# Comparing Adaptivity of Ants using NEAT and rtNEAT

Teo Gelles & Mario Sanchez
Swarthmore College
Class of 2016

**Abstract**

*Although individual ants have an extremely basic intelligence, and are completely incapable of surviving on their own, colonies of ants can develop remarkably sophisticated and biologically successful behavior. This paper discusses a set of experiments which attempt to simulate one of these behaviors, namely the ability of ants to place pheromones as a way of communication. These experiments involved a variety of different environments, and tested two varieties of the genetic algorithm NEAT: the standard offline version, and its online counterpart rtNEAT. Since the experimental environment did not seem to offer any benefit to continuous learning, we had expected NEAT and rtNEAT to have roughly similar learning curves. However, our results directly contradict this hypothesis, showing much more succesful learning with rtNEAT than with standard NEAT.*

## I. Introduction

### A. Motivation

One of the many goals of the Artificial Intelligence community is the development of realistic simulations for behavior patterns of organisms, a field known as Artificial Life. The general motivation behind this field is that by creating these simulations, we will be more able to understand and predict the real world, the potential applications of which are theoretically unbounded and have recently started to manifest in such fields as industrial design, security, telecommunications, and data mining[5]. As can be expected, the most high-level goal of being able to simulate all forms of life and all of its behaviors is far out of the reach of any one experiment, so researchers focus on more basic, simplified models to create and study.

In this experiment, we simulate some of the actions of a standard colony of ants, using two slightly different learning approaches. Our choice of ants as an organism to model is due to the currently-accepted belief that ants exhibit fairly complex behavior as a group while having only simple individual capabilities[3]. In particular, although individual ants have only the most rudimentary sense of direction and a mere few seconds of memory, they are as collective groups able to consistently travel significant distances in order to find food, and return it to the nest. Their main tool in performing this task is through their evolutionary development of pheromones, scents which ants can drop that provide information regarding the path they are travelling. Our simulation attempts, in various ways, to develop this behavior in digital ants, using a few different machine learning techniques.

In our simulations, we use two varieties of a machine-learning paradigm called NeuroEvolution for Augmenting Topologies (NEAT). NEAT is a genetic algorithm that uses a cognitive model called a neural network to emulate the brain activities of organisms. As a genetic algorithm, NEAT maintains a parameter for every member of the population called fitness, which is simply a measure of how well the individual is performing. Normally, there is a set amount of time that a "generation" of organisms is allowed to live before being replaced, at which point NEAT selects individuals with the highest fitness as those most likely to pass on their own neural network to the next generation. This system is known as offline learning. However, there is also a variant of genetic algorithms in general, and NEAT itself, which do not replace an entire generation at once but individual organisms continuously as long as the experiment runs, which is known as real-time, or online learning. This type of NEAT is called real-time NEAT (rtNEAT). Our experiment tests how well a population of ants learns to use their pheromones to navigate a digital world using the standard NEAT vs. rtNEAT.

In some ways, the series of experiments described in this paper can be seen as a continuation of the authors' previous work with simulations using rtNEAT. In that study[4], the authors implemented a simulation of small prey "organisms", simulated as dots, that were equipped with rtNEAT and attempted to learn to evade user-controlled predators, also simulated as dots. The results were compared to those of a standard genetic algorithm, one which did not make use of several trademark qualities of NEAT. That study found that the organisms controlled by rtNEAT learned much faster than those of a standard genetic algorithm. However, rtNEAT's learning was so rapid, likely due to the goals of the experiment being so simplified, that it would take only a few minutes for the organisms to learn perfect evasion. This left open the question of how well rtNEAT would be able to handle more complex tasks, and also how its performance compared to NEAT itself. The experiments of this paper aim to demonstrate whether NEAT and rtNEAT can be used to learn optimal behaviors in more complex environments, and to determine which, if either, is more suitable.

### B. Previous Research

Several papers have been published describing experiments involving NEAT and rtNEAT, many of them written by the creator of the algorithm himself, Kenneth O. Stanley. However, a to our knowledge a direct comparison of NEAT and rtNEAT has never been done, perhaps because the supposed applications of each algorithm are different enough so that testing them together was never a priority. According to Stanley [10], rtNEAT, and real-time learning in general, is optimal for cases where the objective has substantial variability, in which case the real-time algorithm has more opportunity to change relative to the offline version. However, the paper is much more unclear as to the *disadvantages* of rtNEAT, as in, cases where rtNEAT would be expected to perform *worse*. We believe that in a consistent environment like in our simulation, the performance of the two algorithms should be roughly similar.

