A 'Fitness Landscaping' Comparison of Evolved Robot Control Systems

Stevan Jay Anastasoff
Evolutionary and Emergent Behaviour Intelligence and Computation Group
School of Computer Science
University of Birmingham
Birmingham B15 2TT, United Kingdom

Abstract

Most work in evolutionary robotics has focussed on evolving neural network based control systems, rather than high level control programs using genetic programming. One reason for this is that the fitness landscapes generated by a genetic programming framework may be poorly suited to the evolutionary techniques employed. The aim of this paper is to investigate this claim. The first part of the paper demonstrates two simulations of evolving populations of simple robots, one based on genetic programming, the other based on evolved neural nets. The latter part of the paper then introduces a technique for 'fitness landscaping', that is for constructing 3-D visualisations of the sorts of complex, high dimensionality fitness landscapes involved. This technique is applied to the results from the two simulations, and the resulting representations are discussed.

Introduction

Traditional wisdom in the field of evolutionary robotics holds that neural network based control systems are a more suitable evolutionary architecture than genetic programming [5], [10]. Compared to neural nets, the primitives of a genetic programming framework are very high level. The result of this is potentially a "...coarse-grained fitness landscape with steeper precipices." ([5], p366). However, a significant amount of work in evolutionary robotics has still been carried out using genetic programming, with some good levels of success [7], [8], [12], [9].

In the simulations to be presented here these two evolutionary robotics paradigms will be compared, in order to evaluate the statement quoted above. The two approaches are used to develop control systems for basic navigational behaviour (wall following) using a 2-D representation of a simple autonomous robot. Samples from each generation of the two evolutionary processes are then stored in order to be used to create visualisations of the sorts of fitness landscapes generated. These can then be compared with the sorts of landscapes expected from the earlier claim.

The 'fitness landscaping' process involves projecting the genetic material random samples from the evolving populations on to a 2-D lattice according to the level of similarity between the contents of the cells of the lattice. A simulated annealing algorithm determines the distribution of the samples in the lattice. This optimises the distribution such that the most cells have the most in common with their neighbours. The fitness level of the contents of each cell is then mapped in order to generate the final 3-D visualisation. This whole process will be described in more detail in the later parts of the paper.

Two Evolutionary Simulations

The following sections describe in detail the simulation environment, together with the genetic programming and evolved neural net algorithms used.

The Robot Simulation

The simulation models a 2-D robot together with its environment, consisting of a close-ended corridor containing several bends (two right-hand bends, followed by two left-hand bends). This scenario is derived from [12]. The robot's behaviour is then governed by an evolved control program or neural network fed in by the appropriate evolutionary framework. The robot itself is a somewhat simplified version of that described in [5]. It consists simply of two motors, one either side, capable of rotating backwards and forwards independently, and two sensors. The sensors are 'antenna' or 'whisker' like proximity switches. These independently return either a value of true, if they are in contact with an object, or false otherwise. They are located at the front of the robot, protruding at approximately 30 degrees to either side. They are of approximately the same length as the body of the robot.

The model is based on an actual simple Lego robot. The parameters used have been selected to make the simulation correspond as closely as possible to this real robot. A small amount of noise, as has been demonstrated to be beneficial (in [12] for example), was also introduced. The noise levels were again intended to be consistent with comparative runs with the real robot. The entire simulation has been developed using the programming language POP-11, running the PopBugs simulation software.

Both the robot being simulated, and the environment in which it is situated, were both intentionally kept as simple as possible. The idea here was that by having a simpler simulation, simpler (and thus easier to model) fitness landscapes would be generated.

Evolving the Control Programs

A very much standard genetic programming framework was used to supply the control programs for the simulation. The genetic operators used, as well as the particular functions and terminals, were based closely on those found in [7]. An initial population of 50 control programs was randomly generated using the 'ramped half-and-half' method, to create a wide diversity of different sized and shaped starting tree structures. The population size was somewhat smaller than those generally used in evolving robot control programs, but this can be justified by the simplicity of the robot and problem being simulated. The fitness of each of these programs was then assessed by running the simulation and measuring the distance the robot managed to travel along the length of the corridor. The simulation was stopped and distance measurements were taken once the robot's wheels had rotated a fixed number of times. Simulations in which the robot's wheels stopped moving for two consecutive time steps were finished automatically at that point. Parents for each successive generation were chosen using a method of tournament selection. Two candidates were chosen, and the fitter was allocated as the parent with a probability of 0.9. For each pair of parents so chosen, crossover occurred along a random branch in the program tree with a probability of 0.8. Otherwise the two parents were reproduced without crossover. This was repeated enough times to fill the next generation. Fifty consecutive generations were run in total.

