Evolvable Morphologies for Robot Controllers

Jeremy Stober Jonah Gold

16 May 2003

Abstract

We describe an experiment wherein we evolve network morphologies to act as robot controllers for simple tasks. These tasks include color following, wall following, and puck eating. We successfully evolved several effective controllers using a genetic algorithm and populations of genotypes that encoded the morphologies of the network controllers subject to evolution. For each of the tasks, we ran a baseline comparison with a more traditional, fixed two-level feed-forward network architecture. For our baseline comparison, we evolved connection weights using a standard genetic algorithm. We found that the evolved morphologies performed similarly to the baseline case, despite the profound underlying differences between the two approaches.

1 Introduction

Designing algorithmic solutions to the problems that we want computers to solve becomes rapidly less feasible as the complexity of problems increases. This is particularly true in robotics, where many of the behaviors we eventually wish robots to have, such as complex space navigation or object recognition, are not well understood even in biological organisms.

For this reason, the connectionist paradigm is an appealing one for robot controller design. Connectionist networks are ideally-suited to be used in conjunction with methods for automatically acquiring general solutions to problems; principally learning algorithms and evolutionary algorithms. These are typically applied to a fixed network architecture, to search the space of weights between network nodes. These types of methods are excellent for solving many problems more efficiently than they could be solved by human-designed solutions.

But even the use of developmental and evolutionary methods on fixed network architectures is limited by the architecture itself, which is in this paradigm necessarily chosen by a human, and thus may not be well-suited to the problem at hand [1]. This occurs very often in robotics specifically, in the problem of parsing of sensory input into a useful form. Distal sensing processes, which in some way compress the sensory world in which the robot moves, are often required for efficient navigation, but it quickly becomes difficult to ascertain what kinds of architectures will provide useful segmentation of complex sensor data [4]. As we see the limitations of fixed architectures in many instances such as this, a new problem arises: how to find an appropriate architecture for neural computing processes.

For this problem, an obvious and powerful

method presents itself: the genetic algorithm. Inspired by natural evolution of organisms, we may extend our consideration of what is being evolved to include not only the configuration of the neural architecture, but the architecture itself [4]. This gives us a powerful tool for expanding our search space.

Balakrishnan and Honavar offer concepts that can be used to describe a method for representing neural network architectures genetically [2]. A representation is considered to be *complete* if every architecture constructible in the system has a genetic representation. It is *closed* if every genetic representation produces a valid phenotype architecture. It seems fairly important that representations be at least mostly closed, as otherwise much genetic search time will be wasted constructing genotypes that have no possibility of being a good problem solution. Representations that are significantly far from completeness are limited in strength as well, as parts of their search domain cannot be reached. Both of these concepts were interesting to consider as we decided how to approach the problem of developing a genetic system for evolving network architectures.

There are two basic approaches one might take in the evolution of network architectures. The first, exemplified by Harp, Samad, and Guha's GENESYS system [5], is to encode within the genotype only the information for the topology of the network: its layers and connections, but not the weights of the connections. Then, within the fitness function, one can incorporate some other method of acquiring weights, such as back-propagation learning. This is potentially very powerful, and also allows for a more biologically plausible model of the interaction between learning and evolution, a paradigm that has already been shown to be useful, without evolving architectures, by Nolfi and Floreano [6]. However, systems like GENESYS have some problems. They are very computationally costly, for obvious reasons; at every iteration, both the computations required for a genetic algorithm, and the computations required for back-propagation learning (or another similar method) are required. Further, the requirement that the network must be one on which a method of weight acquisition can be applied might impose limitations on how the valid phenotype space may be constructed. In GENESYS, the choice of back-propagation for developing weights means that the only valid phenotypes are those networks that are strictly feed-forward, without recurrence. This limits the types of interesting networks that may occur, and is a serious restriction on the search space for the problem. In the language of Balakrishnan, in order to give their system decent closure, they must restrict the genotype space so that it matches the phenotype space.

The second style of genetic system, as we see it, is that of Husbands, Harvey, and Cliff [4]. In their system, the key principal is open-endedness. Their genotypes encode both the network architectures and the weights of the connections. They have two possible "weights": "excitatory", which serves as a single fixed positive weight, and "veto", which cancels all other weights to a node, if active. The basic unit of architecture is a node, rather than a layer; although input and output must obviously be specified, the structure of the hidden area is left free to develop in evolutionarily beneficial ways. Because back-propagation is not used, any kind of internal structure is possible - it is closed and complete without having to restrict the types of genotypes that may be encoded. Nodes may connect to themselves or connect in recurrent cycles

among other nodes. This is a less powerful starting point, but potentially allows more flexibility and power for interesting architectures to evolve incrementally.

Of course, this methodology has the obvious and significant drawback of the loss of a real valued weight search space. A restricted system of weights means that many kinds of problems will take longer to solve, and it is entirely possible that for real-life applications a GENESYS-like system would be more powerful.

However, due to the openness and computational simplicity of the latter system, it is ideal for experimenting with the concept of evolutionary network morphology. For this reason, we chose a system modeled on Husbands, Harvey and Cliff's for our experimentation.

Evolving network morphologies is a powerful method of great importance to the future of artificial intelligence, and one that lends itself well to application within robotics. It is also a method that has been experimented with comparatively little. In our experiments, we sought to develop a simple framework, based on a fixed-weights Husbands, Harvey and Cliff-like model, within which the possibilities of network morphology evolution may be explored.

2 Experimental Framework

Figure 1 shows the high level view of our experimental design. We implemented a system for encoding networks and defined genetic operations on those network encodings. We then used the Pyro robotics platform genetic algorithm together with task specific fitness functions to conduct the experiments discussed below [3]. For fitness evaluations, we used Player/Stage simulation software. [1]

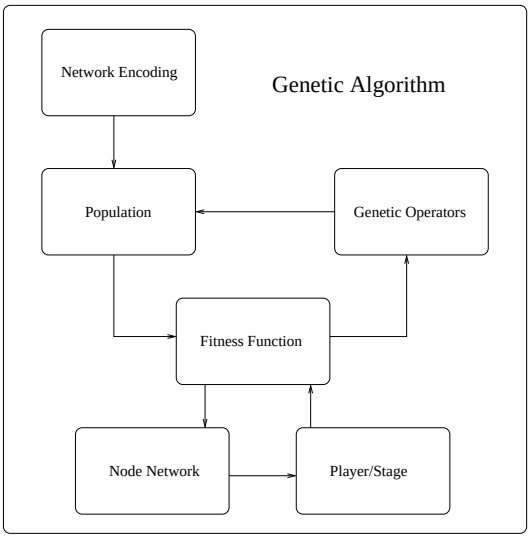

Figure 1: Experimental Design

2.1 Network Encoding

Our genotype is a list of node objects. Each node object containes a list of links associated with that node. Each link specifies the origin node, the destination node, and the link weight. Link weights are positive one or negative one, differing from the Husbands, Harvey, and Cliff approach of excitatory and veto weights.

The number of input and output nodes is specified at the time a population of network encodings is initialized. These remain fixed throughout evolution. The links owned by the input and output nodes are subject to mutation. The number of hidden nodes is also specified at initialization, but the number of hidden nodes is allowed to change through mutation and crossover.

[1] `http://playerstage.sourceforge.net/`

We implemented several types of mutation. The destination value for any link can be changed. Hidden nodes and links for any node can be added or deleted. We initialize added nodes with a random number of random links. If a node is added, links pointing to nodes past the added node's insertion point in the list are incremented so as to preserve the existing link structure of the network and to avoid orphaning the last output node. If a node is deleted, then the links to locations in the list past the deleted node are decremented so as to avoid broken links. Ease of dynamic link updating after genetic operations was the major impetus behind choosing a list of objects as our genotype instead of a faster bit encoding method.

For each instance of mutation, the type of mutation is determined by a list of probabilities. The default probabilities are 10% for adding a hidden node, 10% for deleting a hidden node, 20% for adding a random link, 20% for deleting a random link, 20% for changing a link destination, and 20% for changing a link value.

We use single point crossover with an updating routine to redirect any broken links. This routine consists of taking all links with invalid destinations and modding those destination values by the new network size. Other strategies for crossover may have preserved link structure in more sophisticated ways (see [5]), but for our purposes this simplified method of link destination updating sufficed.

Crossover is also prevented from disturbing the number of input or output nodes, although the input and output nodes themselves can be swapped.

The Pyro genetic algorithm module provides selection and population reproduction after every generation.

2.2 Node Network

We implemented a node network module to instantiate our network encodings as Con-x network instances. This novel extension of Con-x requires some explanation if the figures of evolved morphologies below are to make sense.

Each node in our network is a layer of size one in the underlying Con-x implementation. We do not use back-propagation. We form connections based on the order of the nodes in the encoding. For each node we iterate through the list of links owned by that node and instantiate connections based on that list. Two connections whose weights effectively cancel each other out do not necessarily imply that the target node will receive zero activation. The default sigmoid function in Con-x actually produces an activation of .5 in this case.