Referring back to [10] again, in the experiments described in that paper rtNEAT was implemented along with a combat training simulating game called NERO. In NERO, participants create a set of challenges and obstacle courses for non-player controlled characters (NPCs) equipped with rtNEAT to learn on. After training, the NPCs were tested against a player-controlled team, and success was gauged holistically. We note that in these experiments, rtNEAT was *not* tested against another learning algorithm, unlike in our experiments. Furthermore, we record our data quantitatively, and have more definitive goals as opposed to the unknown potential abilities of the NPCs.

NEAT itself has also been experimented with numerous times by Stanley, with one set of tests discussed in [9]. In those experiments, simulated robots "battled" each other within the following environment: The robots begin with a set amount of energy that decreases over time. Robots can pick up food which restores some energy. If two robots hit each other, the one with more energy wins, and get a boost to fitness. The strategy each robot used was decided by a variant of NEAT, with different core features removed. The results were strongly supportive of the conclusion that NEAT requires all of its principle features in order to operate optimally. However, like the previous experiments described with rtNEAT and NERO, the NEAT experiment did not test the fully featured NEAT against rtNEAT, however it did share the quantitative spirit of our paper. Our experiments are non-competitive, and in many ways test the ability of NEAT and rtNEAT to create cooperative strategies rather than the confrontational ones of Stanley's experiments. We are also not interested in the specific features of NEAT and their utility, but rather how the algorithm and its equivalent but real-time counterpart compare in a task that would intuitively seem to offer no advantage to either.

Of the many experiments summarized in [5] regarding artificial life, ours has a fair number of similarities with some and is widely distinct from the others. From a motivational standpoint, all of those experiments had a direct, industry-level application in mind with their development. Many of them tested brand-new algorithms against previous standards, with the intention of proving the superior performance of their design over those of their rivals'. Interestingly, some of the designs do seem to be inspired by ant colony behavior, such as the Ant-Miner data mining algorithm created by Parpinelli et al. [6]. But once again, no experiments seem to have been interested in directly comparing an algorithm vs. a real-time version of the algorithm. Also, we have had no hand in designing NEAT, so we have no motivation to prove its superiority in any respect.

## II. Experiment Description

### A. Experimental Background

*1) Neural Networks:* Artificial neural networks are one of the main backbones of modern adaptive learning. As the name may suggest, neural networks are a mathematical model that is based off the neurons found in human brains. Like biological neurons, artificial neural networks are composed of small units, neurons, that are interconnected. Through this interconnection, artificial neural networks are able to approximate any mathematical function.

Mathematically, each neuron is a simple single-valued function. While there are many possibilities for what that function could be, throughout our experiments we will be only using the sigmoid function. This function is defined as

$$\text{Sigmoid}(x) = S(x) = \frac{1}{1+e^{-x}}.$$

A plot of this function can be seen in Figure 1

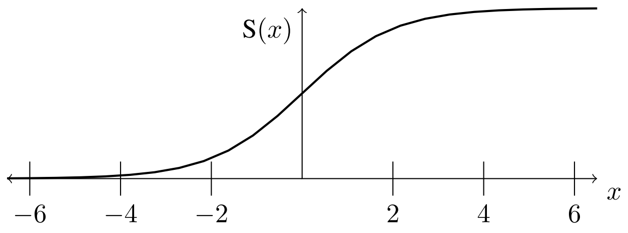

Figure 1. Graph of the Sigmoid Function

While a single neuron does not give any significant computational power, their connection does. Neurons are connected through weighted links between their inputs. That is, the output of one neuron is weighted and then fed into the input of another neuron. For a neuron that has multiple neurons feeding into it, the input is the weighted summation of the outputs of the previous neurons and the corresponding weights on the links. A diagram showing this interconnection is shown in Figure 2.

