we would like to encourage the robot to represent some fact internally — *e.g.* that the robot has received a warning that its batteries are running low. However, we have also said that the designer should not have to know about internal representations. How can we resolve this dilemma?

We get round the problem by allowing the designer to provide reinforcement on the basis of *external* representations. The robot controller has access to a battery of dummy output actuators called *representors*, which the controller can switch on and off just like the robot's other actuators. The designer can specify, for example, that representor 5 is to be set to a value of 1 when the low battery warning is received, and the robot can be accordingly trained. This representor has no actual effect on the functioning of the robot, but by carrying out this training, the designer has ensured that there is at least some machinery in place which is capable of representing the information the designer deemed important. Hopefully this machinery will come in useful in some future learning phase, but the robot is not forced to use this machinery in any way.

3.3 Providing Progress Estimators and Heterogeneous Reinforcement

In most learning schemes there is the concept of an evaluation phase, where the trainer supplies some information as to what the robot should have been doing (supervised learning) or how well it has been doing generally (reinforcement learning). With almost all schemes, the learning task becomes exponentially more difficult as the time between evaluations increases. This is largely due to a problem of credit assignment — if the robot has to carry out a sequence of many actions before it receives some evaluation, it has to try and work out which of those actions were relevant to receiving the evaluation, and what it was about those actions that made them relevant,[3] and this is difficult.

So our optimization task will be made much easier if we can provide evaluation frequently. This is not as easy as we would like — in particular in reinforcement learning frameworks, it is usually difficult to definitively say whether the robot is doing well until it has finished its task. However, wherever possible, the designer should make an attempt to provide reinforcement as often as possible (using a 'progress estimator'). This often means that inaccurate reinforcement will be applied so the system must expect noisy evaluations and be able to cope with this. It is also often possible to give reinforcement for achieving parts of a task and this will also help (*i.e.* a 'heterogeneous reward function'). [12] provides some discussion of these terms.

3.4 Providing Enriched Learning Opportunities

When the robot is starting to learn a complicated task, it will likely initially not be very good at it. This coupled with delayed reinforcement, means that the robot may never even get as far as achieving its goal task, which in turn means that it will never get the full reinforcement which would help it to get there in the first place. This dilemma can be resolved by structuring the environment during the early learning phases so that it is very easy for the robot to achieve its goal. For instance if we were trying to train a robot to search for and dock with a charging station, we could start by putting the robot right in front of the station. When it has got the hang of that, we can start the robot a bit further away, and so on. This method is probably the closest to the psychological meaning of the term shaping, and is closely related to the process of task decomposition. Simulation can also be used to generate environments that are unnaturally rich in learning opportunities.

[3] These problems are known as the temporal credit assignment and the structural credit assignment problems respectively.

4 Shaping Architectures

4.1 Architectural Goals

In addition to being compatible with the shaping methodology presented above, there are a number of other goals that we have set for our architecture. These include:

- We would like our robots to be capable of performing non-Markovian tasks where the decision about what to do next cannot be made purely on the basis of what the robot can sense now. What this means is that the robot must maintain some sort of internal state itself — effectively a memory of previous sensing and actions.
- With many robotic learning systems, it is assumed that the robot runs on a fixed cycle at a single timescale. Every cycle the robot chooses a single action from a fixed repertoire and the cycle repeats at uniform intervals. Actions are normally assumed to have instantaneous effects. In the real world, actions can last for arbitrary lengths of time and may be selected at any time. In addition, many tasks require different actuators to be controlled simultaneously. So we need an architecture capable of managing this.
- Our robot controller will receive sensing data from a variety of sensors and provide control signals to a variety of actuators. In many robotic machine learning systems, all this data is lumped together into a uniform sense vector and a uniform action vector respectively. We feel that this is very wasteful and destroys a vast amount of structure inherent in the data. In particular we are interested in visual data, which by the nature of the imaging process is two dimensional. Turning this into a one dimensional vector throws away a lot of important topological information. Our solution to this problem is the idea of 'neuron species' described in the next section.

4.2 A Neuro-Evolutionary Architecture for Shaping

4.2.1 Justifications

There are probably many different systems that could be used as the basis of a shaping system. However, given the constraints expressed above, we have selected a neural network architecture coupled with an evolutionary architecture similar to a classifier system. Here we discuss some justifications for choosing this type of system.