Activations are propagated in the order that the connections are instantiated. All inputs are set before propagation by directly copying the input activations to the designated input nodes. Due to this method of implementation it is possible for nodes early in the input series to overwrite the input activations of input nodes later in the input series. In practice, the genetic search method prevented such mutations from surviving long in the population.

The order of propagation affects what sorts of connections have meaning in the actual operation of the network controller. Such critical connections are in general hard to analyze and recurrences in the hidden layer prove to be even harder. Many recurrences in the input layer are meaningless as those activations would be rewritten every time step.

We use a direct translation from network encoding to functional network. In the language of Balakrishnan and Honavar, this scheme of-

fers no ontogenetic plasticity [2]. Without backpropagation and without any environmental context or stochastic process to shape mapping from genotype to phenotype, our system lacks additional features that may ameliorate some difficulties involved with pursuing this approach. We will discuss this in more detail in section five.

3 Experiments

3.1 Logic Network Design

As a preliminary measure of the capabilities of our system, we attempted to evolve neural network architectures to perform the basic boolean logic functions: AND, OR, and XOR.

We began our population encodings with the requisite two inputs and one output, three hidden nodes, and between zero and four connections attaching to a given node. Our initial population size was 100 genotypes.We used a mutation rate of .5, a crossover rate of .3, and an elite percentage of .1.

We tested each phenotype on each of the four possible combinations of binary digits: 00, 01, 10, or 11. The total fitness was intialized to four. For each binary 2-digit number, the difference between the network's output and the correct boolean function output was calculated, and the square of that was subtracted from the total. This yielded a fitness between zero and four, four being a perfect response and zero being a perfectly wrong response.As a measure to prevent brute force solutions, networks whose hidden node size grew larger than the arbitrarily chosen value of 7, or which contained an average of more than 7 links per node, also arbitrarily chosen, were automatically given a fitness of zero.

After evaluating the fitness of the population, the genetic operations, as specified, were performed to create a new population. Evolution was considered complete when a network attained a fitness number of 3.95 or higher, equivalent to an approximate average error of less than .1. Several trials were run for every logic function.

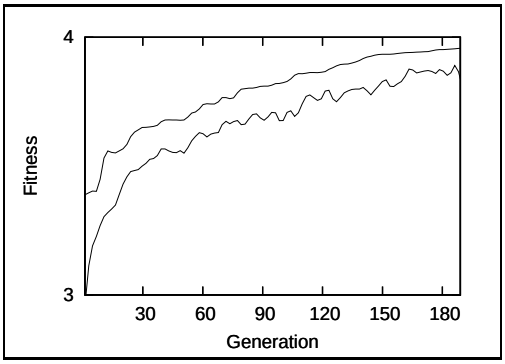

Figure 2: AND Network Evolution

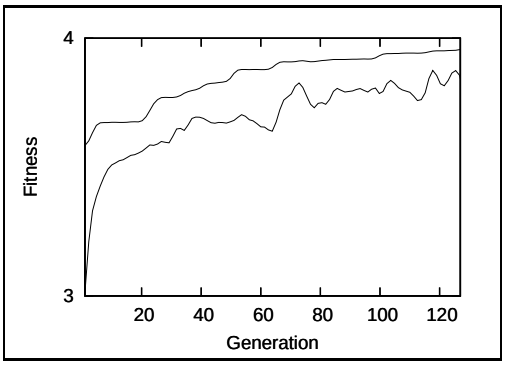

Figure 3: OR Network Evolution

3.2 Logic Network Results

3.2.1 AND and OR

The AND and OR networks were both comparatively simple to evolve. Both the population av-

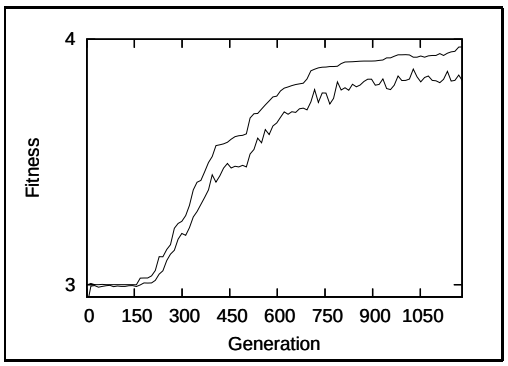

Figure 4: XOR Network Evolution

erage and the best genotype moved steadily and rapidly upward. The successful networks exhibited many of the same traits as the successful XOR network shown in Figure 5. Figures 2 and 3 show the simularity of problem difficulty in the AND and OR case.

3.2.2 XOR

XOR is famously a more difficult problem than OR or AND. This increased difficulty is reflected in the delayed progress apparent in Figure 4. Several attempts to evolve it yielded failure to obtain any results significantly better than a 50% chance of success. Though no rigorous analysis of what sorts of local minima were trapping solutions here was done, some spot-checking seemed to indicate the existance of such local minima. However, on many other runs we met with success, wherein a slight increase from these fifty-fifty odds in our population led to a slow, but steady upward increase, as depicted.

The winning network, i.e. the first to attain the requisite 3.95/4 fitness rating, is pictured, with dotted lines representing connections of weight -1, and solid lines representing connec-

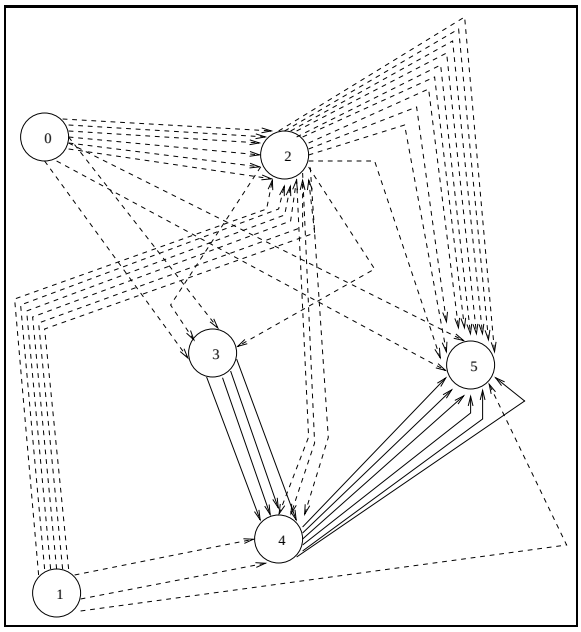

Figure 5: Winning XOR Network, Generation 1184

tions of weight 1. Input nodes are zero and one, and five is the output node. The network has only three hidden nodes, which is in part because due to problems earlier on with *too* many hidden nodes developing, the probability of new hidden nodes being produced by mutation was reduced. An interesting phenomenon resulted from this in the clustering of connections that is apparent here, as it was in practically all of our winning networks. Almost anywhere that a single connection can be found, more than one connection, of the same weight, can also be found. The largest set of connections between two nodes occurs between nodes two and five, with eleven weights of -1 linking the two. This can be seen as our networks evolving the notion of weight between nodes - networks select themselves in such a way that connections of a variety of strengths exist, in the form of double, triple, or n-times connections of uniform weight.

The network pictured in Figure 5 works in an interesting manner. If either zero, one, or both are activated, the weight going into two is such that two's activation becomes zero. Then if either one or zero is activated, the positive feed in from four is large enough to place the output's activation close to one. But if both are active, the additional connections directly from the input to the output, plus their negative influence on four, pulls the output's activation down again. And if neither one nor zero is activated, the negative input on two is small enough that two has some activation, which is multiplied over two the output through the eleven negative connections, pulling the output's activation down to zero.

3.3 Color Follow Task Design

We attempted to evolve a robot controller morphology to follow another colored robot in a simple environment.

For the color follow task we simulated two robots using Player/Stage. We used a rink environment for this task. The rink world represented a clear unobstructed space of about 14 square meters surrounded by a circular wall. For each fitness test, we placed one red robot in a random position in the upper right quadrant of the rink, oriented randomly and loaded with a simple pre-programmed avoid obstacles controller. We placed the red color follower robot in a random position in the lower left quadrant of the rink. We oriented this robot randomly within a range of 0 to 90 degrees, limiting the robot's orientation to facing the upper right quadrant. This robot had a color sensor whose field of vision was 180 degrees. We initialized this robot with the current network morphology being tested for fitness. The robots ran for 500 time steps.

At each time step Player/Stage returned a list of critical values used to calculate fitness. These included right, left, front, rear sensor readings, the range of the colored robot, the position of the red robot in the field of view and speed of translation. With the exception of translation speed, these were the same values presented as network inputs each time step. The controller output translate and rotate values.