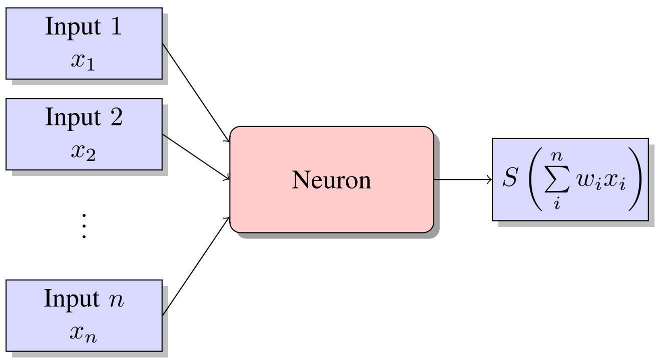

Figure 2. Connections of Neurons

While in general these neurons are allowed to connect in any way, we only concern ourselves with layered recurrent neural networks. Our neurons our found in three layers. The first layer is the layer of input neurons. These neurons receive their input from an external source (the input from the program). These input neurons connect to the next layer. This layer is called the hidden layer and it contains the hidden neurons. These neurons are hidden because they do not directly come into contact with the program. They simply serve as an intermediary between layers. The last layer is the layer of output neurons. In this layer, the neurons receive input from the hidden neurons and the output is sent back to the program. In addition to this structure, our neurons are recurrent. That is, they are allowed to connect to themselves and to previous layers. A diagram of this structure is shown in Figure 3

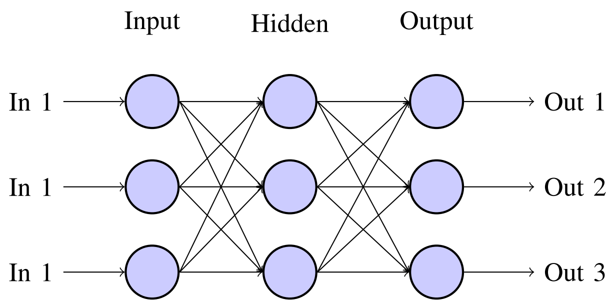

Figure 3. Structure of the Network

Neural networks serve as powerful approximation tools, since by changing the weights, the neural network can change the function that they apply to the inputs. As such, many algorithms to change the weights exist. The two most popular methods are back-propagation and neuroevolution. While back-propagation is extremely useful, it is not the focus of the paper. For an introduction into back-propagation refer to any text on neural networks. The second method, neuroevolution, is described in the next section.

*2) NeuroEvolution for Augmenting Topologies:* Neuroevolution is one of the most common learning methods for neural networks. As the name suggests, this method is modelled after Darwinian Evolution. Initially, there is a population of neural networks all with random weights. Each neural network is then tested against some metric to give it a score - the fitness. This fitness depends on the neural network's ability to perform a task or to approximate some function. Then, a new population is formed by selecting the fittest individuals of the previous population and using them to create a new population. This new population is created by recombining the various individuals or by small mutations to the individuals. This new population is then subject to the same test. By repeating this process, individuals become fitter over time and the corresponding networks become more able to approximate the desired function. A diagram of this standard evolutionary algorithm is shown in Figure 4.

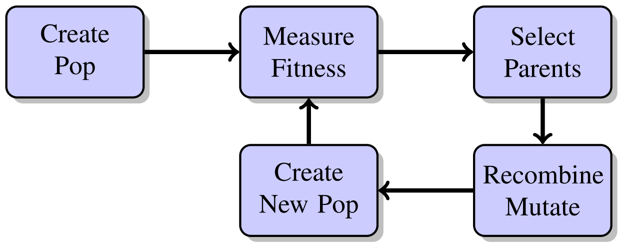

Figure 4. Standard Genetic Algorithm

However, this standard algorithm has many faults when it comes to learning difficult tasks. For one, the topology of the networks, that is the number of hidden layers and the corresponding links between the neurons, remains fixed throughout evolution. This means that the complexity of the behavior remains constant. In addition, often the recombination of different networks takes two good networks and creates a network that is worse than both of its parents. For these reasons, researches searched for a new way to conduct neuroevolution.

In 2002, Kenneth O. Stanley and Risto Miikkulainen developed a new neuroevolution algorithm known as NeuroEvolution for Augmenting Topologies (NEAT) [7]. This new algorithm sought to fix the many flaws that are found in traditional neuroevolution. As such, NEAT is based off the following ideas: Innovation numbers, speciation, and complexification.

The most pressing issue with the old algorithm was its inability to change the topology of the networks as time progressed. The idea of allowing networks to change is not a new idea. First, a mutation was added to the typical set of evolutionary operations that allows the addition of new networks or new links where none existed before. Through this, a network is able to grow in its number of hidden neurons throughout the evolution. However, there is a difficulty with making this work. Once there are networks with differing topologies, it is no longer a simple manner to take two networks and combine them. The solution to this was given by Kenneth in the form of innovation numbers. At the initialization of the evolution, every network has the same topology. As evolution progresses, they each increase in the number of neurons and links. However, within the genetic code (the genotype), an innovation number is used to mark the time at which the new neuron or link was added. By recording the information regarding these innovation numbers, the algorithm is able to combine networks with different topologies.