The function and terminal sets used were quite small, again as stated above with the intention of producing a simpler and more easily represented fitness landscape. Three functions were used: the two conditionals *ifRight* and *ifLeft* each take two arguments. If the appropriate side sensor is activated the first argument is evaluated, otherwise the second argument is evaluated. The third function used is the program connective *prog*. This again takes two arguments, and evaluates each in turn. The terminal set consists of four operators: *forwards*, *backwards*, *turnLeft* and *turnRight*. Each of these moves the robot in the stated manner; forwards and backwards each move the robot one full body length, turnLeft and turnRight turn the robot approximately 20 degrees in the appropriate direction (all values averaged over noise).

Evolving the Neural Networks

The neural net controllers and the evolutionary framework used to develop them were based in part on those used in [5]. Each controller consisted of a simple feed-forward neural net containing two input nodes (one connected to each sensor), two 'hidden' nodes, and four output nodes used to control the two motors. Each node was a noisy linear threshold device, allowing for a much broader range of possible interesting behaviours. This principally differs from those networks used in [5] through the reduced number of input nodes, and the fixed number of 'hidden' nodes.

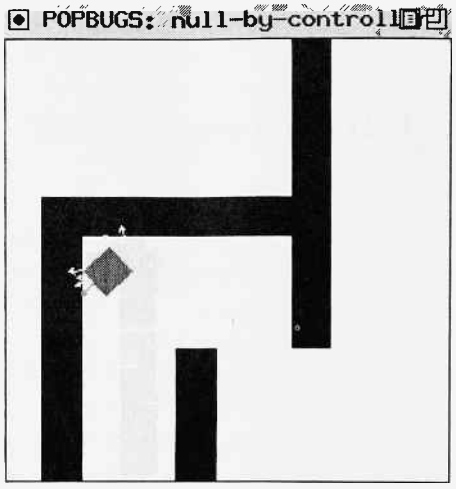

Figure 1: Evolved robot control program, population best from the first generation of the genetic programming framework.

Each network was encoded genetically as a fixed length chromosome consisting of 22 floating point numbers. Since the network topology was fixed, each weight and threshold value in any net could be assigned by the value at a specified location in a given chromosome. A standard 'vanilla' GA, using single point crossover and tournament selection was then used to evolve a population of network chromosomes. As in the genetic programming framework, a population size of 50 was used, again being evolved for a total of 50 generations.

The Resulting Control Systems

A variety of different behaviours evolved over the course of the simulations.

Screen shots for four of these can be seen in the accompanying figures 1 through 4. These same basic classes of behaviour, and the progression through them, were present for both evolved neural networks and control programs. Note that in these figures the sides of the boxes designate the edges of the environment and are impassable.

Those solutions that produced movement in a straight line achieved the most basic level of success, traveling as far as the first bend (figure 1). These programs dominated the first few generations. The next level of sophistication, the 'right-turners' (figure 2), rapidly usurped them however. These travel forward in a straight line until bumping into a wall, and then turn right. This be-

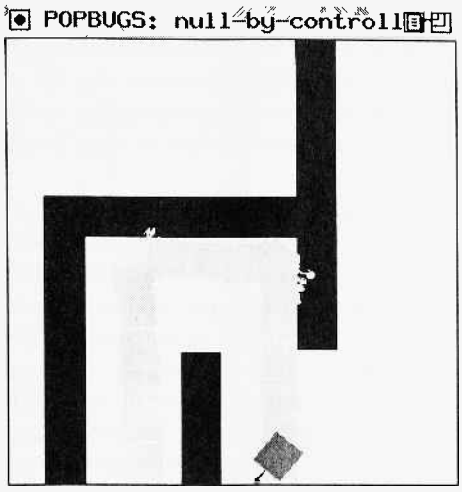

Figure 2: Evolved robot control program, population best from the third generation of the genetic programming framework.

