Evolution of the layers in a subsumption architecture robot controller

Julian Togelius

Dissertation for the Master of Science in Evolutionary and adaptive systems

University of Sussex at Brighton

September 1, 2003

julian@togelius.com

+46-705-192088

Abstract

An approach to robotics called layered evolution and merging features from the subsumption architecture into evolutionary robotics is presented, its advantages and its relevance for science and engineering are discussed. This approach is used to construct a layered controller for a simulated robot that learns which light source to approach in an environment with obstacles. The evolvability and performance of layered evolution on this task is compared to (standard) monolithic evolution, incremental and modularised evolution. To test the optimality of the evolved solutions the evolved controller is merged back into a single network. On the grounds of the test results, it is argued that layered evolution provides a superior approach for many tasks, and future research projects involving this approach are suggested.

Acknowledgements

Many thanks to my supervisor, Ezequiel Di Paolo, for enduring my severe and prolonged indecisiveness and doubtfulness about the topic of this project, and providing good advice on every new idea I came up with. Thanks to Dyre Bjerknes for essential discussions at every stage of the project, and to Helena Marttinen for many things, such as helping me have a life in spite of the project. Many have supported me in various ways, for example by putting up with my tiresome ramblings about robots and time pressure; these include Ludvig Friberger, Johan Persson, Magnus Kihlgren, Jesper Adamsson, Marie Gustafsson, and my parents.

Table of contents

Background

I will here describe the approach to evolving robot controllers I am taking, spell out the theoretical arguments in its favour, and try to counter some conceivable objections to it. I will try to prove its worth for both science and engineering. But first I will situate it in its interdisciplinary context by sketching the research programmes I am building upon.

Evolutionary robotics

Evolutionary robotics is a relatively novel approach to the relatively mature field robotics, which itself is quite recent compared to the age-old dream of building intelligent machines. The basic idea is simple enough: if natural evolution could come up with marvellous stuff like intelligence starting with lifeless matter, could we not replicate this feat in man-made systems, such as computers? If we could, we would have not only have a radically new engineering methodology with enormous practical applicability. We would also have a powerful new tool for studying many important scientific questions. For example, studying under what conditions a particular adaptation (e.g. learning) can evolve in artificial evolution could help us understand under what conditions it evolves in nature, which in turn could help us understand the historic environment of a specis, such as Homo Sapiens.
Additionally, the creativity inherent in the evolutionary process might be very interesting in its own right. To use an analogy popularized by Dawkins, it is quite common for that blind watchmaker to put the cogs and springs together in ways that no human watchmaker would conceive of, and thus put our own design solutions in a new light.
Research in evolutionary robotics has been going on for about ten or fifteen years now, usually with an amalgam of the above mentioned motivations. A plethora of methods have been used, but the most common way of doing ER (as evolutionary robotics will often be abbreviated in this text) is this: a set of artificial genomes, called the population, is created. Each genome is a data structure specifying the configuration of a robot controller, and sometimes a robot morphology as well[1].

[1] A commonplace distinction to make here is that between the genotype, in this case the genome, and the phenotype, in this case the robot controller. Fitness is evalueated for the phenotype, but what is reproduces is the genotype. It might be argued that the population should be a population of phenotypes and not of genotypes, but I have omitted such considerations from the main text to keep the description brief, and it really doesn't help much in this case.

Usually the robot controller is a neural network connected to the sensors and motors of the robot, and the information in the genome is used to specify the synaptic (inter-neuron connection) weights of the network. Then, evolution takes place. All genomes are evaluated, which means that robots with genetically specified controllers are tested on a task and genomes are scored according to how well the controllers they specified did on the task. Good genomes are kept, and bad genomes are replaced with modifications or combinations of the good genomes, and the process is repeated a number of times or until good enough performance has been reached.

Incremental and modularised evolution

This description is very general and there are a lot of implementation details that has been subject of research. Generally speaking, it turns out that a large part of the secret behind making artificial evolution work is in defining the task and its scoring – the fitness function, which must be smooth. For example, if the task is for the robot is to move around in a circle, the performance of the imperfect genomes must be scored in a continuum, where more circular motions get higher scores; straight lines get a low score, curved lines somewhat higher, and so on. With a scoring scheme, on the other hand, where a robot moving in a perfect circle gets score one and all other behaviours zero, the desired behaviour is highly unlikely to evolve. Evolution must be encouraging.

While this seems a truism, it is often not that simple to design a smooth fitness function for a more complicated task. For many tasks, performance is not easily graded on a continuum. A solution that has been put to use in such cases is incremental evolution (Floreano & Urzelai 1999, Meeden, Gomez & Miikulainen, Blynel & Floreano 2003). In incremental evolution, the task changes throughout the evolution of the controller; when evolution has produced a controller good enough to reliably solve the task it has been set, the task is augmented and evolution can evolve solutions to the new task starting from what has already been evolved.

The advantage can be explained in terms of fitness landscapes: some problems, such as collecting coke cans, can have very rough fitness landscapes in themselves. If the fitness is some measure of the robot's proficiency in collecting cans, a robot that at least makes some effort to approach red things in its environment will score just as low as another that moves around haphazardly or not at all; none of them

collects any coke cans. It is hard for the intended behaviour to "get off the ground" With incremental evolution, the to-be can collector may initially be rewarded just for approaching red objects, and after this behaviour has evolved the fitness function changes as to reward actually picking them up, and so on.
Another technique that has been advocated is using more than one neural network, but stick with a single fitness function. Nolfi (1997) reported that dividing the network into separate "modules" improved evolvability of a robot controller. Calabretta et al. (2000) concluded that duplicating an evolved network and letting it differentiate its function somewhat from the original network, where the mechanism for dividing control between the two networks is also evolved, could improve evolvability even more[2]. An analysis of the strengths of modularization will be part of the discussion of layered evolution in engineering. It is worth noting that the experiments referred to in this section did not concern explicit functional division among modules; only emergent differentiation between modules was studied.

Evolving learning

Most early experiments in ER concerned reactive behaviours, or instincts. That is, an evolved controller had no internal state, so it implemented a proper mathematical function: for a given sensor input, its motors always gave the same output. Soon, though, it was realized that many interesting behaviours required some sort of internal state; it was also realized that the ability for a robot to learn might help the evolution of other behaviours as well, something that has been termed the Baldwin effect. So recently, many ER experiments have involved robots that change their response patterns over "lifetime" rather than evolutionary time, i.e. robots that learn (Tuci et al. 2003, Blynel & Floreano 2003, Yamauchi & Beer 1994). As the "perceptron" type of neural networks doesn't permit internal states, other network types have been used, notably the CTRNN (Beer 1995) and the plastic network (Floreano & Mondada 1996), both neurophysiologically inspired.
The one conclusion to be drawn from these experiments is probably that, well, learning is hard. Even when the learning task is something as simple as learning whether going left or going right is beneficial and there is a strong reinforcement signal it usually takes much "helping evolution on the way " by tweaking various

[2] This is reminiscent of the "Commitee of experts" model in traditional pattern recognition-oriented neural networks research.

parameters and ingeniously redesigning the fitness function to produce the right behaviour[3]. All learning studies I have seen also take place in extremely simple environments.

Scaling up

Sadly, the learning experiments referred to above is pretty much the state of the art in evolutionary robotics[4]. I believe it is fair to say that no one has yet evolved any behaviour that couldn't have been hand-coded without too much ado[5]. That's not to say there haven't been any interesting results; the ways in which these behaviours have been evolved and the ways in which they have been implemented in the neural networks are interesting indeed, and negative results are also results. But still, the scaling up of evolutionary robotics seems have lost its steam. And I am convinced that scaling up is necessary not only for the engineering side of evolutionary robotics, but also for the scientific side, as we won't be able to study evolved solutions to more complex tasks if we can't evolve them.

Behaviour-based robotics and the subsumption architecture

Evolutionary robotics is certainly not the only modern approach to robotics. In the mid-eighties, Rodney Brooks (1986, 1991, 2002) invented the subsumption architecture, and thereby gave birth to the active research field behaviour-based robotics (Murphy 2000), which more or less dominates modern academic robotics research. In this field, like in good old-fashioned AI robotics, robots are hard-wired and behaviours are pre-programmed. Here, I will briefly describe the principles of the classic subsumption architecture (sometimes abbreviated SA).

A subsumption architecture is organized into layers, where each layer is responsible for producing one (or a few) behaviours. The layers have a strict vertical ordering where higher layers always have precedence over lower layers; often, the behaviours controlled by the higher layers are more complex and "cognitive" than those controlled by the lower layers. Importantly, all layers except the bottom layer presuppose the existence of lower layers, but no layer presupposes the existence of

[3] Even so, some researchers (Tuci et al. Harvey 2003) think that ER experiments like theirs can contribute to learning theory.

[4] Although some (Cliff 2003) count Beer's (2000) recent experiments as the most advanced behaviour ER has come up with.

[5] Which is not to say that anyone could have hand-coded a *neural network* to do some things that neural networks have been evolved to make robots do.

higher layer; in other words, if the robot is built from the bottom up, each stage of the robot development is able to operate.
A good example of a subsumption architecture is Brooks' (1989) robot Genghis, which has six layers: stand up, involuntarily step, voluntarily walk, walk over obstacles, walk competently and chase prey. If all layers but the first is disabled, the robot can still stand up; if only first and second are active, the robot can move its legs to prevent it from falling over if someone pushes it, and so on. The totality of all layers provides for a lifelike behaviour where the robot chases anything that moves even if there are obstacles in its way. But every individual layer is quite simple, and totally useless without the behaviours underneath it working.
Albeit it has yet to fulfil the grand visions laid out in (Brooks 1991), not to mention those in (Brooks 2002), behaviour-based robotics has proven to be useful in many applications, and has also inspired biological and behavioural research (Prescott et al. 1999). At the same time, hand-wired robotics is limited to human creativity and design prejudices, which is a severe bias to scientific experiments and an unnecessary constraint on product development.