For most real robotic tasks, we don't know exactly what actions the robot should be performing in order to carry out its task. If we did then we could just program the robot directly with that knowledge! Therefore, supervised learning techniques are probably out. We could potentially have used supervised learning as a component of the architecture (*e.g.* to learn a model of the environment as in Sutton's DYNA architecture [18]), but we didn't want to use such a rigid controller architecture.

This puts us in the realm of reinforcement learning. Many reinforcement learning systems are based around temporal difference methods (*e.g.* [17]), but these are usually applied to discrete domains where at each time step, one of a fixed set of actions is to be chosen. This again seemed limiting.

Reinforcement signals are also often used in conjunction with evolutionary algorithms — the reinforcement signal is used as the evaluation function. Evolutionary algorithms have the advantage that they can be made to work with virtually any architecture which gives us the flexibility we need.

In a robotics context, evolutionary algorithms are often combined with neural networks (*e.g.* [8, 15, 7]). Neural networks are extremely flexible, computationally powerful and can deal with sub-symbolic data such as is supplied by the robot's sensors and expected by its actuators. By making the neural network recurrent (*i.e.* we allow loops) and giving the neurons temporal characteristics, we can also have a neural net controller that is capable of having internal state, and that is capable of working at a variety of timescales. Our architecture is therefore based around neurons and uses an evolutionary algorithm for learning.

Most systems for evolving neural networks evolve a population of networks, each one of which is a potential controller in itself. Typically, each network in the population is evaluated on the robot for a short interval, one at a time, until the entire population has been evaluated. Then the best few are selected for reproduction, new networks are produced, and the process continues until a satisfactory solution is found.

This approach has several problems. Firstly, if the robot's environment is not a simulated one, then it is impossible to start each evaluation with the robot in exactly the same position for all the neurons, which can lead to unfair treatment of some neurons. Secondly, since the population can consist of several hundred networks the evaluation process can take an enormous amount of time, particularly if we're letting each network run for a while to try and give it a fair chance of proving its worth. Thirdly, our networks have internal state, which may represent useful things the network has remembered about the environment. When we breed networks and combine them together, what do we do about this internal state? Do we discard it, in which case we may be throwing away useful information, or do we try and preserve it in which case it almost certainly won't make much sense in the new combined network? Fourthly, one of the important points about shaping is the idea of incremental development of the controller. This implies that each optimization phase takes as its starting point the solution from the previous phase. However, GAs normally produce just one solution at the end of a run. Where do we get the *population* we need for the start of the next run?[4] We might try to recreate a population using the previous solution as a seed but to do so seems unlikely to work — we either end up with a population with very little genetic diversity, which is bad for the GA, or we mutate the previous solution so much that we risk destroying the behaviour we have just learned. And finally, since we have no idea which parts of the network are responsible for the behaviour we've just learned, we have no way of protecting that behaviour during subsequent evolution, and it is important that previously learned behaviours should remain in the controller ready to be used if necessary by other behaviours.

Because of all these problems we have opted for an evolutionary algorithm modelled on the 'Michigan-style' classifier system [10]. Rather than evolve a population of networks in an attempt to find the best one, we instead evolve a population of *neurons* that between them compete and cooperate to produce the emergent behaviour of the robot. Now we only have one controller (made up of the population of neurons), which remains continually in the robot. So there are none of the problems associated with 'switching brains' described above. Evolution of the total network happens by a gradual process of replacement and mutation of the neurons that make it up, and this is suited to the incremental development which is the hallmark of shaping.

In order to evolve such a network we must be able to evaluate individual neurons in order to decide which ones to breed from and which ones to discard. However, the trainer can only evaluate the observed behaviour of the robot. We are not allowed to look inside the robot's controller to see what is going on, and so some automatic system for assigning credit to individual neurons based on the global assessment is needed. This is provided by an analog to the bucket-brigade algorithm used in classifier systems [9]. A side effect of this credit assignment is that we now have a much better idea which neurons are responsible for producing the behaviour that has just been learned, and therefore we can take steps towards protecting those neurons from the ravages of evolution so as to preserve them for use in future behaviours.