If any of the robots four sensor readings indicated that the robot was blocked, the current fitness for that time step was set to zero. If the red colored robot did not appear in the color follower's field of vision, the current fitness for that time step was set to zero. Otherwise, current fitness was calculated to be the product of the speed of translation of the robot (normalized), the range of the colored robot (inverted and normalized), and 1 or .5 depending on whether the colored robot was central in the color follower's

field of vision.

Fitness at each time step was accumulated and after 500 time steps this total was returned to the GA.

After fitness calculations, a new population was produced as described in the experimental framework. We used a mutation rate of .3, a crossover rate of .1, and a elite percent of .1.

We ran five identical 200 generation trials of this experiment. The network population was initialized with eight input nodes, three hidden nodes, and two output nodes, with every node having a random number of random links up to four.

Additionally, we initiated a modified trial that included a parameter in the fitness evaluation that encouraged the minimal number of network nodes. After each fitness evaluation completed, we multiplied the resulting total fitness by the inverse of the size of the network.

We also ran a comparison trial using a fixed architecture network with eight input nodes, three hidden nodes, and two output nodes. The layers were fully connected and the weights on each connection were evolved using standard techniques.

3.4 Color Follow Task Results

We examined three sources of data for every trial. These included average and best fitness curves, observation of strategies for high fitness networks, and an analysis of the network morphology in each case.

Spiraling behavior was common in three of the high fitness controllers. The visibility of red caused modifications to behavior consisting of a longer spiral, increased translation, or slower rotation. Two controllers, from trial two and trial five, showed more complicated and more successful behavior than the simpler but less effective spiral strategy. In general, evolved networks demonstrated high complexity and many superfluous or unused connections.

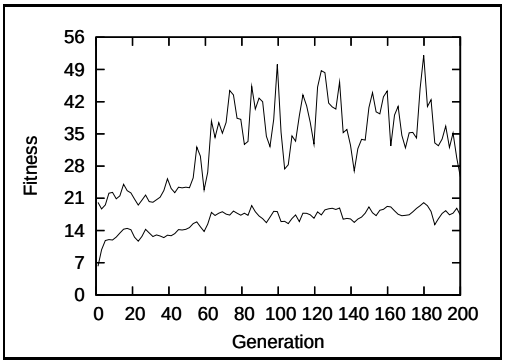

Figure 6: Trial One - Fitness

3.4.1 Trial One

Figure 6 shows the fitness of the best member of each generation (top line) and the average fitness (bottom line) across the entire population. Fitness increased rapidly in the first few generations. Beginning at generation 60, we see the best fitness line fluctuate markedly between values of 28 and 49 while the average fitness line remains fairly balanced at 14.

High fitness networks demonstrated counter clockwise spiraling behavior. The robot moved in tight spirals in the absence of red in the visual field. The robot increased forward translation when the robot saw red in the visual field causing the robot to traverse a wider spiral path. This strategy was prone to run into walls when started at the unfavorable angle of 0 degrees.

Figure 7 traces the strategy for the evolved controller. The dashed line represents the path of the evolved controller. The solid line repre-

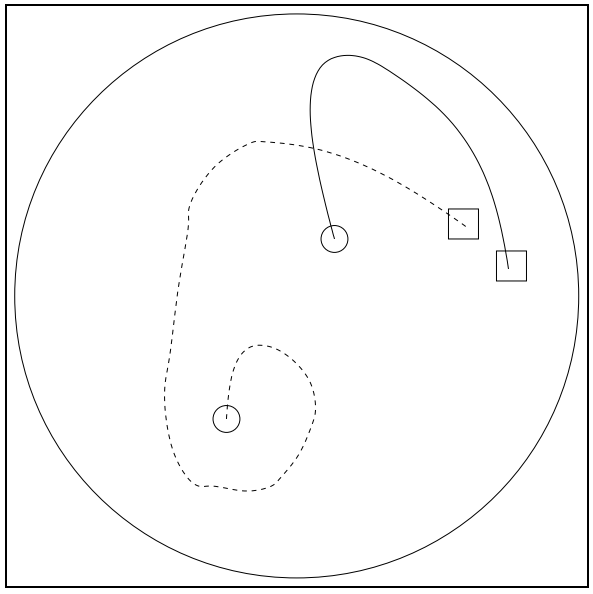

Figure 7: Trial One - Generation 121 Strategy

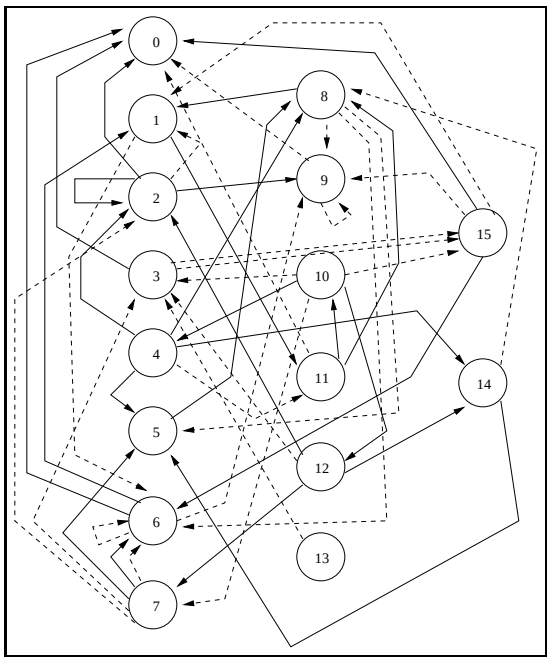

Figure 8: Trial One - Generation 121 Controller

sents the path of the red robot. The circles mark the starting location of both robots. The squares mark the location of the robots at the end the the observation. The spiral motion is clearly evident.

Figure 8 shows the network morphology evolved during this trial. This is one of the most complicated of the networks we observed. The dashed lines represent links with weights of negative one. The solid lines represent links with weights of positive one. The input nodes consist of all the leftmost nodes, zero through seven. The output nodes consist of all the rightmost nodes. The middle nodes are all hidden nodes.

Inputs for this task were mapped to the input nodes as follows: back sensors $\rightarrow$ node zero, front sensors $\rightarrow$ node one, left sensors $\rightarrow$ node two, right sensors $\rightarrow$ node three, color range $\rightarrow$ node four, color position left $\rightarrow$ node five, color position center $\rightarrow$ node six, color position right $\rightarrow$ node seven. The penultimate and last output nodes mapped to translate and rotate motor values respectively.

The connections from higher numbered input nodes to lower numbered input nodes are superfluous connections since, as previously mentioned, recurrent activations were overwritten every time step during simulation. Note that there is only a single connection, from node one to node six, in the other direction. Node six connects to node nine, whose connection to node zero is superfluous. So all connections among only the inputs nodes are useless. Vestigial complexity is a defining characteristic of all our results.

3.4.2 Trial Two

Figure 9 highlights the sudden discovery of a sustainable high fitness strategy after generation

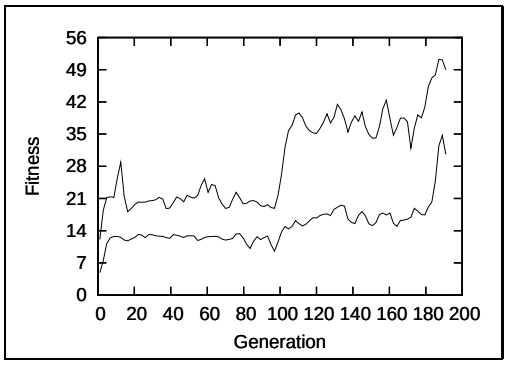

Figure 9: Trial Two - Color Follow Task

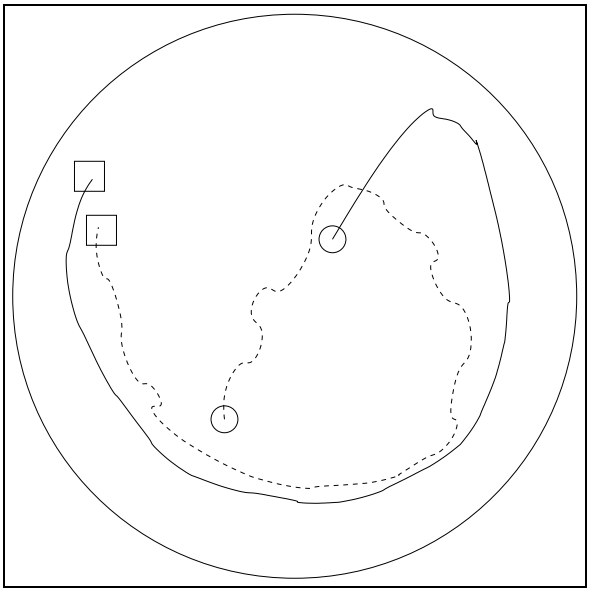

Figure 10: Trial Two - Generation 111 Strategy

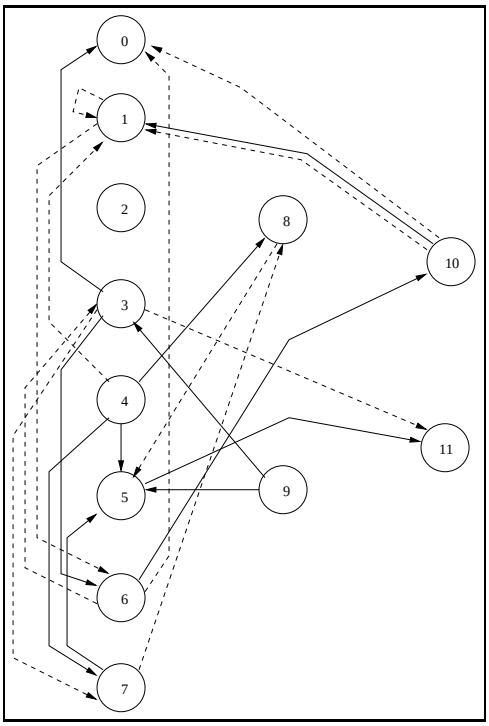

Figure 11: Trial Two - Generation 111 Controller

100. Such evolutionary leaps seem to be a central aspect of this approach. The evolutionary leaps are difficult to predict. Among the trials we ran for this task, this is the most drastic and sudden changes we observed.