My approach: Layered evolution

I propose to merge central features of the subsumption architecture, which has to my knowledge only ever been hand-wired, into evolutionary robotics. In most of the research done under the heading evolutionary robotics the controller is evolved as a single neural network, evolved on the same task with the same fitness measure for all of the phylogenetic process. I call this approach monolithic evolution. There are deviances from this approach - for example, in the section on ER above I mentioned work done in incremental evolution, where the fitness function is changed as certain fitness levels are reached, and some work on emergent modular differentiation – but most work in ER uses monolithic evolution.
There has thus been little work done in breaking up the monolithic structure of controller and phylogenesis; one, to my knowledge untested, approach possible is organizing the controller as a subsumption architecture. Each layer consists of a neural network connected a subset of the robot's sensors and actuators. The layers are connected in a simple structure where higher layers can influence or subsume lower layers. If several layers are evolved together with a single fitness function we have what I call modularised evolution.

Brooks' original evolutionary metaphor, however, was that each new layer was a more recent evolutionary innovation. This metaphor can be adapted into ER by evolving the layers in sequence with different fitness functions. That is, once the first layer has reached a satisfactory fitness score, add the second layer on top of the first and evolve the second using the second fitness function until satisfied, etcetera. I call this approach layered evolution and I believe it holds good promises for scaling up ER.
The relations between these approaches can be drawn thus:

	1 layer	Many layers
1 fitness function	Monolithic evolution	Modularised evolution
Many fitness functions	Incremental evolution	Layered evolution

In this essay I will use layered evolution (sometimes abbreviated LE) to construct a controller for a mobile robot that performs phototaxis in a cluttered environment and is capable of lifetime learning, which is a task about as complicated as anything in ER. I will then systematically investigate if and how layered evolution works better than incremental, modularised, and monolithic evolution for this problem. Finally I will explore the idea of merging the evolved layers together, and seeing if further, monolithic evolution will increase fitness - thus addressing the issue whether there are limitations inherent in the layered strategy.

Why layered evolution makes better engineering

In this section, I intend to explain the engineering rationales behind evolving separate layers of a robot controller instead of evolving a monolithic network - why would layered evolution faster and more reliably produce better solutions to a given problem than the standard approach does?

Beyond incremental evolution

To begin with, evolutionary robotics researchers have long noted that there can be advantages to dividing up a problem into several parts. This is the rationale behind incremental evolution.

The advantages of incremental evolution carry over into layered evolution. As the separate layers perform different parts of the task, they must be evolved with different fitness criteria, as is the case with incremental evolution. One can indeed imagine a layered architecture for the can collector mentioned above (Brooks built one, though he didn't use artificial evolution) where one layer approaches red objects, another picks them up, and so on.

Network size: updating speed and search dimensionality

But my main thesis is that there are more advantages to layered evolution than those which it shares with incremental evolution. The most obvious of these added advantages is probably the reduction of necessary network size.

Neural networks are commonly measured in number of neurons employed. But a fundamental tenet of algorithmics is that the complexity of a computation be measured in the number of times its most frequent operation occurs (Cormen et al. 2001). When updating a neural network, the most frequent operation is propagating an activation level along a synapse, as there are more synapses than neurons in any interesting network topology. In fact, for n neurons in a fully connected network we have n^2 synapses. Something similar applies for most "balanced" feedforward networks. The point is that if we can break up a 10-neuron fully connected network into two 5-neuron networks, we are actually doing away with half of our synapses. Even if we split the 10-neuron network into three 5-neuron networks (perhaps we need one to control the other two) we are making a significant increase in updating speed. This gain will only be magnitudes larger as we get anywhere in the vicinity of the complexity of a real nervous system. E.g., a network of a million neurons would need a thousand billion synapses; a thousand networks each of a thousand neurons would total “only” one billion synapses.

And we are not only gaining in updating speed. In the process of evolution, each genotype can be considered as occupying a position in an n-dimensional genotype space, where n is the number of genes, which in most genetic encodings is directly proportional to the number of neurons plus the number of synapses. Decreasing the dimensionality of the search space is very important in many classic learning algorithms, for neural networks and otherwise (Bishop 1995). So by dividing one network into many, evolution of them can be sped up not only by making them quicker to update, but also by making the evolutionary search space smaller.

Don't change a winning recipe, and don't get caught up in the past

Even disregarding the number of synapses, there are direct efficiency gains in evolution to be made. In incremental evolution, once finished evolving one part of the final behaviour, the entire network is further evolved with a new fitness criterion - including the mechanism responsible for satiating the first fitness criteria. This wastefulness is necessary in monolithic networks as it is often impossible to separate out neurons responsible for any particular behaviour from those responsible for another (Ziemke 2000). Layered evolution has two advantages over incremental evolution from this perspective. The intuitive advantage is that we won't waste fitness evaluations on vainly changing parameters of the part of the network that is already fully evolved. The not so intuitive and not so well corroborated advantage is hinted at by a result in Nolfi (2003). He found that by strategically lesioning a monolithic network he could actually improve its evolvability. This is presumably due to that the state of a network that has been evolved for a while is not that synaptic weights assume "meaningful" values only where the synapses are contributing to mechanism and zero otherwise, as might be intuitively assumed, but that very few synapses have weights close to zero. Thus, when one mechanism has been evolved in a network, any weight change at any position in the network is likely to disturb that mechanism. But another mechanism can't evolve in the network without changing some weight values. Separation of the network into separate layers can be seen as very thorough strategic lesions, only allowing communication between the layers in a few places, and thus making it possible to evolve new functions without disturbing existing functions.

Several network types in one controller

As noted above, several different types of neural networks are used in evolutionary robotics research. In a monolithic network you are obviously stuck to using a single network type. Although some researchers use networks incorporating features from several network types (Di Paolo 2000), adding features to a network adds to both network updating time and dimensionality of the search space. A layered architecture, on the other hand, makes it easy to select the right type of network for each layer - a layer intended to exhibit a reactive behaviour needs only a

perceptron-type network, while others might need a CTRNN or plastic network. This will be demonstrated in my experiments below.

Good design principles

Finally, what I after all consider the main engineering argument for layered evolution is spelled out by Hod Lipson (2000). He claims that though evolution is very good at coming up with creative and indeed surprising solutions to many problems, these solutions are often quite bad from a classical engineer's point of view. Now, human design might not be all prejudice - much work has actually gone into researching engineering principles that allow, amongst other things, scalability[6]. Therefore it would not be a bad idea to bring some classical engineering principles into evolutionary robotics. Which is what I intend layered evolution to do. Modularity has always been fundamental to good software engineering, and one of the advantages that come with modularity is reusability. Today many evolutionary robotics researchers spend much time and effort at more or less replicating what others has done. Some time in the modular future we might have a repository of ready-evolved layers, free for anyone to include in his or her controller architecture and build upon.

How layered evolution could be used in science

The above section is concerned with engineering aspects of evolving layered architectures. But evolutionary robotics is often practiced as science rather than engineering. Therefore it is interesting to see how layered evolution can be used as a scientific tool.

As vintage evolutionary robotics can be seen as a special case (one layer, one fitness function) of layered evolution, all scientific investigations conducted with ER can as well be done using LE. In fact, many more scientific investigations using into the same and with the same methodologies as those currently done in monolithic ER can probably be done, as LE makes it possible to investigate more complex behaviours. Objections to using more than one layer in such investigations

[6] A good example is the internet, a process/structure that has scaled beyond the wildest imaginings of its originators, and which is organized in a strictly layered fashion – e.g Ethernet > IP > TCP > HTTP > HTML – where every layer permits several interchangeable protocols (e.g. IPv4 <> IPv6, HTTP <> FTP). As for the objection that we are constraining creativity, which will be dealt with below: within every layer of the internet there is ample room for creativity, as we have seen with the emergence of DOS attacks, worms, www.superbad.com, and IP telephony.

are answered in the next section. Here, I will focus on what additional opportunities for scientific investigation LE provides over classic ER, these being dependent on the scientific interpretation of the layered control architecture.

Neurophysiological layers

When Brooks proposed his SA, he based it on a "loose analogy" with natural evolution. The question whether vertebrate central nervous systems really are organized in a subsumption-like or at least layered architecture is addressed by Prescott et al. (1999). They conclude that such nervous systems indeed show many features of a layered architecture, with more or less clear ordering of the layers from lower to layer. For example, in many vertebrate animals it has been demonstrated that lesioning a higher layer leaves the behaviour from the lower layers almost intact; a cat without a cortex still searches for food when it's hungry and flees from danger, and a cat without most of its forebrain can still perform simple behaviours. What is not clear, however, is whether these architectural features are the product of phylogeny, the order in which the different brain parts were evolved, or if the brain's organized the way it is just because it is good design and most layers are, or could have been, evolved at the same time. Here artificial evolution might be of some help, by finding out whether a given function is easiest evolved and/or performs best when evolved as a monolithic or a layered structure, and thus what nature ought to have done. In this context, it is worth noting that in the main experiments of this paper the conditional phototaxis layer is in a way prepared for the addition of the associative learning layer, as it has an artificial input when it's being involved. Would results be different if there wasn't any artificial signal, and the new layer would have to figure out by itself how to control the lower layer?