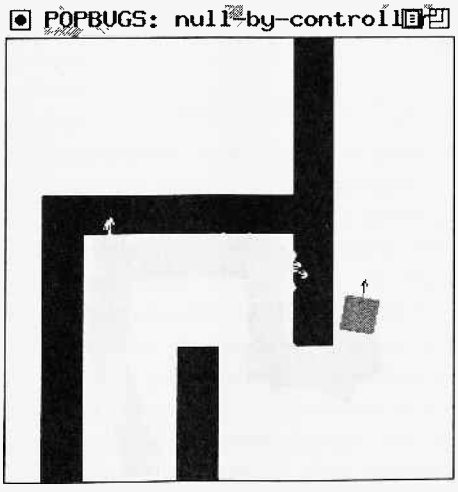

Figure 3: Evolved neural net, population best from the seventeenth generation of the evolved neural network framework.

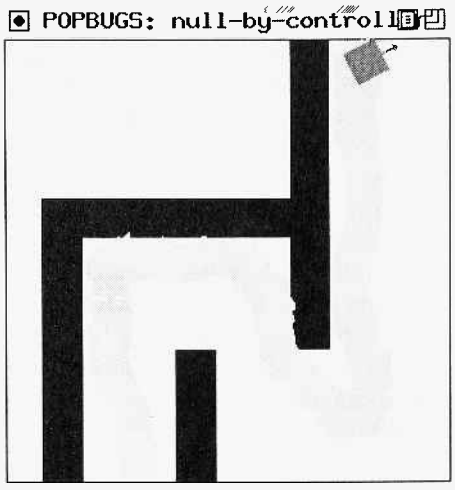

Figure 4: Evolved neural net, population best from the fiftieth (final) generation of the evolved neural network framework.

haviour manages to get them as far as the third bend. Within ten to twenty generations the first of the more sophisticated strategies could be seen to be emerging (figure 3). These cause the robot to veer over to the left until bumping into a wall, and then turn right. This behaviour can be effectively described as 'find the wall on your left hand side and follow it'. This strategy is sufficient to drive the robot the full length of the corridor. However, in the early versions, the optimal turning ratios were yet to evolve, with far too much time spent turning too sharply. Throughout the remaining generations this final behaviour gradually came to dominate both types of control system (although only very late on in the run did it eventually completely supercede the right-turning behaviour), slowly becoming more refined. By the final generation something like the solution demonstrated in fig. 4 had evolved in both evolutionary systems. The turning ratios have been optimised, and the robot is capable of traversing the full length of the corridor.

It is perhaps worth noting that despite the extremely limited number of test cases (one!), perfectly generalised corridor following behaviours have evolved. Although the early strategies exploited features peculiar to the single test case (e.g. right-turning behaviours), the final best scorers are perfectly capable of following any corridor (employing the simple strategy of finding a wall and then sticking to it).

Quantitative Comparison of the two Algorithms

Figure 5 shows a comparison of the performance over time of the evolved neural networks and control programs. Up to the tenth generation, the genetic programming framework can be seen to perform slightly better than the evolved neural networks. However, from the

tenth generation onwards, the neural networks have a slight, though clear and consistent, advantage over the evolved control programs. This suggests that the genetic programming algorithm may be slightly better at finding the simpler strategies, such as moving straight ahead or right-turning, while the neural network algorithm may have a slight advantage in moving from these simpler strategies on to the more sophisticated strategies needed to traverse the full length of the corridor.

This conclusion would fit in well with the earlier claim concerning the structure of the fitness landscapes generated by the two approaches. This will be discussed further later, in conjunction with the relevant fitness landscape visualisations.

The 'Fitness Landscaping' Algorithm

Having looked at the simulation environments and the two evolutionary frameworks, the next sections will present the 'fitness landscaping' algorithm employed. The idea behind this landscaping algorithm is to generate a 3-D visualisation of the fitness landscapes produced by particular evolutionary approaches to a given problem. The evolved populations are randomly sampled, and simulated annealing is used to map these samples on to a 'similarity lattice'. The idea is that syntactically similar structures should be close by each other in the lattice. Once fitness values have been overlaid, it can then be seen whether or not (and how) semantic similarity (demonstrated by fitness) corresponds to syntactic similarity (demonstrated by proximity in the lattice).