In generation 101, the best network did not utilize rotation at all. Forward and backward translation were sensitive to the location of the red robot in the color finder. If red was not visible, or if red appeared in the right field of view, the robot would move backward. If red appeared in the center or left field of view, the robot would move forward.

In generation 111, we observed that the best network had incorporated rotational movements into its strategy (see Figure 10). The result appears to be optimal behavior, with the red color follower robot following the red robot in prototypical fashion.

This near optimal behavior involves a slight zigzag motion to keep the red robot centered, combined with a controlled forward translation to avoid staying too close the the rear of the red robot.

Figure 11 shows the network morphology of the generation 111 controller. The only connection between input nodes that is effective is the connection between node one and node six. Tracing this connection further we see that node six connects to node 10, so this connection clearly had relevance in the overall performance of the network.

We often found the node six's input activations were overwritten in many of the morphologies we analyzed. The network controllers gained greater fitness if red was centered in the visual field. That means that the left and right visual fields were critical to dynamically maintaining the position of the red color. The central field, however, lacked this critical importance and so was prone to this sort of observed overwriting.

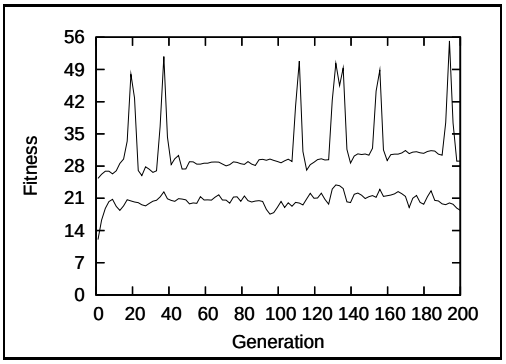

Figure 12: Trial Three - Color Follow Task

3.4.3 Trial Three

Figure 12 indicates the variety of fitness lines for this genetic search. We believe that the spikes in the best fitness curve represent highly unstable equilibria in the search space. The random starting locations also created limited variability between fitness functions. We hoped that this variability would lead to more general solutions, but the short bursts of high fitness could also be explained in terms of particularly advantageous starting positions. More significant perhaps, is the relatively stable flat average fitness line and the relatively small distance between the best and average fitness lines. This static fitness echoes the difficulties observed during the XOR task.

The high fitness network we examined demonstrated counter clockwise spiraling behavior. The spiraling reduced to rotation in the absence of red in the visual field. When red entered the visual field, translation increased, sometimes quite dramatically, while rotation remained constant. When red left the visual field, the slow

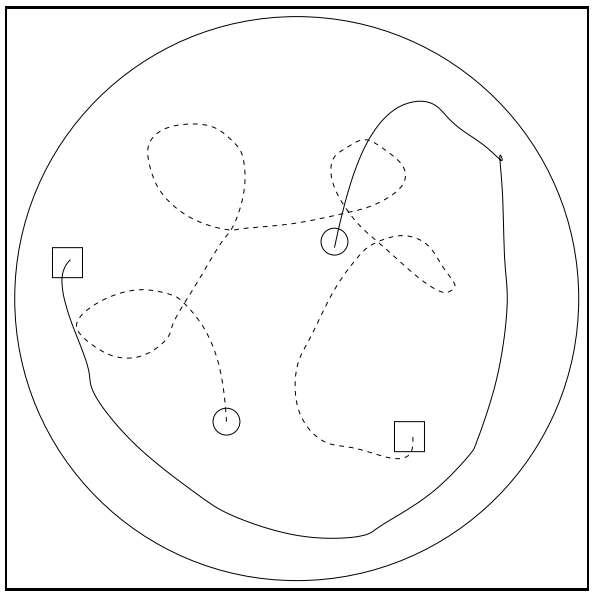

Figure 13: Trial Three - Generation 111 Strategy

spiraling behavior resumed (see Figure 13). The increase in translational speed when red was visible makes sense considering that fitness for high translational speed only matters in those cases.

The morphology for the network whose strategy is described above is shown in Figure 14. The only relevant connections in this morphology are those of input nodes three, four, and seven. Since node seven receives activation if color is detected in the right visual field, it is clear that forward translation is dependent on red in the right visual field. This in combination with the rotational tendency described above explains the robot's behavior. The double connection between node seven and node twelve effectively doubles the weight between nodes seven and twelve, explaining the marked increase in translation speed observed.

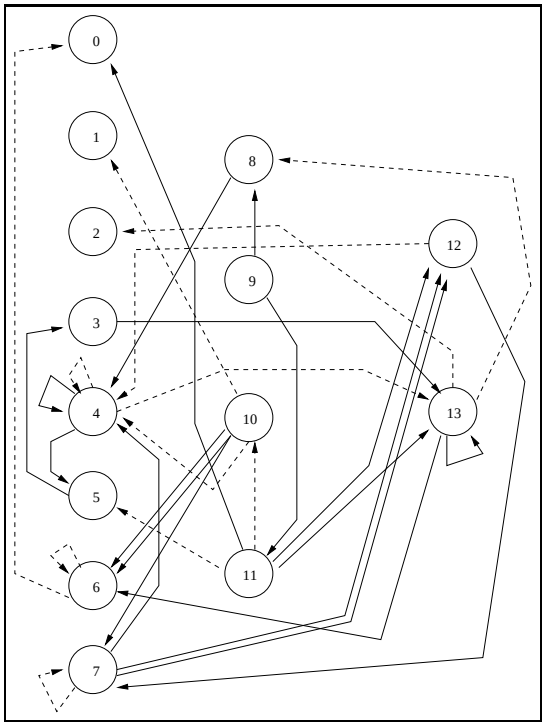

Figure 14: Trial Three - Generation 111 Controller

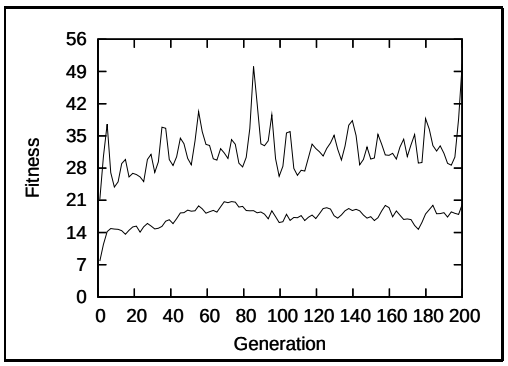

Figure 15: Trial Four - Color Follow Task

3.4.4 Trial Four

Figure 15 shows the fitness curves for this trial. Note that the best fitness line, while unstable in the short term, fluctuates throughout between the ranges of 28 and 35 with only a few deviations. Compare this with the previous trial. The differences between the peaks and valleys of the best fitness lines between trials vary significantly.

The strategy is shown in Figure 16. The high fitness network examined demonstrated counter clockwise rotation behavior without forward translation in the absence of red in the visual field. As red entered the visual field, rotation ceased and forward motion increased. As red color traveled toward the center of the visual field the robot began to rotate to facilitate centering of red in the visual field. The color follower passed the red robot and lost visual contact. Clockwise rotation resumed. When visual contact was again established, the color follower moved across the rink to intercept the red robot on the other side.