Behavioural layers

But as the term behaviour-based implies, we need not necessarily go to the neurophysiological level to find interesting interpretations. The great ethologist Konrad Lorenz hypothesized that animal behaviours was functionally organized hierarchically, and out of this and other sources a movement in ethology called the behaviour systems approach has followed. Here, individual behaviours are organized into systems and subsystems with positive and negative influences on each other. It would certainly be interesting, and probably both practically and

theoretically useful, to start with an architecture modelled on a behaviour systems description of some organism and try evolving the constituent subsystems.

Parasitism, symbiosis and multicellularity

We might also move down the evolutionary ladder to find interesting parallels to LE. It is hypothesized that multi-cellularity started with parasitic and/or symbiotic relations between very simple organisms. In the main experiments of this paper, the conditional phototaxis and obstacle avoidance layer are certainly working in symbiosis with each other, while the associative learning layer can be seen as parasitic on the other two as it often forces the robot to avoid the closest light source and go for another. Thus evolving the different layers at the same time but with individual and even conflicting fitness criteria might throw some light on issues of organism interaction.

In sum, I believe there are a number of plausible interpretations of layered architectures, and a fair deal of research with these interpretations in mind could only be done with an evolutionary methodology rather than Brooksian hand wiring.

Objections answered

The reader might at this time have come up with a few objections to the approach I propose. Sure, it might be easier to create complex control systems with my methodology, but in moving from the received way of evolving integral controllers to evolving separate layers, aren't we missing something out? Do all advantages usually ascribed to evolutionary robotics transfer well to layered evolution? In this section I will address some worries that I suspect may be elicited by my approach.

Unconstrained design

First, it might be objected that explicitly designing the macro architecture of the controller interferes with one of the basic tenets of evolutionary robotics, unconstrained design. Evolution should be set to work on it's own without any human design-imperialist attitudes forcing the prejudices of our species on the innocent robot - only then can evolution's creativity blossom. I think there's something to this objection, but not too much.

It's true that in, in my experiments below, evolving obstacle detection and phototaxis as separate layers I preclude the faint possibility that phototaxis would in

some way evolve to take advantage of the obstacle detection mechanism. It is also conceivable, if improbable, that the phototaxis mechanism (or the obstacle detection ditto) would in some way evolve to take benefit of the signal that is presented to the associative learning network whenever a light source is reached. But these behaviours apparently work just fine anyway. And in evolving the obstacle avoidance mechanism on top of the one for phototaxis, the former is allowed to take advantage of the outcome of the latter (albeit indirectly, through the angle in which obstacles are approached) in determining its own outcome.

More importantly, if we never manage to evolve a certain complex behaviour at all, we obviously won't discover any ingenious evolutionary solution to the production of that behaviour. And that evolution can come up with novel and interesting solutions within one layer is evident from the top-layer learning network evolved in my first experiment below. I was actually surprised that a feedforward plastic network could perform learning at all - probably I was stuck in the Hebbian prejudice that some sort of direct connection between the reinforcement and light-source-reaching indicators would have to be evolved. I'm glad to have been proven wrong.

Artificial and natural evolution

Secondly, and related to the first point, a behaviour evolved using standard evolutionary robotics techniques might be taken as a proof that a similar behaviour could have evolved in this fashion in nature. The dynamics of the evolution of this behaviour might be used as argument that natural evolution of this behaviour shows similar dynamics - see for example research on punctuated equilibrium and biases in learning. Here it might be objected that layered evolution could not be used to study natural evolution in the same way. I could, in principle, evolve separate layers and combine them into a mechanism that would very improbably emerge in nature. There is some sense in this, but I think we can ensure that layered evolution can make good models of natural evolution by adhering to two principles: 1. Keep the controller purely neural, i.e. networks are only connected to sensors, actuators and directly to each other, and no other logic is prespecified (such as having the input network b consisting of whether an odd or even number of outputs of network a are activated). 2. Evolve the layers in an evolutionarily plausible sequence. For example, I think it plausible that an organism first benefit from evolving photo (or

chemo-) taxis and then, when it populates a more difficult environment, some form of obstacle avoidance. If each addition of a new layer can be justified, the whole architecture can.

Proximal and distal perspectives

Thirdly, and still relatedly, Nolfi (1997) made a distinction between *proximal* and *distal* perspectives in designing robot controllers, a distinction which is also referred to in further work in emergent modularity (Calabretta et al. 2000). The proximal perspective is that of the robots own sensorimotor system, and the distal perspective is that of a human designer. Nolfi further argues that in designing modular controllers, the best we can do from a distal perspective is to facilitate for evolution to create modularised solutions. The rest, i.e. deciding how the task should be divided, which module should do what and how the modules should communicate, should be left to the developmental process involving the agent and its environment to decide. Using these principles, layered evolution could be criticised on the grounds that functional differentiation is done from a distal perspective, and thus that valuable information that is only available from a proximal perspective is discarded.

The problem with this critique is that the proximal perspective only has an informational advantage for functional differentiation when all modules are evolved together. If, as in layered evolution, modules are evolved in succession, there is no way of knowing from the proximal perspective how many and which functions are to be present in the final controller. Thus differentiation from a proximal perspective suffers from an informational *disadvantage* and is likely to produce suboptimal differentiation schemes, as it tries to distribute a behaviour among a number of modules without regards to what behaviours will be added in the future, something that is only known from a distal perspective.

One can actually hypothesize a general conflict between temporal and topological structure in the evolution of a controller, and layered evolution as a way of resolving this conflict[7].

[7] Ziemke (2003) makes a related distinction between temporal and topoligical structure within an evolved network.

Symbol grounding

Fourthly, there is the problem of "symbol grounding". In the connectionist tradition, it was and is common to model cognitive functions with neural networks in a very abstract fashion, with no regards to whether the manner in which data was presented to and extracted from the network was biologically plausible. For example, a network that was meant to model language acquisition could be fed verb classes and output past tenses according to some complex formalization. It might seem, and so it definitely does to this school of cognitive scientists, that one is only postponing some work on "input/output mechanisms", work which can certainly be done later. But in fact, we don't know how any plausible I/O mechanism could input or output actual language to or from such representations as used in the connectionist models. So we might be overlooking the really hard problem. In fact, we might actually overlook the problem we are trying to address itself, as we don't know even *whether* any plausible I/O mechanism could do it. Very possibly work such as in the example might just be solving a pseudo-problem, and in biological reality the task is accomplished in some very different way, the critique goes. Something similar might be held against, for example, the associative learning network I will develop in my first experiment - taken for itself, its inputs and output seem highly abstract and idealized. Am I then not solving a pseudo-problem? Cliff's, as well as Brooks', solution to this problem was embodiment and situatedness. By connecting inputs and outputs ultimately to the physical world, or a simulation of it, they acquire meaning - our symbolic interpretation of them becomes quite literally grounded. I claim that this is the case for every evolved network in a layered structure as well. It is the organism/robot as a whole that needs to be in contact with the world, not every network. The output of the associative learning network is meaningful only because, connected to the conditional phototaxis network, it guides goal-seeking behaviour.

Generality

Finally, not all behaviours might be suited to layered implementations. But finding out which behaviours are and are not is an interesting question in itself, and LE is probably our best shot at answering this question. If the evolution of a behaviour is not significantly simplified by LE, then it can be concluded that the behaviour is not

by itself suited to layered implementations, which might in turn give us a clue about where to look for the neural substrate of this behaviour in nature.

First experiment: Layered evolution

To test my main hypothesis that layered evolution faster and more reliably produces desired behaviours than monolithic and incremental evolution, I compared these methodologies when applied to the problem of designing a neural controller for a simulated robot that would perform goal-seeking behaviour and lifetime learning in a cluttered environment.

This task is interesting because it requires some quite different behaviours, from the simple reflex-like obstacle avoidance, via the less predictable but still possibly reactive conditional phototaxis to the quite different learning behaviour. It is also interesting because learning behaviours are usually quite difficult to evolve (see e.g. Tuci et al. 2003, Yamauchi & Beer 1994) and as far as I know it has not been done in a cluttered environment.

Methods

Environment

The simulated robot "lives" in a rectangular area, with the dimensions 200 * 200 length units. The area can be empty, or populated with any number of light sources and obstacles. Light sources have no extension, and each broadcasts light in its unique colour. Obstacles are rectangular with side dimensions between 10 and 20 le. At the start of each trial a prespecified number of light sources and obstacles are placed randomly. The randomization of this and other elements of the simulation is done in order to avoid that the controller evolves to exploit contingencies in the agent-environment setup, e.g. a fixed distance between starting position and any of the light sources; evolved behaviour should be as robust as possible.

Agent

The robot is circular with a diameter of 10 length units, and is propelled by two parallel wheels that can move forwards and backwards independently, thus giving the robot the ability to move straight and turn any angle, including turning on the spot. The robot is not an accurate simulation of any existing physical robot, though it is loosely inspired by the Khepera micro robot.

Movement

The robot's maximum speed is 5 length units per time unit. When the robot reaches the border of the area, it ceases moving in the direction perpendicular to the border, but the component of its movement that is aligned with the border is unaffected, e.g. if it runs diagonally into a vertical border it continues moving vertically. If the robot runs into an obstacle, on the other hand, it simply stops. That is, any movement-step that would have ended inside an obstacle is not carried out. The reason for this uncompromising nature of the obstacles is my desire to make cluttered environments hard to navigate, thus giving the obstacle avoidance behaviour more of a challenge. I would also argue that in many natural settings, it is desirable not to run into obstacles – we wouldn't want a cleaning robot that continuously rammed into the furniture, even if that was the best way to navigate the room, and a fish that ran into rocks all the time would be prematurely battered.