According to the initial claim quoted at the beginning of the paper, the neural network landscape should be significantly smoother, with gradual changes in fitness evident in neighbouring localities in the landscape. The genetic programming landscape should illustrate sharper precipices, a result of the higher level primitives being employed.

Other Visualisation Techniques

A number of multidimensional, and other, visualisation techniques have been employed for studying evolutionary fitness landscapes. Summaries of some of these can be found in [11]. The technique to be employed here is most similar to Sammon mapping, initially presented in [13] and described in application to evolutionary algorithms in [4]. It also bears much in common with the Sammon mapping derivative Genotypic-Space mapping [3].

Sammon mapping is a technique for detecting structure in a set of N vectors of dimensionality L, and mapping this structure on to a set of N 2- or 3-dimensional vectors. This is done by minimising the dissimilarity between the distance between vectors in the higher dimensional and lower dimensional vector sets. Any domain appropriate measure of distance can be employed, although most typically Euclidean distance is used. The level of dissimilarity is given by the equation:

$$E = \frac{1}{\sum_{i<j} \delta_{ij}} \sum_{i<j} \frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}}$$

where δ_{ij} denotes the distance between the ith point and the jth point of the higher dimensionality vector set, and d_{ij} denotes the distance between the ith point and the jth point of the lower dimensionality vector set. A steepest gradient descent method is then used to minimise this value.

There are a number of significant problems that arise when trying to apply Sammon mapping to the domain under consideration here. Firstly, suitable distance metrics must be found. This is particularly important for the evolved control programs, where even the dimensionality of the solutions does not remain constant. More seriously, the computational cost of comparing large numbers of points becomes rapidly prohibits any but the smallest landscapes to be generated in a reasonable amount of time. In fact, the computational complexity increases exponentially with the size of the mapping [3]. Additionally, the gradient descent method used has a tendency to get stuck on sub-optimal minima, as observed in [11].

The proposed solutions to these problems are discussed in more detail below. In summary, computational overheads are reduced by generating dissimilarity measurements only from local neighbourhoods (as suggested by [3], and stochastically sampling the candidate solutions, rather than comparing all possible candidates. Further, simulated annealing has been used to generate overall dissimilarity measurements significantly lower than those obtained using the normal gradient descent method.

Developing the Algorithm

The landscaping algorithm was initially developed using a simpler evolutionary model than the robot-navigation models described above. A simple bit-string genetic algorithm was employed. The exact algorithm used was the standard GA used in [2]. This was applied across a number of optimisation problems taken from the test suites employed in [2] and [1].

In each case the genetic algorithm was run, and a total of 400 (enough to fill a 20x20 lattice) bit-strings were randomly selected from throughout the run, chosen evenly from throughout each generation. These were initially distributed across the lattice at random.

In existing work applying Sammon mapping to evolutionary algorithms (eg [4]), the highest fitness member of each generation is generally used. However, the solutions generated by an evolutionary approach to a problem emerge from the interactions of all members of all populations throughout the course of a run. It is likely therefore that there will be characteristic features