This behavior, while not considered prototypical for the task design, does exhibit the added benefit of effective searching. Given the initial

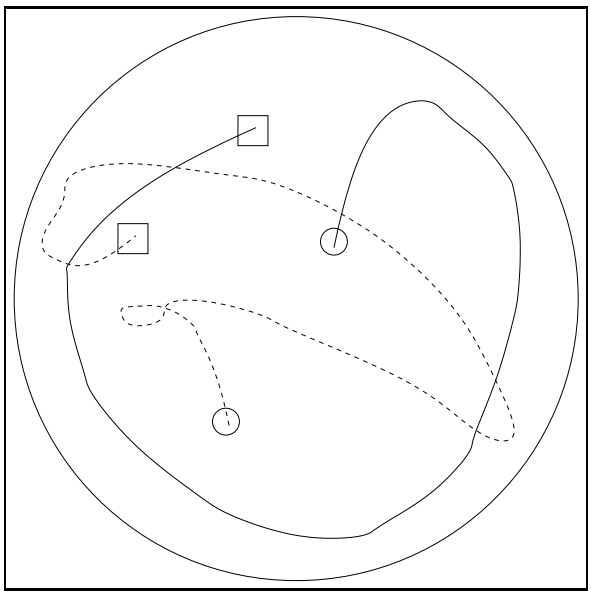

Figure 16: Trial Four - Generation 141 Strategy

randomness of the starting positions and orientation, any strategy that incorporates some aspect of search is more general than other comparable strategies.

Figure 17 shows the morphology of the network whose strategy is described above. Input nodes two, four, and seven seem to be the most relevant. The front and right inputs, nodes two and four, have opposite weighted connections to node thirteen implying that rotation is a function of the front and right sensors. This agrees with the observed behavior of the robot described above. Node seven, the input indicating red in the right visual field is connected with a positive weight to node thirteen, the output node controlling rotation. This node also has an indirect negative connection to node thirteen via hidden node nine.

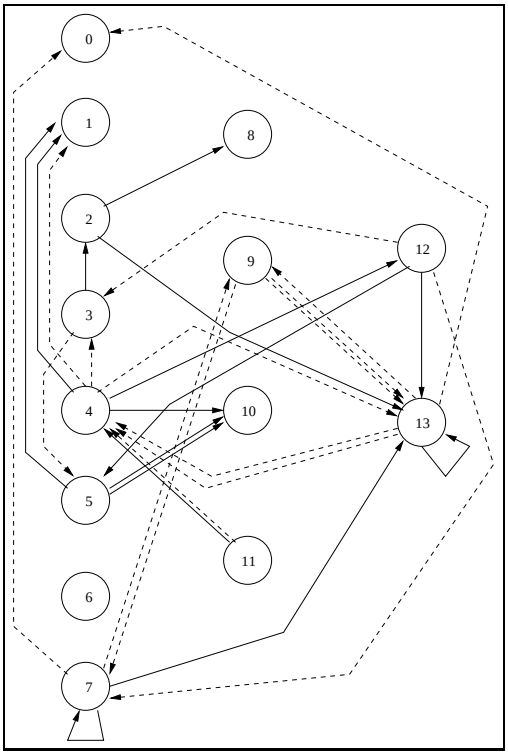

Figure 17: Trial Four - Generation 141 Controller

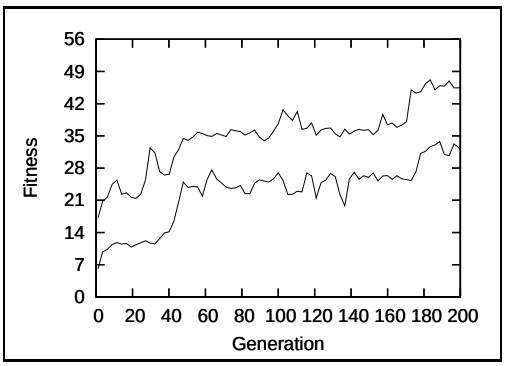

Figure 18: Trial Five - Color Follow Task

3.4.5 Trial Five

Figure 18 demonstrates the more incremental approach taken by the search in this trial over the dramatic rise in fitness found in trial two. Two points of sustainable sudden increase are observable around generation 40 and generation 160. The best and average curves are closely correlated across all 200 generations. This pattern of incremental improvement is indicative of the composition of small but effective changes disseminating through the population over a long period of time. Why this sort of incremental improvement is not evident in other trials is a question that we cannot answer.

Generation 161 shows similar near optimal behavior as that found in trial two, generation 111. The robot clearly attempts to keep red in the center of its field of view. The same zigzag strategy is employed with a bias toward counterclockwise over-rotation. Figure 19 traces the path of both robots in our observation case. This strategy proved slightly more effective then the strategy exhibited in trial two. The color following robot was more sensitive to frontal obstructions and was better able to maintain adequate and

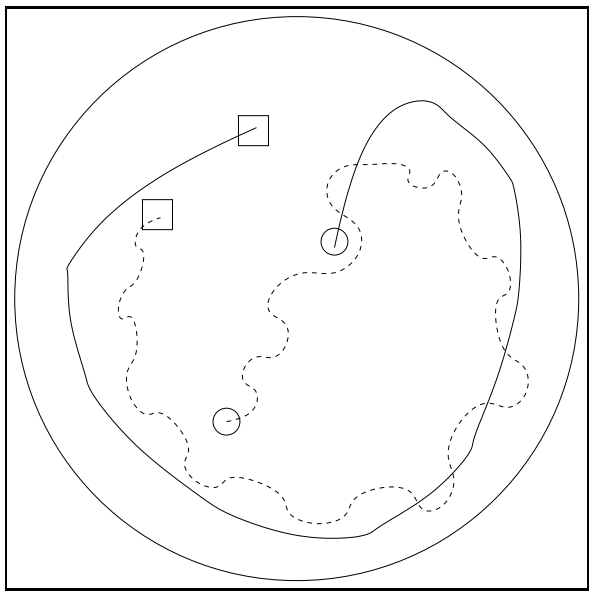

Figure 19: Trial Five - Generation 161 Strategy

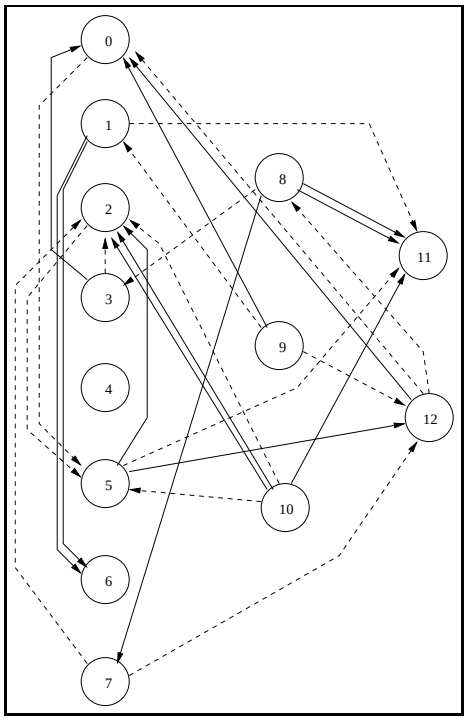

Figure 20: Trial Five - Generation 161 Controller

fitness enhancing distance between itself and the red robot.

The network morphology for trial five is quite complex. Despite the complexity, it appears that only nodes one, five, and seven have any direct relevance to the output (see Figure 20). Interestingly enough, all the hidden nodes receive no activation. Node one, the front sensor node, is inversely related to translation. This explains why the color follower robot did not get too close to the rear of the red robot in our observed case. The two connections from node five and node seven to node twelve also make sense in the context of the task. Node five and node seven represent the left and right fields of vision so their connection to the output for rotation helps explain the observed wiggle motion.

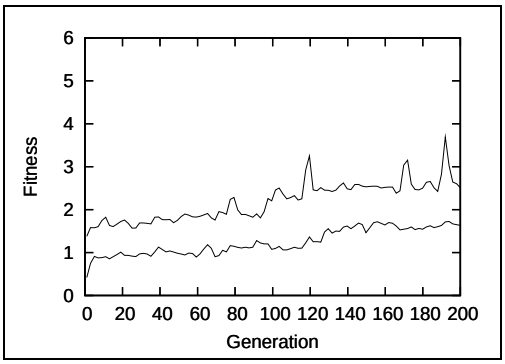

Figure 21: Minimal Network Trial - Color Follow Task

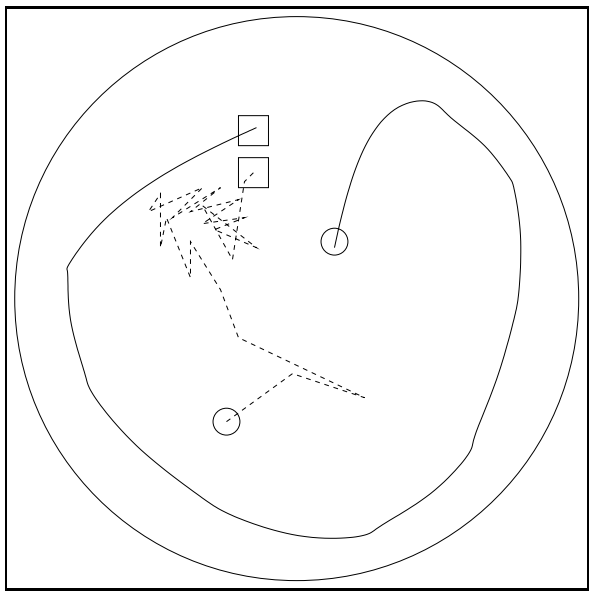

Figure 22: Minimal - Generation 111 Strategy

3.4.6 Minimal Network Fitness Function

For this trial we modified the fitness function so that minimal networks were rewarded. We did this by dividing fitness by the inverse of the size of the network. Since the minimal network size was ten, the fitness values are at least a factor of ten less than in other cases (see Figure 21). We did not limit the number of connections. Networks were allowed to evolve arbitrary numbers of connections without penalty.