Sensors

The robot is equipped with two types of sensors: light sensors and infrared sensors. A robot may have any number of sensors positioned in any locations on its body. Light sensors are tuned to specific colours, e.g. a red light sensor can only detect light from the red source. The sensor's output value is negatively correlated to the angle between the sensor and the source and also to the distance between robot and source. There is a further a cut-off function so that a sensor outputs nothing at all if the angle between sensor and source is more than 0.5 radians (about 28 degrees). Infrared sensors detect obstacles. A sensor outputs a positive value if there is an obstacle in its direction and within 15 length units of the robot; outputs are higher values the closer the obstacle is.

In all parts of this experiment the robot used one red and one green light sensor, both pointing straight forward (deviation 0 from the robot's direction of motion). In some parts of the experiment the robot was also equipped with three infrared sensors, fitted on the front of the robot with deviation –1, 0 and 1 radian (about 56 degrees) from the robot's direction of motion.

Subsumption architecture

The robot is controlled by one or more layers, where each layer consists of a neural network. Communication between layers is restricted to that higher layers (those

added later) can influence lower (earlier) layers using a hard-coded, task-specific link. A sensor can form input to any layer, but motors are controlled either by the lowest layer or by a subsumption mechanism, which is a small piece of external logic, that simply directs the output of a higher layer to the wheels if that layer wants to, and otherwise lets a lower layer control the wheels.

Neural networks

Each layer is a neural network. In this experiment, these are restricted to be of three types: feedforward perceptron-like networks, feedforward plastic networks and a hybrid feedforward network where each synapse can be either plain or plastic. (The software supports various forms of recurrent networks, but these have not been used in this experiment.) Plain (perceptron-like) networks consist of neurons arranged in a feedforward manner similar to networks used in backpropagation learning. Each neuron has a tanh activation function, and synapses (one-way connections between neurons) have genetically determined strengths in the range [-2..2].

Plastic networks are networks where connection strengths are randomly initialised, but can change under the influence of neural activity according to genetically specified rules. These rules are taken directly from the work of Floreano & Mondada (1996), who introduced synaptic plasticity to evolutionary robotics. Each synapse has one four plasticity rules and an update factor, which is in the range [-2..2] and determines how quickly the weight is changed. The rules are:

- Plain Hebbian: strength change = (1 – strength) * presynaptic activation * postsynaptic activation.
- Postsynaptic: strength change = (Plain Hebbian strength change) + strength * (-1 + postsynaptic activation) * presynaptic activation
- Presynaptic: strength change = (Plain Hebbian strength change) + strength * (-1 + presynaptic activation) * postsynaptic activation
- Covariance: d is defined as tanh (4 * (1 – abs (presynaptic activation – postsynaptic activation)) – 2). If d is > 0 then the strength change = (1 – strength) * d, otherwise strength change = strength * d.

In non-evolutionary learning experiments with feedforward networks it is common to add an extra input with a constant positive value to the network to allow it to have a positive output value even when none of its “real” inputs are positive, and so I have done in my experiments.

Genetic encoding

All synaptic parameters are encoded directly on the genome. A genome for a plastic network consists of a double-precision floating point array, where each number represents a synaptic strength. A genome for a plastic network consists of an array of integer values representing update rules for each synapse, and an array of floating point update factors.

Evolutionary algorithm

Networks were trained using an evolutionary algorithm resembling Inman Harvey's Species Adaptation Genetic Algorithm, SAGA. The population consisted of 30 individuals. In each generation, all genomes were evaluated and sorted on fitness. The lower (worse) half of the population was then removed, and replaced with copies of the upper half of the population. After this, all genomes except the five uppermost (best) were mutated, where mutation consists in every single gene of the genome having a 5 % chance of being changed to a randomly chosen value in the permitted range for the value. So a twenty-synapse network not among the best on average has one value changed in every generation. If nothing else is specified, evolution was run for 100 generations, each including 30 genome evaluations, which comprised five trials each and the lowest fitness score of these five trials used as fitness for the genome. This was done to decrease noisiness of the fitness landscape. In each trial the network was updated and the robot moved 200 times for the phototaxis and obstacle avoidance task, or 400 times for the learning tasks. All evolutionary runs took more than a minute and less than an hour on a 366 MHz Pentium II, the exact time depending mainly on the number, size and type of networks involved.[8]

[8] Given the abundance of evolutionary algorithm "recipes" and the variety of approaches used by ER researchers, the reader might wonder whether I have any justification for using this particular algorithm. In fact, I believe that a genetic algorithm with a population of one and neutrality, a so called Netcrawler, is generally the best algorithm for noise-free fitness landscapes – this has been proved to be the case for a certain class of landscapes by Lionel Barnett (2001). As the landscapes resulting from the problems in this experiment are very noisy – obstacle and light source positions are randomised for every trial, giving identical networks potentially wildly varying fitness scores – I believe a population-based approach is called for. Otherwise a single high fitness score for a bad genome might hinder any good genomes from developing. On the other hand, private investigations have led me to disbelief in crossover as a driving force in evolution, and so I don't see any point in large populations.

Tasks

The robot's ultimate task of learning which light source to move towards in a cluttered environment can easily be broken up into subtasks. Indeed, this was part of the reason I chose it. The first subtask is conditional phototaxis, which means that the robot should be able to reliably move towards one of two light sources and stay there or approximately there. Which light source is to be decided by an external input. The second subtask is obstacle avoidance, by which is meant that the phototaxis subtask should be completed even when there are ten randomly placed obstacles in the way between the robot and the selected light source. Remember that the robot stops dead in front of any obstacle. The fitness measure used in these tasks is 200 – (mean distance to the correct light source).

The third subtask is learning. Here, each genome evaluation comprises 10 trials, and the network is reset between each trial. In half of the trials, the goal is the red light source, and in half of them it is the green. The robot does not know which light source its goal is at the start of the trial, but can only learn from trial and error. To this end it is provided with three signals: one that is positive when the robot runs into the red light source, and zero otherwise, an identical signal for the green light, and a reward signal that is positive only when the robot runs into the correct source, negative if it runs into the wrong source and neutral otherwise. Each time the robot runs into a light source, be it the right or the wrong one, both light sources are assigned random positions. Which light source is correct and which is not does not change for the duration of the trial. In tasks involving the learning subtask, fitness is calculated as the number of light sources of the right kind reached minus the number of light sources of the wrong kind reached.

In the first experiment, I investigated some ways of evolving controllers to do this task.

Results

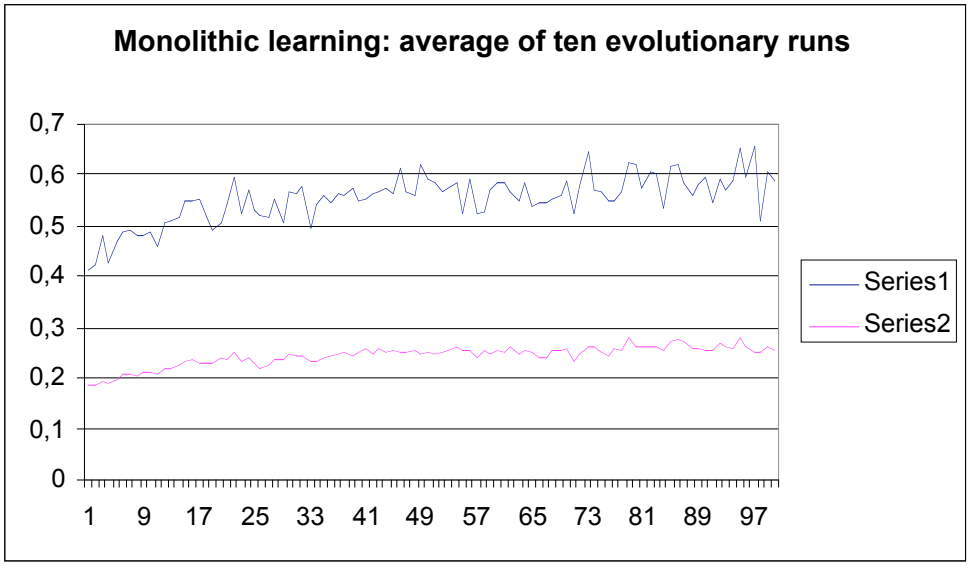
Monolithic evolution

To test whether it was possible to evolve a solution to the entire task using standard ER techniques, I constructed a monolithic neural network which I thought would be able to produce the desired behaviour were the correct synaptic strengths and synaptic learning rules to be evolved. The network had nine inputs: two light sensors and three infrared sensors as described above, two inputs signalling when a

red or green light source had been reached, one reward signal and one stable input. The two output neurons were connected to the wheel motors, and between input and output was a neural layer[9] of six interneurons. As learning requires plasticity and I wanted to retain the possibility for the reactive behaviours to use stable synapse weights, I implemented a hybrid neural network where each synapse could be either of the standard, hard-coded type or plastic.

The network was run for 100 generations 10 times, but no signs of intelligent behaviour were found. Average fitness of the population was around zero. Typical evolved robots either circled around constantly in wide or narrow circles, or just stood still. No analysis of the evolved networks was done.

Given the known difficulty of evolving learning behaviours in this way, I then followed Tuci et al. (2003) in adding a little tweak in order to give evolution any chance of succeeding: the probability of the goal being the red light source was twice that of the probability of the goal being the green source. This way, the fitness landscape would be less flack, as a controller would get some reward for seeking out red light sources, though a high score would require learning to evolve.