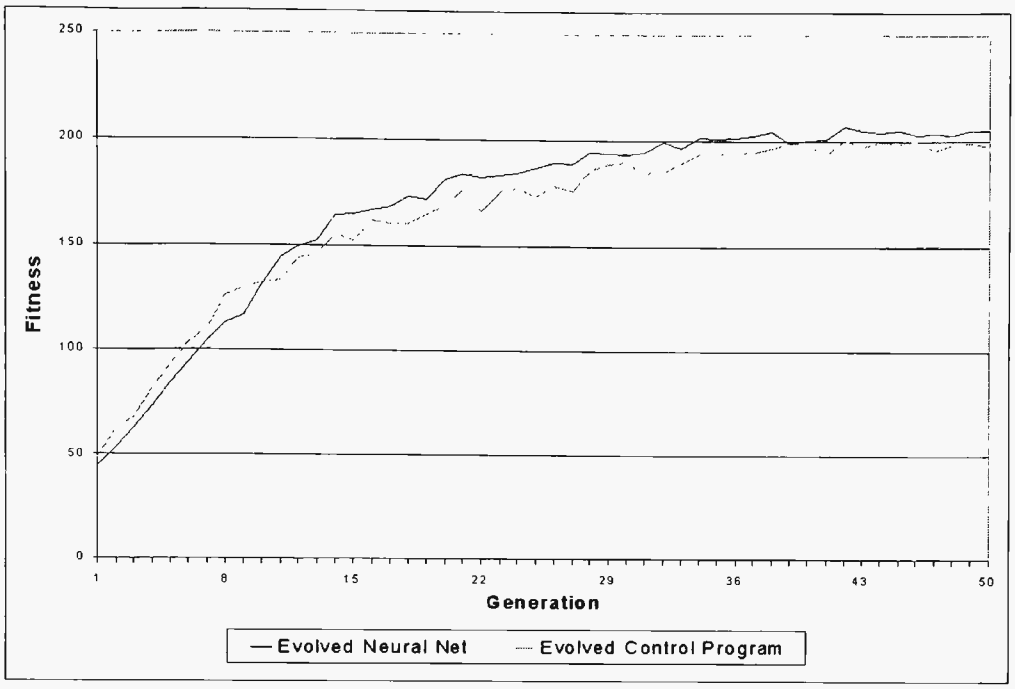

Figure 5: Comparison of fitness over time for the evolved neural networks and evolved control programs. Values averaged over 10 simulation runs.

of a particular evolutionary path that will not be captured simply by considering the fittest member of each generation. However, generating a mapping from all solutions generated throughout the course of a run is an extrememly computationally demanding task. For this reason, stochastic sampling of each generation was used, selecting enough members from each population to generate a reasonable landscape, while keeping computational overheads manageable.

The similarity of two bit-strings can be defined as the number of loci that are identical in each. The total similarity value of the lattice is then calculated by taking each cell in turn, and summing its similarity with each of its neighbours. The sum of all such neighbourhood similarities is the total similarity value for that particular lattice arrangement. Dissimilarity can be calculated as a simple inverse of this similarity value.

The simulated annealing algorithm was then utilised in order to minimise this dissimilarity value. A straight gradient-descent algorithm was also tested, but generally resulted in arrangements with lower final dissimilarity values. The simulated annealing algorithm worked as follows: two cells in the lattice were swapped at random, and the total dissimilarity of the new arrangement was calculated. The new arrangement was then accepted or rejected based on the change in total dissimilarity and according to a Boltzmann probability function. The following annealing schedule was used: the temperature was initially set to 5000, and reduced geometrically by a factor of 0.9 to a minimum of 1. The algorithm was iterated 1000 times at each temperature level, with a total of 150 iterations of this process. These parameter settings were determined largely by trial and error, based on an initial algorithm taken from [1].

Once the simulated annealing was complete, the fitness values for each cell in the lattice were plotted to generate the final 3-D visualisation.

Landscaping the Evolved Control Programs

The next stage was to adapt the process as used in the simpler GA cases to the more complex evolution of the robot control systems. The only essential difference would be the similarity functions used. The functions used would have to reflect the ease with which one program or neural net could be transformed into another, something akin to an inverse of Hamming distance [6]. After some experimentation the following procedure was found to give reasonable results for the genetic programming population: for every sub-tree from the first program that is also a sub-tree of the second program, increase the similarity by one. As it stands, this would produce an asymmetric result, such that the similarity between x and y is not necessarily the same as the similarity between y and x. To rectify this, the same process

is then applied in the opposite direction. The procedure then becomes: for every sub-tree in the first program that is also in the second, and for every sub-tree in the second that is also in the first, increase similarity by one. This still has the slight problem that large programs will, through this definition, tend to produce higher similarity values than small programs. This is settled by normalising according to the combined lengths of the two programs and scaling up to give a value from 0 to 20 (this range chosen somewhat arbitrarily as matching the range produced by the earlier genetic algorithm test cases).

For the neural network chromosome, similarity could be defined much more simply. A sum was taken of the differences between the values of each gene in the two chromosomes being compared. This was then inverted and scaled to again give a value in the range 0 through 20.

Further, the additional complexity of applying the annealing algorithm while using the more involved genetic programming similarity function resulted in enormously increased computational demands. For this reason, and given the limitations of available resources, the annealing schedule was reduced somewhat. Each temperature setting was iterated only 500 times, and a total of only 100 iterations of this process occurred. For consistency this same schedule was used also for the neural network landscaping process. The final resulting fitness landscape visualisations can be seen in figures 6 and 7.