As stated before, another of the issues regarding standard neuroevolution methods is that networks which have adapted to perform in different situations are allowed to recombine to create poor networks. To solve this, a technique known as speciation is used. The name is self-explanatory. The individuals in the population are split in to different groups called species. Members from different species are not allowed to combine with members from a different species. In order to test whether or not two members are from a different species, their topology and weights are compared. If either of them are too different, then they will not be allowed to mate. By doing this, the issue is resolved. In addition to this, there is a mechanism known as species protection that protects new species. This is done through fitness sharing. That is, each species is allocated a total fitness that is then shared within the species. This ensures that new members from a new species are able to survive long enough to grow to their potential.

The final main idea behind NEAT is complexification. As stated before, the initial population is made of small networks with a relatively small number of neurons. However, as time progresses, the networks become more complicated and grow in the size of hidden neurons. This is the idea of complexification. It is useful because the search space is small at the beginning. After optimizing in the small number of dimensions, more dimensions are added to the search space. Thus, the amount of the large search space that needs to be considered drops dramatically. This leads to better behavior more quickly.

*3) Real Time NeuroEvolution for Augmenting Topologies:* When NEAT was first created, it was solely an offline learning algorithm. However, in 2006, Kenneth introduced a real-time version of his NEAT algorithm. This algorithm is aptly known as rt-NEAT.

The difference between offline and realtime NEAT is in the way that a new population is created. In realtime NEAT, when the creating new population step occurs, instead of replacing the entire population, only a few individuals are replaced. Then, a few new individuals are created an added to the population. In this way, the population is under constant change, yet it never changes all at once.

In order to ensure that the new individuals will have time to survive to gain fitness, there is age-protection. That is, before a certain number of time steps, a new individual is protected from being selected by the algorithm to die. Thus, by doing this, the population is constantly getting better without being replaced at once.

In our previous project, rt-NEAT gave many benefits over offline genetic algorithms. The first benefit is that it gives a feeling of continuity to the evolution. In many applications, such as video games, this quality is an important one. In addition, in scenarios where many individuals are co-evolving, having continuity of evolution is less disruptive and sudden than typical genetic algorithms.

*4) Ant Biology:* Biological ants have been studied widely for their ability to achieve complex tasks when grouped together. This is made more amazing by the fact that the ants have a very basic intelligence. Thus, most of their intelligence comes from their communication - the so called swarm intelligence [3].

Ant colonies consist of many different specialized roles. Two of these roles, the scout and the scavenger, are the main focus of our research. The scout ants are the first to leave the colony in search of food. Once outside the nest, the scout ants move in more or less a random manner. Upon reaching food or an enemy, the ant will return to its base while placing a scent trail, a pheromone trail, on the ground. If the scout ant has found food, the pheromone trail will lead another ants towards the same food source. Similarly, if the scout ant has found some danger, the warning pheromone will warn the other ants of the colony. Thus, as the name states, scout ants are the ants that find the food.

Once food has been found, the job of the scavengers begins. The scavengers will follow the pheromone trail placed by the scouts. Upon finding food, they will follow the trail back to base while placing their own pheromone trail down. As more food is gathered, the pheromone trail will increase in strength. In turn, this will lead more ants to the food. As the food source runs dry, the ants will stop leaving their pheromone trail. Therefore, due to evaporation of the pheromones, the trail will vanish when the food runs out.

This combination of roles allows the ant colony to efficiently find and scavenge food from their surroundings. In addition to finding food, these pheromones also allow the ant colony to defend themselves from predators. This communication is what gives ants the evolutionary advantage they need to survive.

## *B. Experimental Design*

In order to more comprehensively test the capabilities of NEAT and rtNEAT, our tests consist of two different sorts of experiments. The second type is a much more simplified version of the first, and serves as a sort of control group to see whether a more narrow set of behaviors than those necessary for the first experiment are learnable.