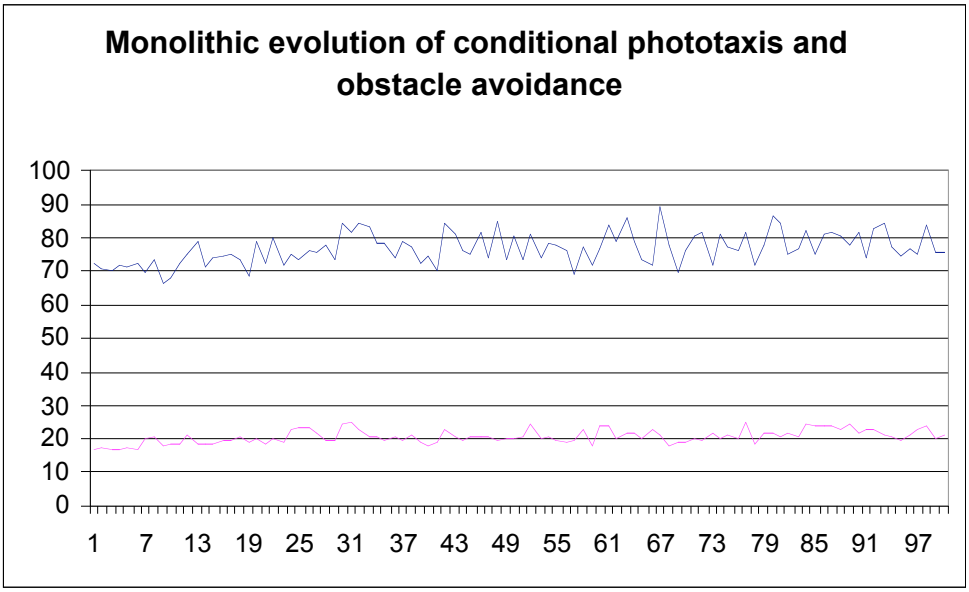
In the graph above, as in all fitness graphs in this dissertation, fitness runs along the vertical axis and evolutionary time along the horizontal axis. The upper line shows the

[9] What we have here is nothing less than a terminological disaster. Both behaviour-based roboticists and connectionist researchers have long used the word layer for two quite different things. In a typical feedforward network, a neuron in a particular layer has inputs from all neurons in the layer before and outputs to all neurons in the layer after, but not to those in the same layer. To lessen the confusion I will try to not use the word "layer" in its connectionist meaning, and when I have to use it I will try to mark its usage clearly by e.g. prefixing "neural" to layer.

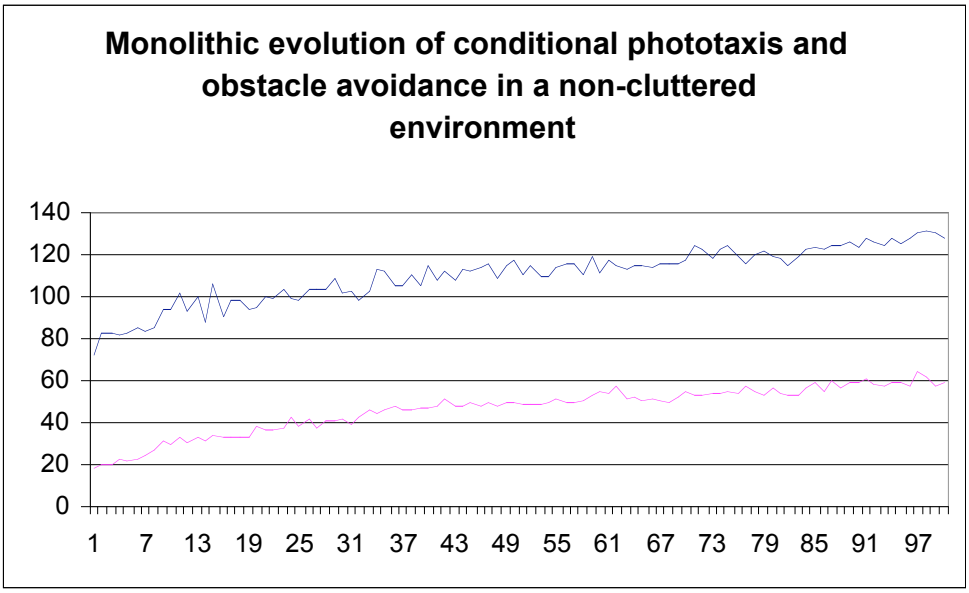
fitness of the best genome in the population in every generation, and the lower line shows the average fitness of the population.

As can be seen from this graph, this did not help much. A very modest increase in fitness results from the robot evolving some sort of phototaxis towards the red light, but neither was the phototaxis ever very efficient, nor did any signs of obstacle avoidance or learning show up.

Apparently the full task was too complicated to evolve in a monolithic fashion. Therefore I tried evolving only the two lower behaviours in this way, using a somewhat simpler network without the inputs that only related to the learning behaviour, but with a conditional input added for conditional phototaxis.



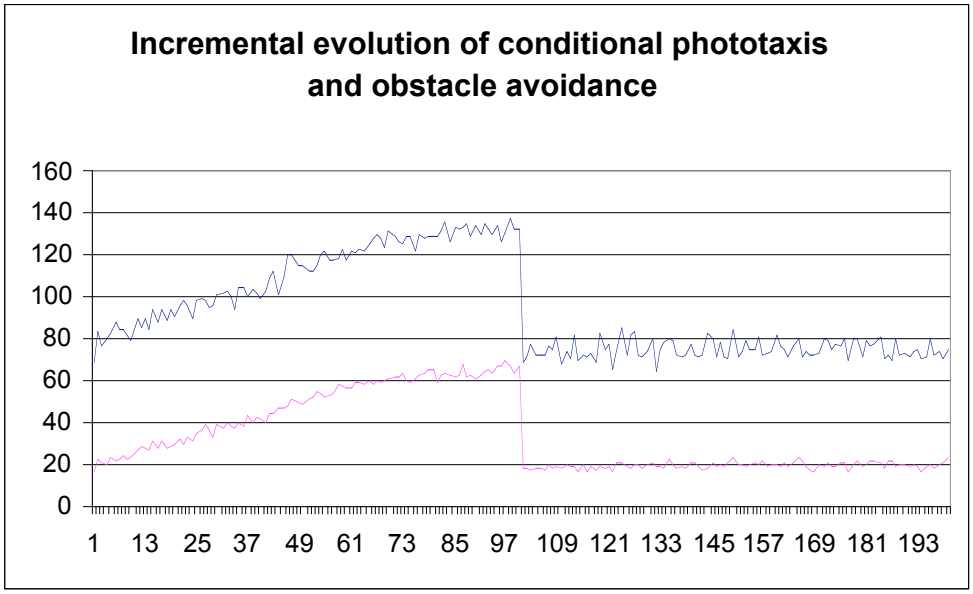
This did not go well. Performance was steadily at chance level. All evolved solutions I investigated moved about erratically. Therefore I tried evolving the same network as above in a non-cluttered environment, to see if at least conditional phototaxis would evolve.



Conditional phototaxis did evolve. After 100 generations a typical evolved controller moved toward the red light source if the conditional input was high and towards the green light source otherwise. The manifestation of that behaviour was as follows: by default, the robot rotated either left or right. If any light sensor signalled a high value, this put an extra effort to the inner wheel of the robot, that is, the wheel that rotated least in the default condition. This led the robot to move in an approximately straight line towards the light source. The conditionality was implemented in that a high value on the conditional input dampened the signal from the green light source to the inner wheel. (It is easily seen that this implementation of the behaviour required at least three neural layers in the network, as well as nonlinear activation functions.) In some evolved controllers, and under some conditions, the robot could enter a state where it circled both light sources; once it had rounded one light source it spotted the other and moved towards it, and so on. This imperfection in the behaviour could probably be eradicated with more sensors, but the behaviour produced was good enough for grounding learning, as it nearly always approached the light source specified by the conditional input first.

Incremental evolution

Next, the same monolithic network was evolved incrementally; for the first 100 generations no obstacles were present, and for the next 100 generations ten obstacles were present in the area.



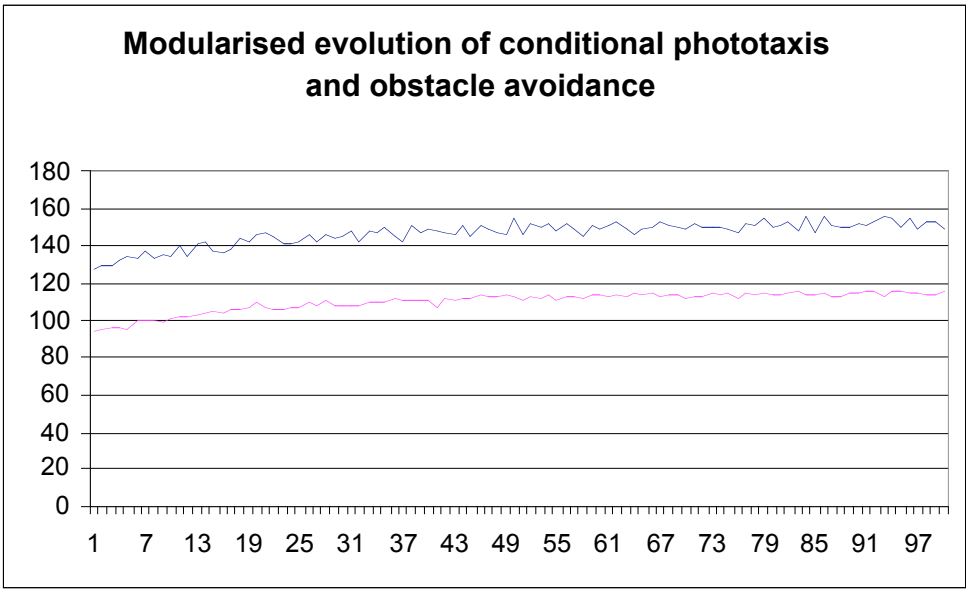
As can be seen from the graph, performance increases steadily for the first 100 generations and then drops dramatically to the same level as for generation 0, which is the level exhibited by a randomly wired controller. That the network is able to evolve conditional phototaxis but not obstacle avoidance is confirmed by behavioural observation; the best controller from generation 99 performs good in a non-cluttered environment, while the best controller from generation 199 moves erratically, though with some goal-direction, and does not have a good strategy for avoiding obstacles; when encountering an obstacle it often oscillates between turning left and right, without moving past the obstacle.