For comparative purposes, the evolved neural network landscape before application of simulated annealing to minimise the dissimilarity is also shown in figure 8. This landscape has effectively been generated by distributing evolved solutions across the lattice at random, and then plotting their fitness values.

Discussion

The Landscaping Algorithm

In the initial development of the landscapiong algorithm, characteristics of each of the fitness functions being tested were evident to some degree in the generated visualisations. In 'plateau' type landscapes, wide, flat areas, gave way to narrow ridges, leading to sharp peaks. When fitness changes in these landscapes it does so sharply, but with clear paths still progressing from plain to ridge to peak. Other 'linear' type landscapes produced visualisations that undulated up and down smoothly, but with peaks rising progressively across the course of the visualisation.

These test cases are certainly very suggestive that key features of the underlying structure of the fitness landscape are being captured by the visualisation. However, a lot more work is really needed on a wider variety of problems to really determine the extent to which the mapping algorithm is capable of capturing the structure of any sort of complex landscape.

Comparing the neural network visualisation with the randomly generated landscape using the same data is also suggestive that the algorithm is capturing at least some relevant features. Although any interpretation of the landscapes is a highly subjective matter, there certainly does appear to be a tighter grouping of peaks, rising steadily from lower regions at the extreme corners of the final visualisation, than in its random counterpart.

The Robot Control System Landscapes

Two important differences can be noted in the landscape visualisations. In the first instance, as noted above, it can be seen that the peaks for the neural network approach are far more tightly bunched than those on the genetic programming landscape. Secondly, it can be seen that the genetic programming landscape is criss-crossed with numerous deep crevasses and canyons. Overall, there can certainly be seen to be more structure to the neural network landscape, suggesting a less problematical search space for a given evolutionary algorithm to navigate. These observations lend credence to the conventional evolutionary robotics wisdom.

Some further analysis in conjunction with figure 5 can be speculated on. Initially, increases in fitness in both systems are fairly level. In both landscapes very low regions are adjacent to mid level regions. However, the evolutionary paths needed to reach peaks can be seen to be somewhat different in each case. Many of the mid level plateaus in the genetic programming landscape are separated by crevasses from peaks, where as the neural network landscape rises more consistently towards the highest peaks. This may help explain the slightly higher mean fitness levels achieved by the neural nets after mid-level fitnesses have been achieved.

However, it certainly seems plausible (if not likely) that key characteristics of the fitness landscapes in this particular situation are a product at least as much of the particular fitness function as of the evolutionary control strategy. For a proper comparison, a much wider range of control problems should ideally be addressed. The particular corridor following problem addressed here will tend to produce fitness plateaus, with large jumps in fitness occurring as each bend in the corridor is reached and passed. Other types of problem may produce different structured fitness dynamics that will interact in different ways with opposing evolutionary strategies. Only if the two approaches produce similar landscapes on a variety of problems can it really be said that the original claim is supported.

It should perhaps be emphasised that any analysis of generated landscapes at this stage is really very much a subjective and speculative process. The process used, however, has still demonstrated some value in suggesting possible lines of research into understanding the behaviour of the specific domain under consideration. Fur-