The high fitness network we observed demonstrated k-turn behavior. When red entered the visual field the robot moved forward. When red left the visual field the robot moved backward. Throughout rotation was constant, leading to a k-turn behavior that resembled the spiraling behavior in effectiveness (see Figure 22). This behavior also leads to the secondary benefit of an effective search technique.

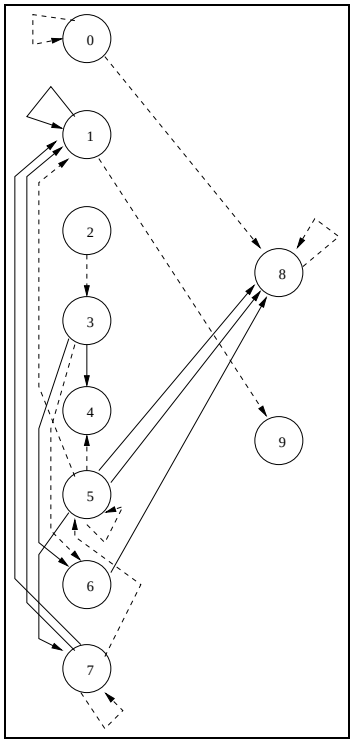

Figure 23: Minimal - Generation 111 Controller

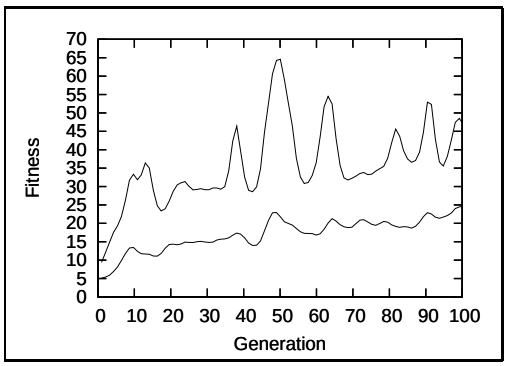

Figure 24: Comparison Network Trial - Color Follow Task

Figure 23 shows that the network morphology is clearly minimal. Output is sensitive to nodes zero, one, and five. The value of six is dependent on nodes two and three. Three's positive and negative connections to node six do not necessarily cancel due to the nature of the sigmoid function. An increase in forward motion with activation of the central field of vision is supported by the test observations. Additionally, the connection between node zero and node eight is of interest. Since this network controller's strategy involved backward motion, activation of the back sensor causing backward motion is obviously not conducive to a general solution.

3.4.7 Comparison with Evolved Weight Controller

Figure 24 clearly illustrates a smoother fitness curve in both the best and average cases. The smoother curve is representative of the relative stability of the two approaches. The network encodings showed considerable volatility in the very short term due to the nature of the genetic operations defined on the population members.

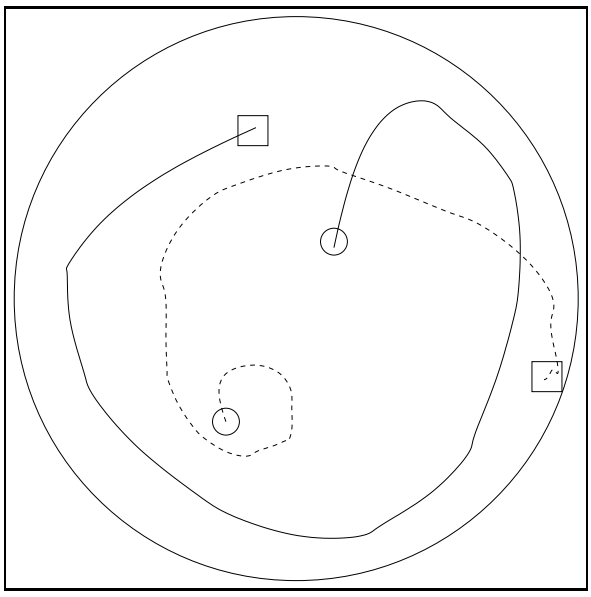

Figure 25: Comparison - Generation 64 Strategy

In the comparison case, the genetic operations on an array of real values are considerably less volatile. Over many generations, however, the volatility of both approaches is similar.

The fixed architecture controller evolved a similar spiral strategy to the network controller observed in trial one (see Figure 25). This seems to indicate that the effective evolvable strategies overlap between the two approaches. Since the fitness functions for both approaches were the same, this is not surprising. We expect that despite our interest in the effects of encoding representation, differences among approaches will be dominated by the choice of fitness function. Beyond these expected similarities, many interesting differences are manifest which motivate the discussion below.

3.5 Wall Follow Task Design

We designed a wall follow task to test our system's ability to evolve architectures to control a robot moving in a non-trivial space, staying close to walls but avoiding collisions.

The environment that we simulated our the robots' wandering in was based on a real-life blueprint for a section of a hospital. This blueprint was scaled up in size by a factor of 1.5, and some walls were removed and doorways widened to make it a space in which a Pioneer would better fit. Nonetheless, it contained several rooms of various shapes, corridors, and other interesting navigational features. The envrionment is depicted in Figure 26.

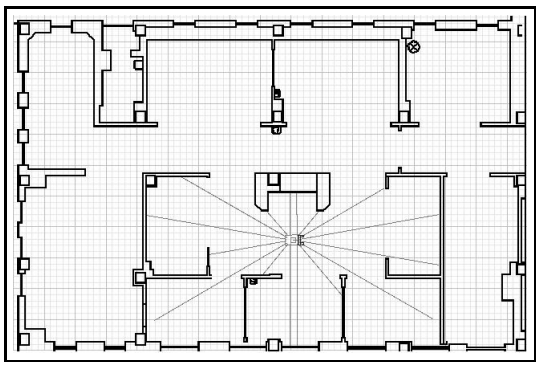

Figure 26: Pioneer in Wall Follow World

We wished to avoid starting robots in places that lead to obvious problems, such as on top of a wall, but wanted behaviors that could be applied to multiple starting locations rather than just one. Due to the complex nature of the hospital environment, there was no economical way to randomly choose a good starting location for our robot within the world. We made the compromise of picking nine starting locations for our robot, throughout the world. These were arbitrarily chosen, but to encourage productive movements early on, and thereby speed up our genetic algorithm, all had the robot's initial line of sight facing an open space rather than a wall, and left the robot a decent distance from a wall on all sides. Each phenotype being tested was placed in an initial position randomly chosen from this list of nine.

Our robot brain received four inputs; maximum sensor values from its front sensors, front-right sensors, front-left sensors, and rear sensors. Its two outputs were, as in the previous experiment, a translate value controlling its forward and reverse movements and a rotate value. Our initial population contained five hidden nodes and between zero and four links connecting to each node.

The robot was allowed to run for 300 time steps. Fitness began at zero, and was incremented every time step. The value it increased by in a time step was the product of the higher of the robot's right and left sensor values, which increase with proximity to walls, and the rate of the robot's movement at that step. To discourage local minima of backward movement, a robot moving backward halved its fitness.

If the robot's sensors detected that the robot was stalled for some reason, presumably by a wall, the robot received no fitness, and after 25 time steps of being stalled a phenotype's life ended.

After every phenotype's fitness had been determined in this manner, the genetic operations were performed, producing the next generation of genotypes to be tested. Testing continued for 600 generations.

We used a population size of twenty, and elite percentage of .2, a mutation rate of .3, a selection rate of .8, and a crossover rate of 0.1.