Here we see that not only did satisfactory obstacle avoidance not evolve, but also that the already evolved mechanism for conditional phototaxis was damaged. This may have been due to that the fitness landscape lost its gradient when one phototaxis strategy was as inefficient as another in the presence of obstacles, that the existing neural pathways for phototaxis collided with those that evolution tried to use for obstacle avoidance, or a combination of both.

As obstacle avoidance did not evolve, it was not meaningful to try to add learning on top. It is also hard to come up with a good scheme for incrementally evolving learning.

Modularised evolution

I then tried evolving obstacle avoidance and conditional phototaxis as two separate layers, but at the same time.

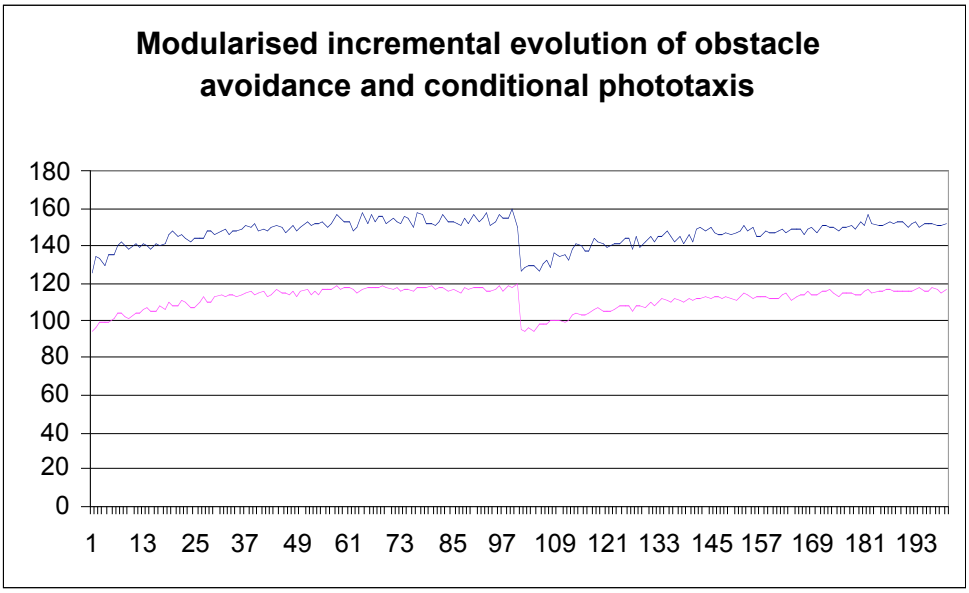


The fitness increase is significant, but not great. Behavioural analysis shows that the robot performs good conditional phototaxis and decent obstacle avoidance.
I also tried evolving learning in this modularised fashion, but those results were no good at all; mean fitness hovered around zero and behaviour was erratic[10].

Incremental modularised evolution

I then tried combining modularised and incremental evolution; this approach differs from layered evolution only in that the first layer is continually evolved even as the second layer is added, and that the second layer is present while the first is evolved. For 100 generations, the robot performs conditional phototaxis in an uncluttered environment, and then ten obstacles are added.

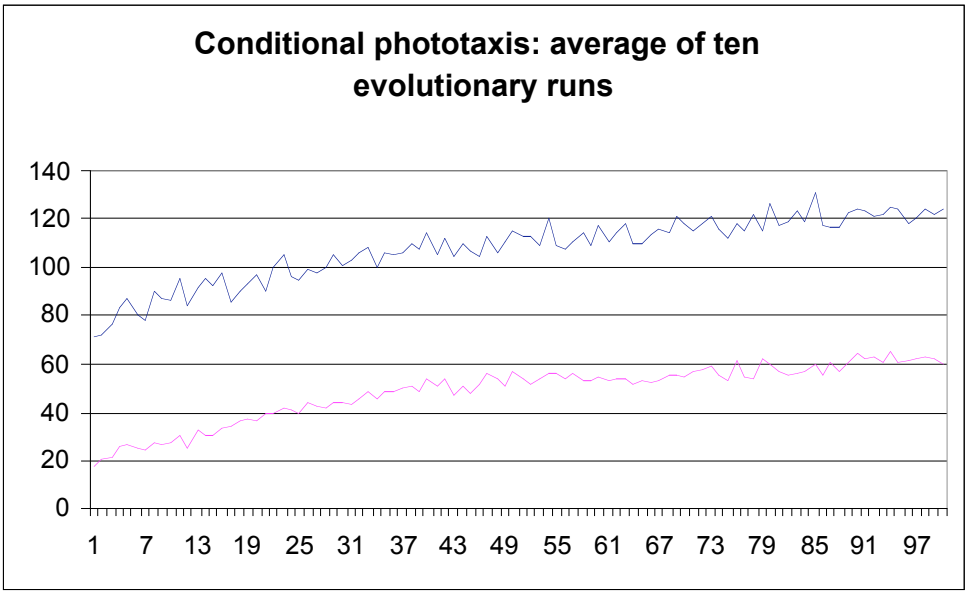
[10] The graph got lost and lack of time hindered me from creating a new, but it wasn't much to see anyway.



The robot performs good phototaxis both at generation 99 and 199, and the obstacle avoidance is as above (decent), but the latter takes some time to evolve.

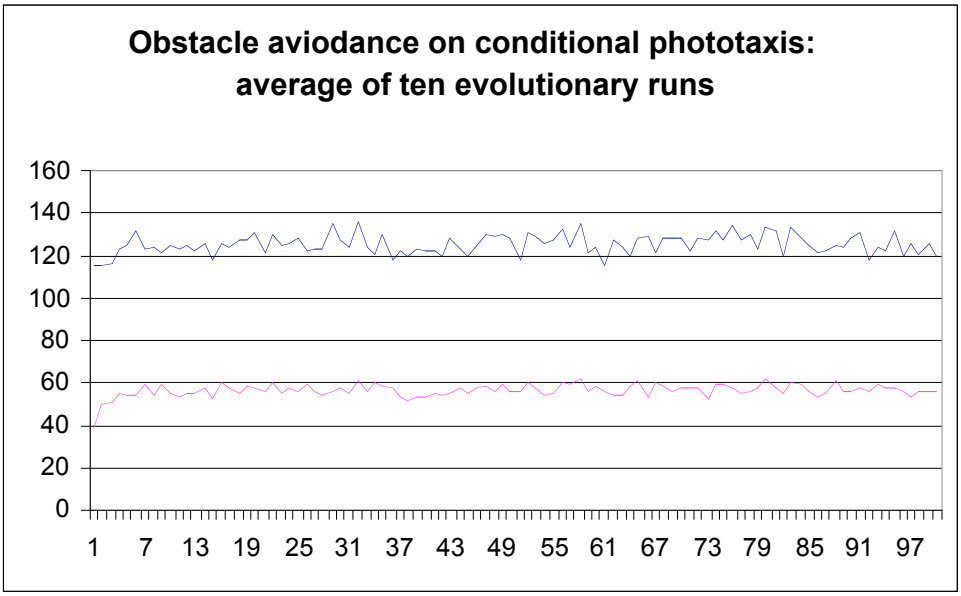
Layered evolution

Finally, I tried evolving the behaviours incrementally and in separate layers. First the conditional phototaxis layer was evolved. This consisted in a plain network with four input neurons, three interneurons and two output neurons.



Rather quickly, good conditional phototaxis is evolved. The manifestation of the behaviour is essentially as described for a monolithic network in an uncluttered environment described above.

Then, the obstacle avoidance network – four input neurons receiving three infrared and one stable input as described in methods, and three output neurons outputting motor signals and a subsumption signal – was evolved.

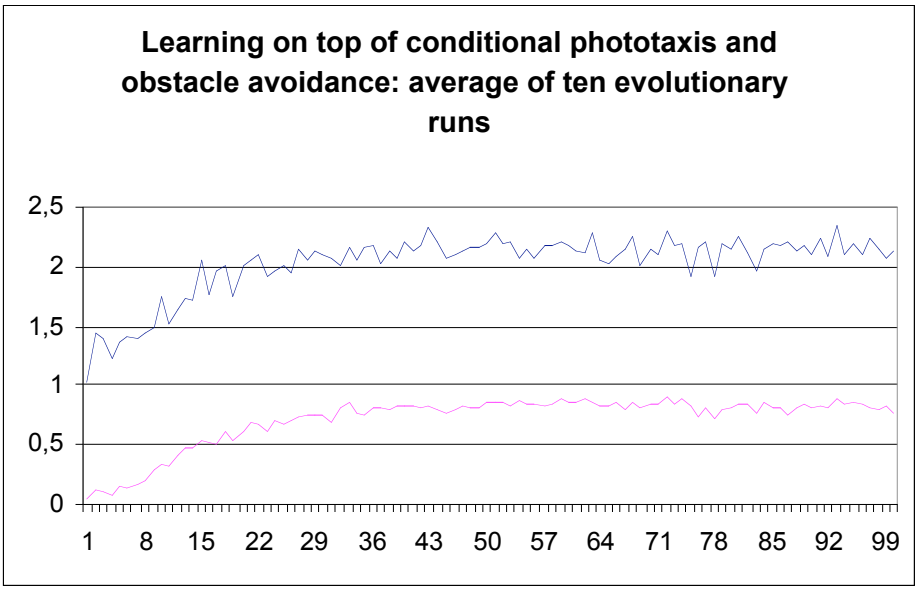


The graph may look surprising, but the apparent lack of gradient is only because the obstacle avoidance gets as good as it gets within the first five or so generations. It does improve fitness: conditional phototaxis on itself in a cluttered environment gets much lower fitness scores. This suggests that obstacle avoidance, when evolved for itself, is very simple to evolve – a suggestion reinforced by the tiny size of the obstacle avoidance network. What is striking here is that the same behaviour evolved so slowly in incremental modularised evolution, probably because the conditional phototaxis layer was evolved at the same time and many "good" mutations in the obstacle avoidance layer was wasted because of "bad" mutations in the conditional phototaxis layer.