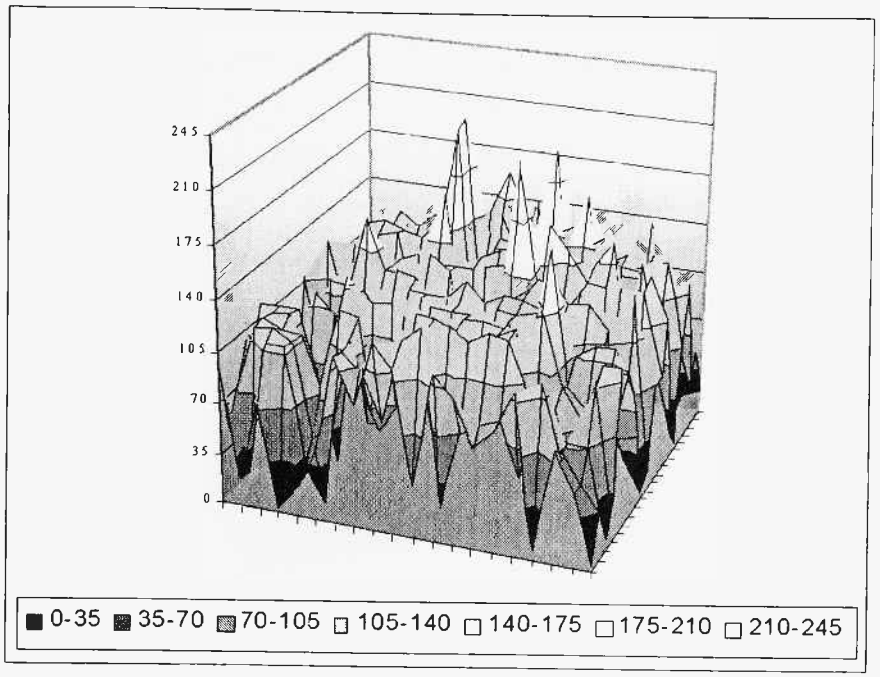

Figure 6: Fitness landscape visualisation of evolved neural net robot controllers

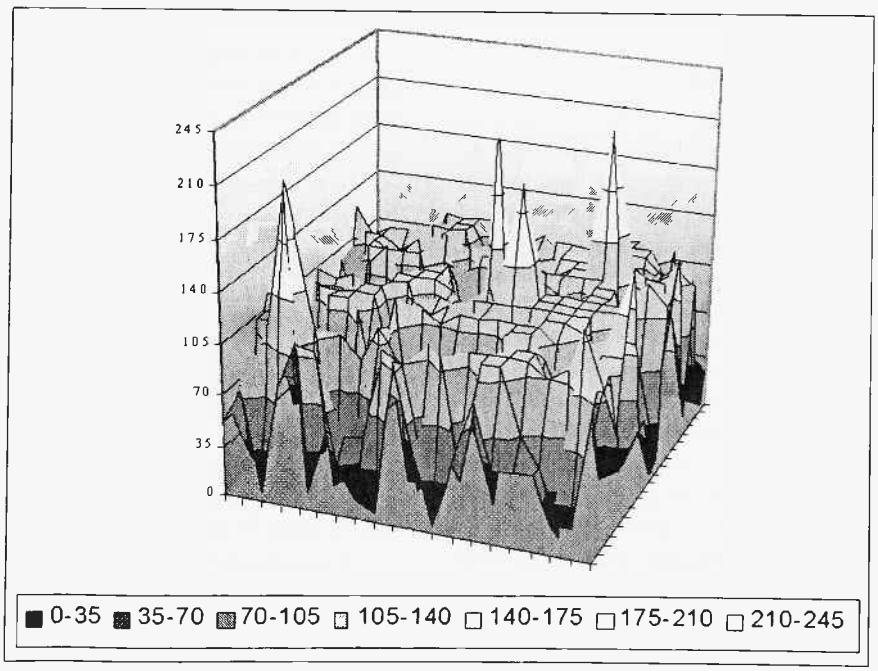

Figure 7: Fitness landscape visualisation of evolved high level robot control programs