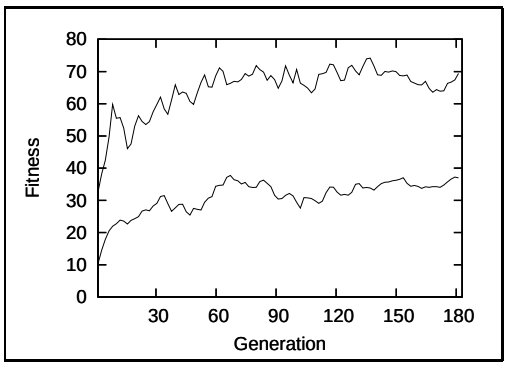

Figure 27: Fitness

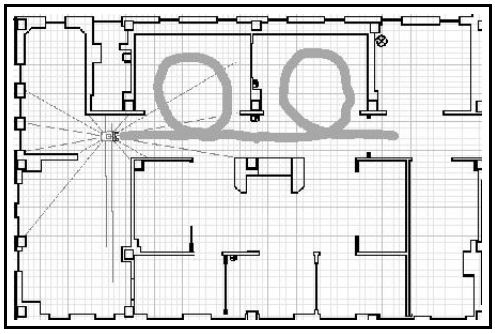

Figure 28: Generation 160 Strategy 1

3.6 Wall Follow Task Results

This experiment ran for slightly more than 180 generations. Throughout that time, the best network's fitness was dramatically higher than the average fitness for the population (Figure 27). This makes sense given the fact that the task essentially stopped, and the chance for fitness to go up declined, when a robot became stuck against the wall. Thus the winning robot was always one that, while still maintaining decent speed, managed to avoid collision for a high number of time steps. Both the best fitness and the average fitness fluctuated but appear to have

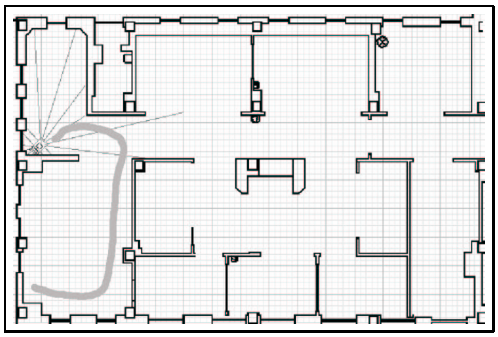

Figure 29: Generation 160 Strategy 2

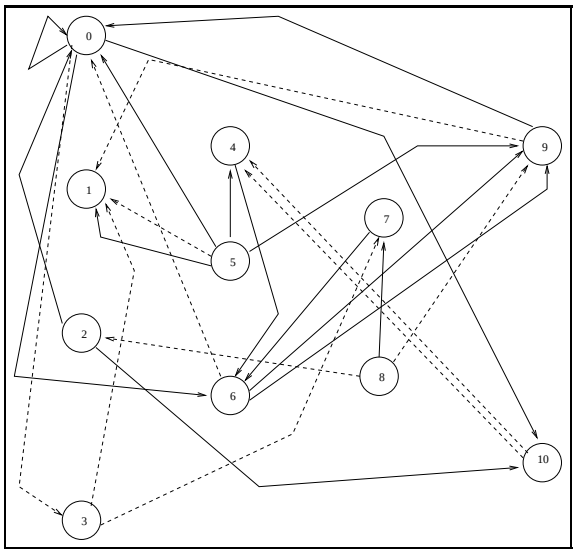

Figure 30: Generation 160 Controller

steadily but slightly increased, over this time.

The winning strategies tended to be something like one illustrated in Figure 28. The robot walked forward if its front sensors were unobstructed, and turned left with increasing rapidity as it came closer and closer to a wall in the front. This strategy worked well in many parts of the environment, but was not a very strong general strategy, as it left the robot with problems if it was headed towards a lower left hand corner straight on, as pictured in Figure 29. It is likely that the deficiencies of the developed strategy are due to the human selection of starting locations and the layout of the map that the robot explored. A more general solution, that might enable the robot to turn in both directions, could possibly be created by randomizing both of these to a greater degree.

The successful robot controller network from generation 160 (Figure 30) exemplified reasonably well the behavior found above.Input nodes zero through three take the back, front-left, front, and front-right sensor inputs, respectively, and output nodes nine through ten feed into the translate and rotate of the robot's motor. The front-left input is ignored altogether. Stronger input on the front sensor makes the robot turn left faster, as it should to avoid a wall using its strategy, and stronger input on the front-right makes the robot move forward slower, meaning it moves faster when nothing is in its right field of view. Its back sensors being activated also correlate with an increase in translation speed, which may be a coincedence, but also could be a result of the benefits of the robot's moving forward.

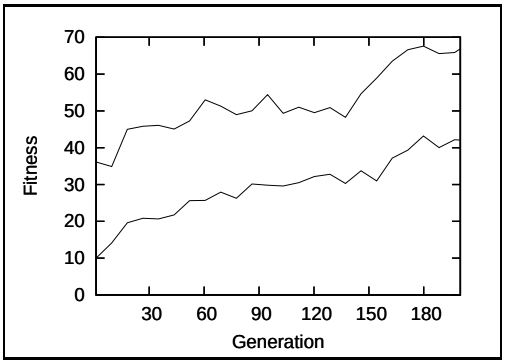

Figure 31: Fitness

3.6.1 Comparison With Evolved Weight Controller

The evolved weight controller, again, exhibits similar long-term performance, but is less susceptible to dramatic short-term change than our system (Figure 31). This if for the same reasons described in the comparison trial for the color following experiment; the genetic operators in our network encodings have more potential to change genotypes drastically,for better or for worse, then do those in the evolved weight genetic algorithm.

The strategies themselves were fairly similar, as one might guess from the similar performance. The successful evolved-weight controllers we looked at also only turned in one direction, did so faster as they approached a wall, and were in most other aspects similar to those of the evolve morphology controller.

3.7 Puck Finding Task Design

As a final experiment, we evolved a network architecture to control a robot searching for and "eating" red pucks in a simulated world.

The world was circular, identical in shape to

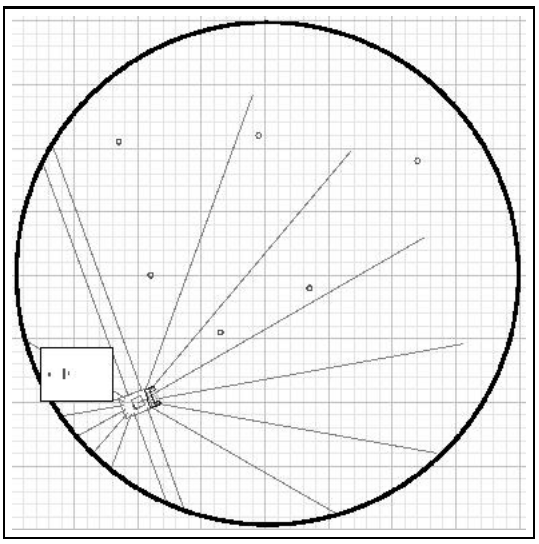

Figure 32: Pioneer in Puck Finding World

the "rink" world used in the color following experiment. In each new fitness test the robot, a Pioneer, began on the lower left hand side of the rink, facing toward its center. Six small, red pucks were placed at random locations within a square inscribed in the circular rink. The world is pictured in Figure 32.

The control networks' initial parameters were very similar to those of our color following networks described above. Four inputs contained values between zero and one representing the maximum sensor readings from the robot's front, front-left, front-right, and rear sonar sensors. The other four contained information from a preprogrammed blob-finding algorithm; one was a value between zero and one representing the range of the closest red blob, the other three coded for the blob's direction: front, center, or right. If the blob was not in the robot's front sensor range, all three of these contained a value of zero.

We began our population of twenty with three hidden nodes, and between zero and four links going to each node.

Each phenotype was allowed to run over 300 time steps. On each time step, the robot's fitness, which started at zero, was incremented by the robot's forward translation rate if a red blob was centered in the robot's field of view, half this if a red blob was to the left or right side of the robot's field of view. If the robot's blob finder did not detect a red blob in a time step, the robot received no fitness for that time step.

The robot's gripper was pre-programmed to automatically close about and store any puck that came into contact with it. A puck that was stored in this manner was removed from the world, and did not return until the next phenotype was tested. A robot that successfully "ate" a puck in this manner received 20 fitness for that time step, a far greater value than the maximum it could receive in a time step by simply heading toward a puck. This was meant to loosely mirror a real life organism's need to come very close to food in order to eat it and survive.

A phenotype that collided with a rink wall was halted; it did not receive a chance to back up and continue accruing fitness.

After each controller genotype was tested, the robot was returned to its initial starting position, and the pucks were again randomly repositioned.

Upon the completion of a generation, the usual genetic operations were employed to produce a new generation. We used an elite rate of .1, a mutation rate of .3, a selection rate of .8, and a crossover rate of .1.