The final component of this system is the notion of neuron species. The full network is made up of several different species of neuron, which have different connectivity characteristics and different mutation operators. In particular, there is at least one species for each type of sensor and at least one for each type of actuator. This allows us to create, for instance, vision neurons that know about the two dimensional topology of the vision sensor. One species of visual neuron might always connect to a 3×3 square in the vision array (we know from both machine and biological vision research that such 2D local neurons can be very useful), and might have a mutation operator that includes moving this square around the image. This makes use of the 2D topology of the vision array in a way that seems sensible. Other neurons species can take advantage of other special sensor and effector properties in a similar way.

[4] Even if we take the best few solutions from the previous stage they will likely be very similar owing to the phenomenon of convergence in GAs.

4.2.2 Architecture Details

Figure 1 shows how neural network fits together, and how the neural network is connected to the rest of the reinforcement and evolutionary system. Lack of space prevents us describing the system in much detail, but here is a summary of some important points.

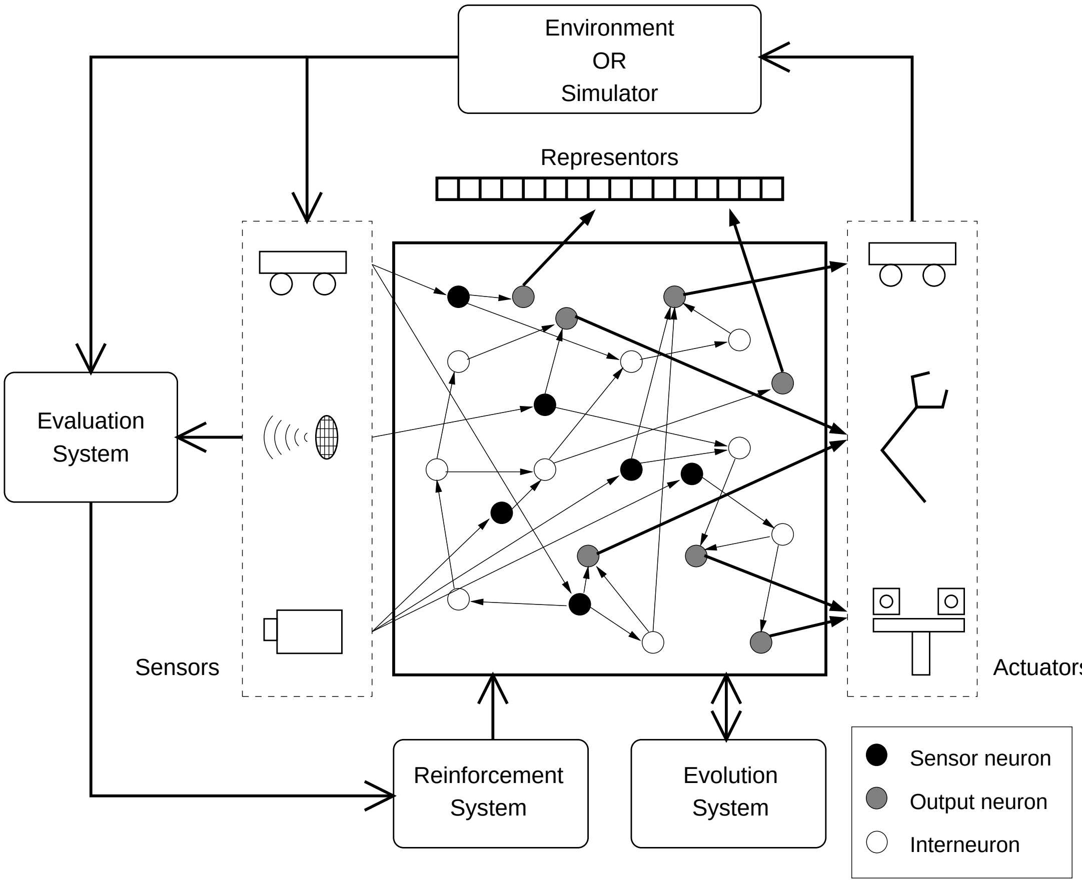

Figure 1: A neuro-evolutionary architecture for shaping.

Physical Structure The robot's controller is composed of a population of neuron like units, which compete and cooperate to produce the overall behaviour of the robot. The neurons are not arranged in layers or groups. The neurons belong to various different species which have slightly different properties. Some can only connect to particular sensors, some to particular actuators, and some can only connect to other neurons.