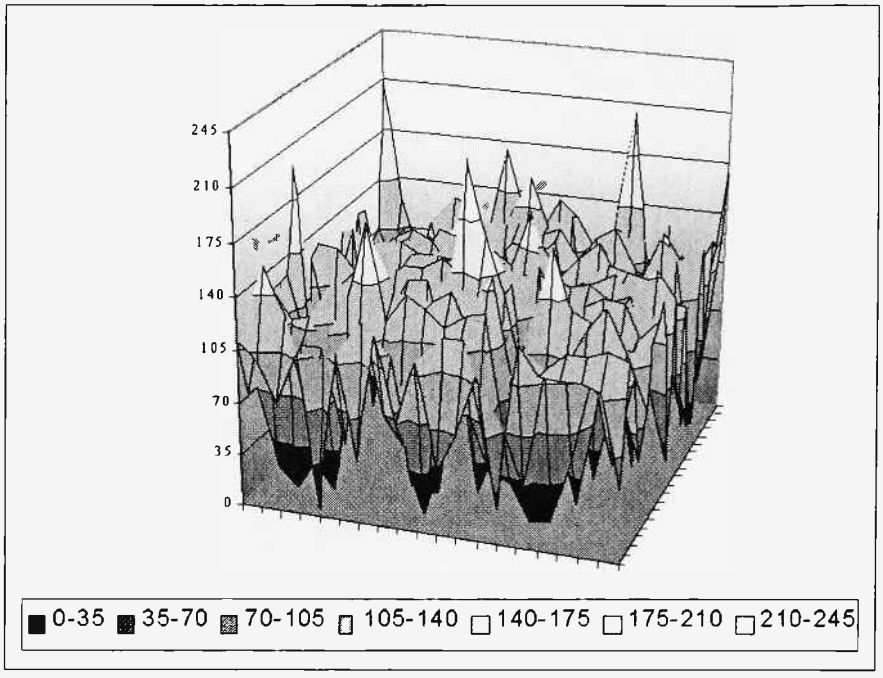

Figure 8: Random landscape visualisation of evolved neural net robot controllers

ther work in analysing the resultant visualisations is intended to quantify more specific properties.

Conclusions

The two approaches taken to evolving corridor following behaviour in a simple autonomous robot resulted in a variety of different solutions, varying in their level of success. However, the desired behaviour was evolved, using both genetic programming and evolved neural networks. A method for visualising evolutionary fitness landscapes was then demonstrated, and applied both to some sample genetic algorithm problems, and to the evolution of the two differing types of robot control system. The resulting landscapes provide some (although very inconclusive) support for the claim that genetic programming techniques result in fitness landscapes poorly suited to evolutionary robotics in comparison to evolved neural nets. In particular, the visualisations suggested one possible (though again highly speculative) reason why neural nets had improved performance over control programs only for moderate to high fitness levels. Further investigation, directly involving a wider selection of problems, would absolutely be necessary to establish anything more conclusive. However, the methodology employed here does still hold promise. The techniques developed could be used as one possible tool in this sort of a more thorough investigation.

Acknowledgements

Thanks to James Neil for comments on the fitness landscaping algorithm. Supported by funding from the Engineering and Physical Sciences Research Council. Work carried out in part at the School of Cognitive and Computer Sciences, University of Sussex.

References

[1] Ackley, D. H.: Bit Vector Function Optimization. In L. Davis, ed., *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann, 1987.

[2] Anastasoff, S. J.: Evolving Mutation Rates for the Self-Optimisation of Genetic Algorithms. In Floreano, D., Nicoud, J. D. & Mondada, F., eds. *Proceedings of the 5th European Conference on Artificial Life*. Springer-Verlag, 1999.

[3] Collins, T.: Genotypic-Space Mapping: Population Visualization for Genetic Algorithms. Technical Report, KMI-TR-39, Knowledge Media Institute, the Open University, Milton Keynes, UK, 1997.

[4] Dybowski, R., Collins, T. D., & Weller, P.D.: Visualization of Binary String Convergence by Sammon Mapping. In Fogel, D. B., Angeline, P. J., & Back, T., eds. Evolutionary Programming V, Proceedings of the Fifth Annual Conference on Evolutionary Programming. MIT Press, 1996.

[5] Harvey, I., Husbands, P., Cliff, D.: Issues in Evolutionary Robotics. In Meyer, J., Roitblat, H. L., Wilson, S. W., eds. *From Animals to Animats 2, Proceedings of the Second International Conference on Simulation of Adaptive Behaviour.* MIT Press, 1993.

[6] Kauffman, S. A.: *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, 1993.

[7] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[8] Koza, J. R.: *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[9] Lee, W. P., Hallam, J., Lund, H. H.: A Hybrid GP/GA Approach for Co-evolving Controllers and Robot Bodies to Achieve Fitness Specified Tasks. In *Proceedings of IEEE 3rd International Conference on Evolutionary Computation.* IEEE Press, 1996.

[10] Nolfi, S., Floreano, D., Miglino, O., Mondada, F.: How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. In Brooks, R. A., & Maes, P., eds. *Artificial Life IV. Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems.* MIT Press, 1994.

[11] Pohlheim, H.: Visualization of Evolutionary Algorithms – Set of Standard Techniques and Multidimensional Visualization. In W. Banzhaf, et al., eds. *Proceedings of the 1999 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 1999.

[12] Reynolds, C. W.: Evolution of Corridor Following Behavior in a Noisy World. In Cliff, D., Husbands, P., Meyer, J. A., Wilson, S. W., eds. *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior.* MIT Press, 1994.

[13] Sammon, J. W. jr.: A Nonlinear Mapping for Data Structure Analysis. In *IEEE Transactions on Computers*, vol. C-18, pp. 401-409, 1969.