It is also worth mentioning that the obstacle avoidance never gets really good. If the robot runs straight into an obstacle it ususally spends some time jerking back and forth before finding a way to get by. Ideally it would smoothly avoid the obstacle in a way that respects the initial path of the robot and minimises the disturbance caused by the obstacle. As we can see from the obstacle avoidance experiments herein, we are not to expect better performance on this task with the present combination of network, sensors and architecture. Whether the key to improvement lies in the network (we may another neural layer, or neurons or synapses with state), the sensors (maybe they need longer range) or the architecture (we may need

"upwards" connections from the conditional phototaxis layer) is a question for future experiments.

The uppermost layer was the learning layer. This was a plastic network with four input neurons, two interneurons and one output neuron. The input neurons were connected to sensors for having reached any of the light sources, a reward signal and a stable signal as described in methods.



Here we see a dramatic fitness increase in only a few generations. After only a few generations at least one controller with learning ability was present, after fifteen generations very good learning networks had been evolved and after thirty generations fitness did not increase anymore. This suggests that learning taken for itself is not a very hard behaviour to evolve. An analysis of an evolved network confirms this suggestion.

Initially I was rather surprised that learning would evolve at all in a feedforward network. My idea of synaptic plasticity was that there ought to have been a direct connection between for example the sensor for reaching the red light source and the reward input, associating the red light sensor with a reward, an arrangement that is impossible in a feedforward network. Evolution found other solutions. In one controller the following was the case: in the default mode, there was a covariance negative connection between the reward input and one of the interneurons, another between that interneuron and the output neuron, and a plain Hebbian connection between the stable input and the same interneuron. So the controller defaulted to outputting zero, which meant go for the green lights. If the reward input ever

became negative, the following process was triggered: The covariance connection between the input and the interneuron was strengthened, and the activation of the interneuron strengthened the Hebbian connection between the stable input and the interneuron. The sustained activity quickly strengthened the connection between the interneuron and the output neuron, which pushed the network into the stable state of a positive output, meaning that the robot should go for red lights.

It is interesting that the evolved solution did not use the sensors for having reached red or green neurons at all; apparently the network for the learning layers could have been even smaller and thus evolved even quicker. On the other hand, this solution means that the network would not be able to learn a new configuration if the preferred light source changed in the midst of a trial. Evolving networks for such a task would be an interesting next step; in particular, it would be interesting to see whether a feedforward network could do it.

Last but not least, it should be pointed out that the time taken to evaluate a genome, and therefore also the time taken for a complete evolutionary run, is much shorter for the layers in layered evolution than for any other parts of the experiment.

Second experiment: Merging layers

While it certainly seems that layered evolution quickly and reliably evolves desired behaviours, it cannot be excluded solely on the grounds of the first experiment that there are some advantages of monolithic or incremental evolution that we have overlooked. I.e., is the maximum fitness for a layered architecture lower than that of a monolithic architecture on this task? To find out, I decided to merge together the layers of a ready-evolved layered architecture, and see if this would increase fitness beyond that of the pure layered structure.

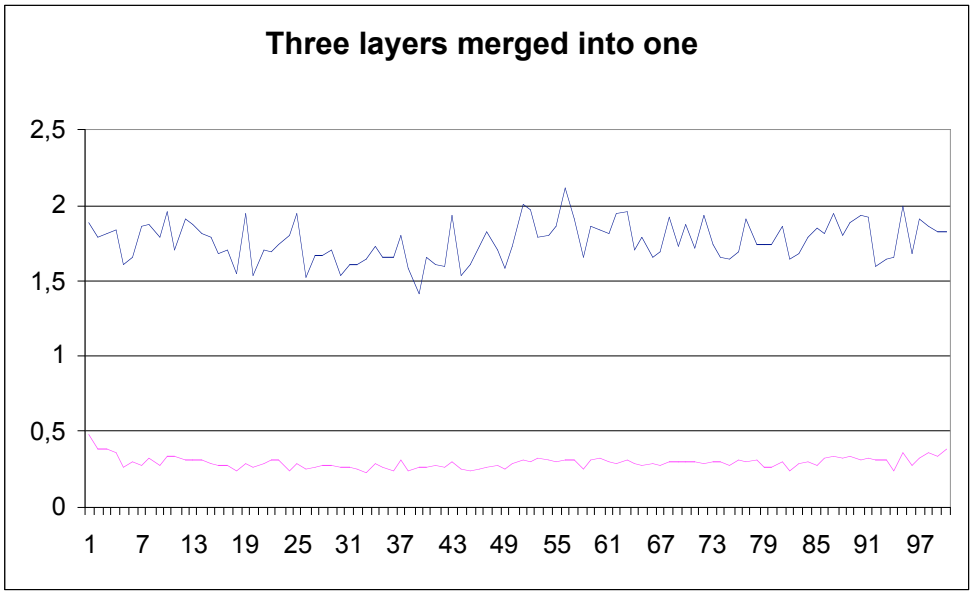
Methods

To the rigid layered architecture used in the first experiment, a mechanism was added for connecting layers with arbitrary but genetically specified connections. Between every two adjacent layers, a connection layer, which was essentially a variable length list of connections, was introduced. At the start of an evolutionary run, all connection layers had zero connections. Every time a genome was mutated (which happened to 25 out of 30 genomes every generation) every connection in the layer had probability = mutation rate of being deleted. Also, there was five times the mutation rate probability that a new connection would be introduced, from a randomly selected neuron in the layer above the connection layer to a randomly selected neuron in the layer below; strength was a random number in the range [-2..2]. When the genomes fitness were evaluated, neural activations were propagated not only within the network but also potentially from higher to lower layers via the layer connections, which in every other respects worked like normal synapses.
It is easy to see that in the absence of selection pressure, each connection list will contain on average five connections, but also that in theory any number is possible. The nature of the mutation mechanism, however, makes it hard to retain very large number of connections between any two layers, but I don't believe this to be a factor in the present task. The conditional phototaxis layer, for example, totals 18 synapses and so it would anyway be hard to meaningfully use more than, say, 10 connections from the obstacle avoidance layer[11].

[11] There is much research going on into novel encodings of neural networks, e.g. Gruau encoding. I have yet not seen any experiments where the whole network is encoded using variable length lists and neural indexes like my interlayer connections are encoded. Such an encoding would not have the desirable property of sublinear length encoding, but it would put very few constraints and

Results

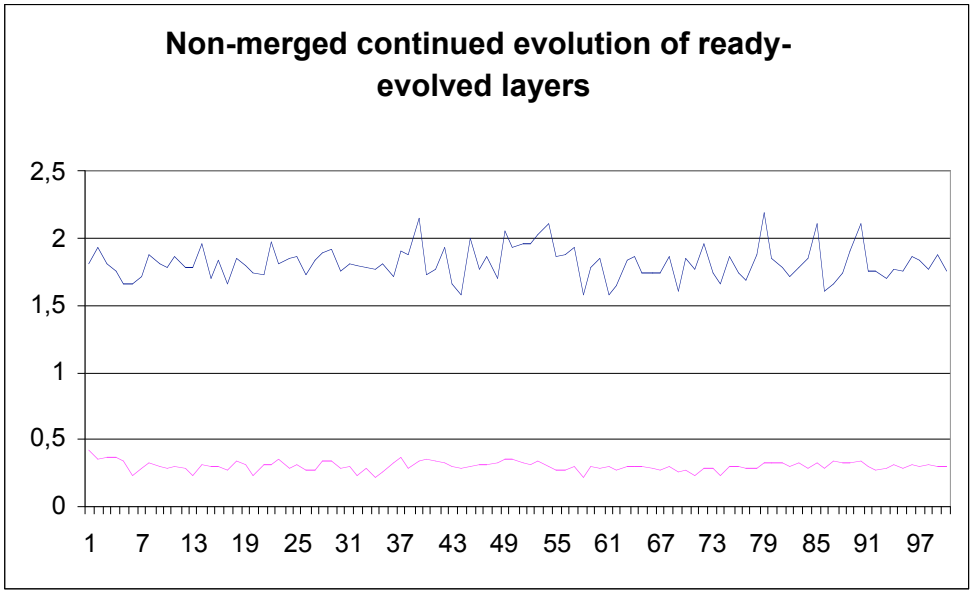
One genome for each of the three layers of the layered evolution part of the first experiment was taken from the final generations of that experiment. A new evolutionary run was seeded with new modular genomes based on these layer genomes. At the outset, no interlayer connections were present, but these were allowed to be created and mutated in the process described above.



No fitness increase was seen, and no behavioural differences were noted. The initial drop (generations 0-5) in average fitness can be explained with that the homogeneity of the initial population, where every genome incorporated copies of the best layer genomes, is broken up; most mutations in any evolutionary algorithm decreases fitness. The mean number of connections in the connection layers was around three, which indicates some selection pressure against interlayer connections other than the hard-coded.