Since the neuron population will change as individual neurons reproduce and die off, the connectivity of the brain cannot be specified globally. Instead, the neurons are arrayed on a two-dimensional grid called, simply, the 'Grid'. Each neuron connection then specifies a point in the Grid to which it would like to connect — the connection is actually made to the neuron closest to that point.

This connectivity scheme is used to ensure sparse connectivity between neurons, which assists credit assignment. In conjunction with the evolutionary system, the aim is also to evolve regions of specialization on the Grid so that neurons of similar function are found close together.

Neurons The neurons are reasonably conventional sigmoid units, except that their output is a function of time as well as the current values of their inputs. This allows neurons to respond to events at a variety of timescales. Neurons in fact have two outputs, a *value output* and a *confidence output*. Correspondingly, they also have two sets of inputs. The value output is the only thing that other neurons can directly see, but confidence is also propagated through the network and is used to determine which actuator neurons get to fire.

Credit Assignment As mentioned above, a 'bucket-brigade' algorithm is used to apportion credit to different neurons. Whenever reinforcement is received by the system, it is initially distributed to those actuator neurons that were most recently active. These neurons then propagate reinforcement back to neurons that they are connected to. The amount of reinforcement propagated back depends largely upon the correlation between the confidence of the actuator neuron and the confidence of the neuron that connects to it. If the actuator neuron is highly confident at the same time as a connecting neuron is confident, then a high percentage of reinforcement (positive or negative!) will be transferred. Reinforcement is then propagated back to other neurons in a similar fashion.

Evolutionary System Neurons that accrue a high degree of reinforcement are selected for reproduction. We use a tournament selection procedure so that neurons compete against neighbouring neurons on the Grid. When neurons reproduce, they do so by mutation. The new neuron appears in a neighbouring location on the Grid. These two features are intended to encourage different specializations to appear in different regions of the Grid. We do not use the cross-over operator. One of the primary reasons for using crossover in traditional GAs is to mix together different parts of solutions (schemas) to form a more complete solution. However, in our system, each neuron represents a partial solution, and the total solution is made up of many different neurons each with their own specialization. We think it is unlikely that any advantage would accrue from using crossover between neurons with different specializations, and crossover between similar neurons would probably produce an effect similar to mutation.

Relationship with Classifier Systems The architecture was partly inspired by an evolutionary architecture used for learning called a classifier system [10]. A classifier system also consists of a population of entities which compete and cooperate to produce the total behaviour of the system. Similarly a bucket brigade algorithm is used to assign credit to individual classifiers, and a genetic algorithm is used to breed new classifiers from high-scoring parents. Traditionally, each classifier consists of a condition/action pair, each represented as a binary string. Classifiers can be triggered by environmental input (*e.g.* robot sensors) or by actions from other classifiers, and can also trigger environmental output (*e.g.* robot actuators). Our architecture is partly an attempt to generalize classifier systems to having fuzzy conditions and actions. The confidence of a neuron indicates the degree of matching to its 'condition', and the value represents its 'action'.

The Robot For our shaping experiments we will be using a Real World Interface, Inc. B–21 mobile robot. This has three on-board PCs for computation and is equipped with a pan/tilt head and camera for vision work, as well as a variety of sonar, infra-red and touch sensors.

5 Future Work

At the time of writing the architecture described above is in the process of being implemented, and so we are unable to present concrete results. Our first experiment will involve learning a very simple motor control task using immediate reinforcement, such as sinusoidally waving the robot's pan/tilt head from side to side at a specified frequency. This will allow us to see if we can do any learning at all!

After getting the initial learning experiments finished the next stage is to attempt to train a task that requires the assistance of shaping techniques in order to find a solution. Vision based tasks are particularly hard for learning systems owing to the large amount of data that must be handled, and so would provide a good challenge for our methodology. As a first stab, we will try to train the robot to track small fast moving objects in its field of view by moving its pan/tilt head. We think this task represents a good choice because we know that with a moderate amount of effort we can probably engineer a solution. On the other hand the task is more complicated than anything we have seen a general learning system handle. Therefore it provides a good testbed to see if shaping techniques can allow us to train robots to perform complex tasks with less trouble than it would take to engineer them.