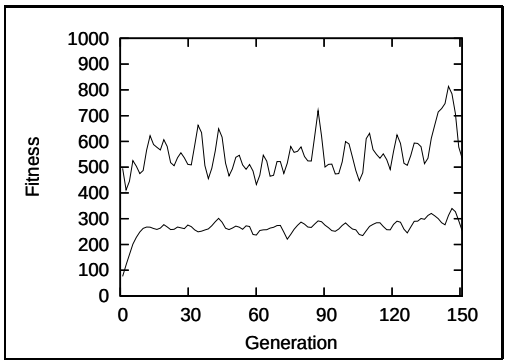

Figure 33: Fitness

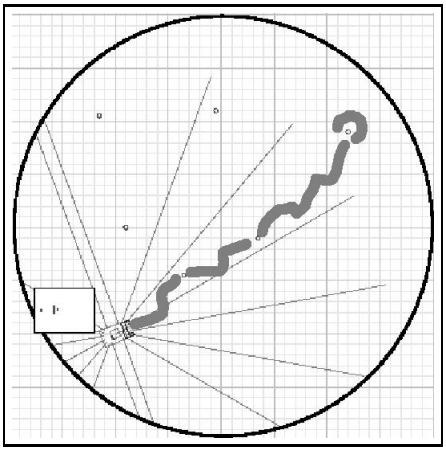

Figure 34: Generation 130 Strategy

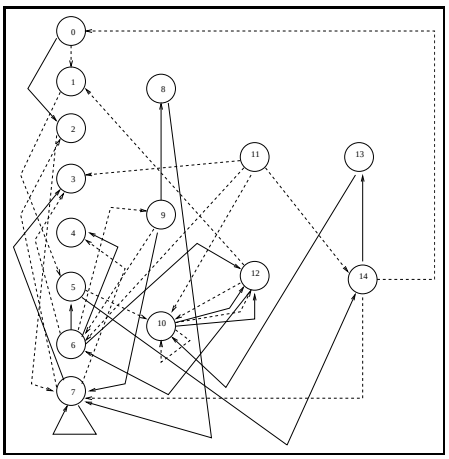

Figure 35: Generation 130 Controller

3.8 Puck Finding Task Results

3.8.1 Evolved Architectures

The puck finding task proved easy to do fairly well, but hard to achieve long-term significant performance increases after initial success. An initial upward surge in the first ten generations lifted the average performance of the robots on the puck-finding task to approximately where it would remain for the rest of the run (Figure 33). The sharp spikes in the best population member may again be attributed both to the amount of luck inherent with a task featuring random positioning and the powerful nature of our genetic operations between generations. The average fitness spikes much less than the best, predictably.

The basic behavior of all winning phenotypes observed was fairly simple. The robot would rotate until a red puck came into its field of view, and then make zigzagged motions, the one depicted, towards the red puck. There was some variance in the size of the zigzags, and also in speed of the robots forward motion. A few ob-

served robots, including the network whose architecture is depicted, would sometimes cease all forward movement and just rotate when near a wall. This served as a simple form of collision detection/avoidance, although the architecture depicted did not reliably start moving again when it was in the vicinity of the wall after rotating - sometimes it just kept turning. The behavior is traced in Figure 34.

The network architecture is pictured in Figure 35. As nodes 4, 6, and 8 (blob range, front blob detection, and front-right blob detection, respectively) lead to dead ends, only the front-left blob data is used in tracking the pucks. This makes sense with a little bit of analysis of how the robot moves. It moves right when no red blobs whatsoever are in its field of view. Then the constant negative input from node 11 to the rotate node, node 14, dominates the positive input from node five in the likely event that there is some activation on the front-left sensor. If there is input in the left blob, the robot will instead rotate left, towards the input it is detecting. This will continue until it has moved such that the input is slightly outside of the left blob detector, at which time it will start to move right again. The entire time, it moves forward at what is close to a constant rate, influenced only slightly by last cycle's translation. Note that this strategy does indeed incorporate a method for avoiding walls - if the robot does not detect any blobs, which is likely if it's facing a wall, it will keep turning right and, hopefully, eventually turn around, find a blob, and head toward it.

The evolved weight controllers again performed comparably to the evolved architectures on this test, though this time they had the slight edge, managing some slow increase rather than basically flatlining after the first few generations. Predictably, their curves were smoother. The reason for their advantage here is unclear, but might be explained by the amount of input nodes in this task and its comparative simplicity - the pre-connected fixed architecture might have a fairly simple problem, while the evolved architecture must evolve its connections so as to use its input nodes in useful ways, amplifying the complexity of its search space.

The successful strategies in the evolved weight controllers were quite similar to the evolved architecture - rotate until a puck comes into view, and zig-zag towards it. The heightened fitness seems to have mostly been attained not by a more efficient collection method, but through faster movement using the same collection method. The controller we examined made significantly larger zigzags, at a faster rate, than any of the evolved architecture specimens. Its wall avoidance didn't seem to be much better, though - it also experienced problems wherein it tried to stop and turn due to a wall, but was trapped by the wall and turned back and forth repeatedly until it bumped into the wall and "died".

4 Discussion

The current status of theoretical and experimental knowledge in the area of evolving network morphologies makes thorough analysis difficult. As with most methods in connectionist and evolutionary artificial intelligence and robotics, most guiding heuristics are gained through experimentation. Understanding the theoretical underpinnings of such a complex system is often a difficult or impossible task. This is the case with the system we chose to explore. We can identify weak points and strong points in the method, but our understanding of the causes of

the deficiencies or successes is still very limited. We are left to conjecture on the important results and speculate on possible modifications.

Our system demonstrated remarkable variation in evolution across the multiple trials of the color follow task. Not only did strategies vary widely, but the fitness curves differed dramatically between trials. One hypothesis, partially verified through empirical observation, is that the initial random population determines what strategy will be adopted and refined throughout evolution. Each initial population in our system consists twenty essentially random controllers, i.e. twenty different random strategies. The best strategy quickly percolates through the rest of the population via selection and crossover.

As described, this variety of effective behaviors was less prevalent in the puck find and wall follow experiments; successful robots acted in more or less the same ways more often than not. As with anything involving evolved networks it is hard to analyze, but this is probably because the solutions found for both of these were fairly simple to arrive at, and there were few other solutions of equal simplicity to be found. In this case, the constitution of the initial population is of less importance, as it is likely somewhere along the line the simple and effective mechanisms the solutions used, such as turning towards blobs and turning left at walls, will be found by mutation.

As we saw in trial two, sudden monumental evolutionary jumps are still possible. But the question still remains as to whether such sudden increases in fitness, no doubt the result of a few key genetic operations, will scale to larger and more complicated problems, where a single extra node or connection will not have a large impact on the overall system.

This observation touches on two important issues, scalability and granularity. Our system's genetic operations are fairly unstable relative to the size of the encodings being evolved. Adding or deleting a node changes a significantly larger percentage of the network than mutating real valued weights in a much larger population of weight arrays. The granularity of our system is relatively coarse compared to the fixed architecture comparison runs.

With larger networks, and more complicated problems, this granularity would become less apparent and may even aid scalability by increasing the speed of the genetic search in the larger space. It is precisely these larger, more complex problems that we envision appropriate for a system such as this. The elimination of direct human design may be necessary to facilitate general solutions in highly complex problem domains.

5 Conclusion

Real evolution is much more complex than the simple sytem we utilized above. While incorporating more complex algorithms to facilitate genetic searches is computationally infeasable, several extensions and modifications to the current approach are worth considering.

Originally, we had planned on encoding our network architectures using parametric, phrase structure L-systems. Such a grammatical description proved difficult to construct, although L-systems may provide a method of advancing the translation from phenotype to genotype in the current method. By compressing the genetic representation of a network, either through L-systems or some other method, genetic operations can be constructed that operate on functional sub units of networks, instead of at the node level. We envision multiple levels of genetic change.

Additionally, a non-trivial mapping to phenotype might be implemented where genotypes are translated in the context of environmental or stochastic factors. This kind of extension would necessarily require a more complicated fitness evaluation.

Finally, as already discussed, in the long term it might be desirable to try work in a framework with both learning and evolution, evolving architectures on which the connection weights are learned through backpropagation. This obviously has precedent in biological systems, and allows for the strengths of evolution and learning approaches to interact in a beneficial manner.

The current technique yielded precisely what we had hoped. While we did not experiment with tasks that can be considered truly complex, simple robot controller morphologies proved to be interesting and readily evolvable.

References

[1] Karthik Balakrishnan and Vasant Honavar. Evolutionary design of neural architectures. Technical Report CS TR 95-01, Iowa State University, Department of Computer Science, 1995.

[2] Karthik Balakrishnan and Vasant Honavar. Properties of genetic representations of neural architectures. In *Proceedings of the World Congress on Neural Networks (WCNN '95)*, pages 807–813, 1995.

[3] Douglas Blank, Lisa Meeden, and Deepak Kumar. Python robotics: An environment for exploring robotics beyond legos. In *ACM Special Interest Group: Computer Science Education Conference (SIGCSE)*, 2003.

[4] Dave Cliff, Inman Harvey, and Phil Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2:73–110, 1993.

[5] Steven Alex Harp, Tariq Samad, and Aloke Guha. Towards the genetic synthesis of neural networks. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 360–369, 1989.

[6] Stefano Nolfi and Dario Floreano. Learning and evolution. *Autonomous Robots*, 7(1):89–113, 1999.