For comparison, new evolutionary runs with the same starting conditions as in the above experiments but without any merging taking place (i.e. the connection layers were restricted to length zero) were performed.

biases on network topology. This in contrast to Gruau encoding, which is heavy with topological constraints and biases, and is sublinear only in special cases.



No notable difference between this condition and the merging condition can be seen.

These results may be taken as an argument that the layered controller architecture was indeed optimal; if it wasn't, the merging together of the layers, which makes it possibly to use communication channels between the layers that couldn't have been used in the strictly layered solution, would probably have evolved solutions with higher fitness. Of course, there is always a chance that the optimal architecture is so radically different from the layered architecture of my experiments that the former could not have been evolved starting with the latter, however merged. But as I see it, our best theory is that the layered architecture is indeed optimal for the task, and the result from the merging experiment is corroborating this theory.

Discussion

What has been achieved

In this dissertation, I have developed an approach to robot controller design that mixes evolutionary robotics with certain features from behaviour-based robotics. I have argued the distinctive advantages of this approach for science and engineering, and defended it against some conceivable criticisms. Further, I have developed a simulation environment in my approach can be implemented, and predictions from my theory can be tested. The most important prediction was that both incremental and modularised evolution would be better at evolving solutions to tasks than monolithic evolution, but that layered evolution would be superior to all three other methods. This was indeed shown to be the case for the task at hand, learning which light sources to approach in a cluttered environment. Some behavioural and structural analyses were done of evolved controllers, and the evolved strategies and structures conformed pretty well to expectations, except for the learning network. Finally, I have developed a method for merging together the layers of a layered architecture, and on the grounds from the results of a merging experiment argued that the layered solution is indeed optimal for this task.

What has not been achieved

All experiments in this dissertation have been done with one task (and its constituent subtasks), one robot architecture and one type of environment – and both robot architecture, network types and environment were quite simple. Thus it is hard to draw any conclusions about the suitability of layered evolution for robot tasks in general; one may argue that my choice of tasks was abnormally suited to the approach. As for the scientific side of evolutionary robotics, I have argued principles in my theoretical section, but have proved little in practice. I doubt that the experiments conducted herein will contribute much to the solution of any important scientific problems, at least not directly.

Finally, I have not done any mathematically sophisticated analysis of my results, but I did not judge this necessary, as the most important feature in the results – whether a particular behaviour evolved or not – was quite clear-cut in every case, and other features, such as how the behaviour was manifested, is not easily

amenable to any meaningful mathematization. This could of course be because of my ignorance of appropriate methods and formalisms.

Directions for future research

While the experiments herein serve as some sort of test of my hypotheses and proof of my points, more could be done with the specific task at hand. Some might argue that my experiments were unfairly biased against monolithic and incremental evolution as the monolithic networks totalled a smaller number of neurons, or that more neural layers would have been needed for a monolithic network to perform the entire task. I don't believe this – after all, a sufficiently large three-layer perceptron has universal approximation capability – but experiments could be done where the size of monolithic networks was itself evolved. Another limitation of my experiments is that I only used feedforward networks. It would be interesting to try with e.g. CTRNNs, especially as the obstacle avoidance network only ever performed sufficiently, but never very well. Maybe more than reactive behaviour is needed for really good obstacle avoidance. It would also be interesting to see whether the top layer could support re-learning, i.e. changing goals during agent lifetime. Would this be possible with a feedforward plastic network? A final, but very interesting, extension of my original experiments would be to evolve the layers without any prespecified connections, but with evolvable layer connections, and see what sort of connections would be evolved. (This was actually suggested to me early in the course of the project, but has been left out in order to keep the dissertation focused.)

Moving on to a broader perspective, the suitability of layered evolution has to be tested for many more tasks. One approach would be to replicate other experiments done in evolutionary robotics and investigate whether the tasks therein benefit from layering, but it would it be more interesting to look at the behaviour-based robotics literature and try to replicate those experiments; this would represent real scaling up of ER. Of the most interest is of course tasks that behaviour-based robotics researchers have failed to solve – here evolution might see solutions that humans have been blind for. As BBR experiments are typically done on real, physical robots, such experiments would have to be done either directly on hardware – still quite seldom done in ER – or in much more complex simulations.

The ideas taken up in the section on layered evolution and science in the background chapter are worth taking up, in particular evolving behaviours in behaviour systems descriptions of actual organisms, and modelling symbiotic and parasitic organism relationships with relationships between layers in a controller. Such investigations would probably be best classified as computational neuroethology (Cliff 1991). A first step towards modelling conflicting relationships between layers could be to add a layer whose task it was to do classical pole balancing, a task that conflicts heavily with moving in straight lines and continous speeds. The genome could be scored according to the worst performance of either pole-balancing or the other tasks, meaning that the whole organism must succeed[12]. This would resemble the situation of a parasite, that struggle to reproduce itself while avoiding that the host dies.

Finally, I hope that experiments in layered evolution can help to shed light over some of the same questions as those investigated in emergent modularity, such as the role of modularisation in general.

[12] This idea came up in a discussion with Dyre Bjerknes.

References

Barnett, L. (2001). Netcrawling – Optimal Evolutionary Search on Fitness Landscapes with Neutral Networks.

Beer, R. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior* 3(4):471-511.

Bishop, C. (1995). *Neural networks for pattern recognition*. Clarendon Press.

Blynel, J. & Floreano, D. (2003). Exploring the T-Maze: Evolving Learning-like Robot Behaviors CTRNNs. In*Proceedings of EvoROB2003: 2nd European Workshop on Evolutionary robotics*.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2, 14-23.

Brooks, R. A. (1989). A robot that walks: emergent behaviour from a carefully evolved network. *Neural Computation*, 1, 253-363.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, 47, 139-159.

Brooks, R. A. (2002). *Robot: the future of flesh and machines*. London: Penguin.

Calabretta, R, Nolfi, S., Parisi, D. & Wagner, G. P. (2000). Duplication of modules facilitates functional specialization. *Artificial Life*, 6(1), 69-84.

Cliff, D. (1991). Computational neuroethology: a provisional manifesto. In J.-A. Meyer and S. W. Wilson (eds.) *From Animals to Animats: Proceedings of The First International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: MIT Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2001). *Introduction to algorithms, second edition*. Cambridge, MA: MIT Press.

Di Paolo, E. (2000). Homeostasis…

Floreano, D. and Mondada, F. (1996). Evolution of Plastic Neurocontrollers for Situated Agents. In P. Maes, M. Matarc, J.-A. Meyer, J. Pollack & S. Wilson (Eds.), *From Animals to Animats 4, Proceedings of the International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press.

Gomez, F. & Miikulainen, R. (1996). Incremental Evolution of Complex General Behavior. Technical Report AI96-248, Austin, TX: University of Texas.

Lipson, H. (2000). Uncontrolled Engineering: A review of Nolfi and Floreano's *Evolutionary Robotics*.

Murphy, R. R. (2000). *Introduction to AI Robotics*. Cambridge, MA: MIT Press.
Nolfi, S. (1997). Using emergent modularity to develop control systems for mobile robots. *Connection Science*, (10) 3-4: 167-183.
Nolfi, S. (2002). Evolving robots able to self-localize in the environment: the importance of viewing cognition as the result of processes occuring at different time scales. *Connection Science* (14) 3:231-244.
Prescott, T. J., Redgrave, P., & Gurney, K. (1999). Layered control architectures in robots and vertebrates, *Adaptive Behavior*, 7, 99-127.
Tuci, E., Quinn, M. & Harvey, I. (2003). An evolutionary ecological approach to the study of learning using a robot based model.
Urzelai, J. & Floreano, D. (1999). Incremental Evolution with Minimal Resources.
Yamauchi, B. & Beer, R. D. (1994).
Ziemke, T. (2003). On ’Parts’ and ’Wholes’ of Adaptive Behavior: Functional Modularity and Diachronic Structure in Recurrent Neural Robot Controllers.

Software notes

The full source code for the program developed for and used for the experiments in this dissertation is listed in the appendix, and is availabe in digital form upon request from julian@togelius.com, and hopefully also att http://julian.togelius.com. Here I will give a brief overview of the program.

The whole project was programmed in Java 1.4.2, though somewhat earlier and all later java versions will probably work with the code. No work has gone into any sort of user interaction, but there is some quite helpful visualization. Throughout my work I have struggled to keep the code as modular as possible, so that adding e.g. new varieties of tasks, networks or robots won't be too much of a problem. Generally (with the main exception of neurons and synapses, partly for performance reasons and partly for no particular reason) an object-oriented methodology has been used, meaning that each entity of some kind is modelled as an object containing its own data structures and methods for accessing and mutating them. As is ususal with java, the code is organized into packages, classes and interfaces, where some classes are subclasses of others.

The package *evolayers* is the main package, which also contains the startup class, EvoLayers. The class Task is the superclass of the numerous task classes, which each implements one part of one of my experiments. Network is superclass of PlainFFNetwork, PlasticFFNetwork and ComboFFNetwork, which together with the superclass implement neural networks. An object of the class of Evolver contains a method implementing artificial evolution. A Genome, or rather one of its subclasses, contains everything needed to build a Network and methods for e.g. mutation.

The package *simplebot* is the robot simulator. Here, the class SimpleBotModel does all the actual simulation work; the others are merely for visualisation.

The package *netvisualisation* provides real-time visualisation of the neural networks, so that changing neural activations and synaptic strengths can be monitored as the robot moves around.

All code is free for anyone to use, though I would be delighted to know of it if anyone did.