6 Related Work

There has been quite a lot of work on hierarchical structuring of skills in order to speed up learning, but relatively little of it has dealt with real robots and controllers that can work at multiple timescales, with complex actions and non-Markovian environments (as ours is intended to do). Probably the closest work is that by Dorigo *et al.* [6], who also uses the term shaping to describe a process of behavioural decomposition by the designer coupled with an evolution element. In his case the learning element is a reasonably conventional learning classifier system using binary messages, which doesn't have the computational power of our neural network, but which may be simpler to learn with. Colombetti, Dorigo and Borghi [3] have also developed a general training methodology which they call behaviour engineering, which has some similarities to our own shaping methodology, but doesn't take as strict a stance as us about the designer not having to know about how behaviours communicate.

Bonarini uses fuzzy classifier systems to control robots, although so far I have only seen results from simulation [1]. The fuzzy systems are more computationally powerful than standard classifier systems but as far as I can tell they do not yet make use of internal messages or multiple timescales. I have also not seen them being used with task decomposition.

R.E. Smith [16] draws a strong analogy between classifier systems and neural networks. However, both his classifiers and his neurons are binary valued. It does represent one of the few examples of a population of neurons being evolved, although the neural net structure is very rigidly fixed compared to ours, and only works in simulation.

Another group to evolve a population of neurons is Moriarty and Mikkulainen [13] who use 'symbiotic evolution'. Their system seems to rely on neurons cooperating in a very weak fashion — essentially they cooperate by ignoring one another. Our system encourages much more active cooperation.

Another popular alternative to neural networks/GAs is Q-learning [19]. This is most commonly used in very simple discrete worlds, but it has been used to investigate the business of task decomposition in such worlds (*e.g.* [11]).

Connell and Mahadevan [4] have also successfully used Q-learning in conjunction with the subsumption architecture and task decomposition to accelerate learning on a real robot. Again the interface between behaviours had to be fairly strictly defined.

Dorigo (again) and Bersini [5] have shown that the bucket-brigade algorithm used with classifier systems is extremely similar to the temporal difference method used in Q-learning.

One of our main aims is to train robots to perform complex visual behaviours. Another group with a similar aim is based at Sussex university. While they are also using temporal neural networks in conjunction with genetic algorithms and real robots, their approach differs considerably in that they are evolving populations of networks, rather than populations of neurons. They have a special gantry robot setup that makes it feasible for them to run a 'mobile robot' for the very long times [8] that it takes to evaluate populations of networks.

References

[1] Andrea Bonarini. Evolutionary learning of fuzzy rules: Competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*. Kluwer Academic Press, 1995.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. AI Memo 864, AI Laboratory, Massachusetts Institute of Technology, 1985.

[3] Marco Colombetti, Marco Dorigo, and Giuseppe Borghi. Behaviour analysis and training — a methodology for behaviour engineering. *IEEE Transactions on Systems, Man and Cybernetics*.

[4] 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.

[5] Marco Dorigo and Hugues Bersini. A comparison of q-learning and classifier systems. In D. Cliff, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proceedings of the 3rd International Conference on the Simulation of Adaptive Behavior*, Brighton, UK, 1994. MIT Press.

[6] 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.

[7] D Floreano and F Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. In J.-A. Meyer D. Cliff, P. Husbands and S. Wilson, editors, *From Animals to Animats 3: Proceedings of the 3rd International Conference on the Simulation of Adaptive Behavior*, Cambridge MA, 1994. MIT Press.

[8] 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: Proceedings of the 3rd International Conference on the Simulation of Adaptive Behavior*. MIT Press, 1994.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] Stefano Nolfi, Dario Floreano, Orazio Miglino, and Francesco Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. Technical Report PCIA-94-03, Institute of Psychology, CNR, Rome, 15 Viale Marx, 00137 - Rome - Italy, May 1994.

[15] 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.

[16] Robert E. Smith and H. Brown Cribbs,III. Is a learning classifier system a type of neural network. Technical report, University of Alabama, March 1994.

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

[18] R.S. Sutton. Integrated architectures for learning. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, Austin, Texas, 1990. Morgan Kaufmann.

[19] C.J.C.H. Watkins. *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, England, 1989.