*1) General Simulation Properties:* All experiments, regardless of their type, take place on a 1880x960 grid, each entry of which corresponds to a pixel on screen. Ants on the grid take up a single pixel, but are displayed graphically with a circle of radius 5 pixels. The ants are spawned at random points within a 200x200 rectangle, their "home", near the bottom-center of the screen. Along with the ants on the grid are five food sources, each located about 650 pixels away from the center of the home rectangle, at intervals of $37°$ starting from $15°$ above the $x$-axis, which works out to the distribution of the food being symmetric around the home. The edges of the screen are impassable walls, so all distances and appearances are standard Euclidean. Other than those walls, however, no other contact is recorded, so ants can overlap each other as well as the food sources indiscriminately. Along with these objects on the grid is the ant pheromone, which can take on four possible "colors": blue, red, green, and black. While different color pheromone can be placed on each entry of the grid, only the dominant color is displayed or matters for purposes of the simulation. Ants are allowed to move, place pheromone, and pick up food once per every time step. In total, each run of the simulation is 180,000 time steps. If rtNEAT is used, five ants are replaced every 100 time steps. If regular NEAT is used, the entire population is replaced every 3000 time steps. In order to prevent the ants from entering a movement trap, where due to the discrete nature of the grid they would oscillate between two spaces that each specify the other direction as optimal, the ants are allowed to move in a continuous fashion, and only when they place pheromone is their position interpreted as a discrete location.

Each ant has two fitness values associated with it. One is altered every time step, by the following methodology. If the ant does not have food, it is rewarded dependent on its distance to the nearest food source, by the following formula:

$$\text{fitness}_{\text{towards food}} = 10e^{-\text{distance}/200} \tag{1}$$

This results in between .5 and 10 fitness points for an ant every time step. If an ant without food touches a food source, it picks up food automatically and receives a substantial boost to fitness:

$$\text{fitness}_{\text{pick up food}} = 10,000 \tag{2}$$

Once an ant obtains food, it is no longer rewarded for its distance towards the nearest food source. Instead, it is rewarded at an even greater scale dependent on its distance towards the home base:

$$\text{fitness}_{\text{towards home}} = 200e^{-\text{distance}/200} \tag{3}$$

This results in between 10 and 200 fitness points for an ant every time step. Finally, if an ant with food touches the home base, its food is taken from it and an enormous boost to fitness is received:

$$\text{fitness}_{\text{returning food}} = 200,000 \tag{4}$$

However, although this fitness variable is recorded and updated, the actual fitness used to determine an ant's likelihood to be chosen to reproduce is the above fitness averaged over the ant's lifespan

$$\text{actual fitness} = \frac{\text{fitness}}{\text{number of time steps alive}} \tag{5}$$

This gives ants a motivation for moving and finding food, rather than the otherwise acceptable strategy of standing still and accumulating fitness points. In practice, averaging the fitness basically caps it at around 200 points, allowing newer ants the opportunity to match, and eventually overtake, the fitness of the older if they display similar or better behavior.

*2) Experiment Type 1: Full Learning:* The first type of experiment run is the more comprehensive. In these experiments, two different types of ants are used, each with the ability to place pheromone. One type represents the scavenger ants as described earlier. Each of these ants has a neural network which consists of 49 input neurons and 7 output neurons. The input neurons take the amount of the dominant pheromone at each grid entry within the 3x3 area surrounding the ant, as well as 0's for the non-dominant pheromone so that the ants can tell which pheromone is dominant. This accounts for 36 of the input neurons. Another 9 input neurons are used to identify the type of each grid location surrounding the ant, namely whether the spot is a food, home, or wall location. This represents the ant's very basic sense of direction. Three input neurons are given random inputs, so that ants do not behave entirely deterministically when travelling across the largely empty grid, and have a natural impulse to experiment and explore. Finally, a single input neuron is a boolean determined by whether or not the ant has food. Of the output neurons, two are used to determine how far the ant will move horizontally and how far the ant will move vertically, both of which can go as far as 1 grid space. The remaining five output neurons represent how much the ant wants to place of each of the four pheromone colors, plus the possibility of placing no pheromone. To decide, the maximum of these five neurons is taken, and the corresponding pheromone is placed. For each run of the simulation, 200 of these ants are in use at any given time.

Along with the representations of scavenger ants, this experiment uses scout ants. Unlike the scavenger ants, which move based solely on the information in their immediate vicinity, scout ants have hard-coded within them the locations of the five food sources. Also unlike the scavenger ants' complicated neural networks, the scout ant neural networks consist of just two input and two output neurons. The two inputs to the network are the relative position of either the nearest food source (if they do not have food) or the home territory (if they do have food), and the two output neurons are, like the scavenger ants, how far to move horizontally and how far to move vertically. These ants place blue pheromone at every time step, so adding in information to the network to control pheromone usage is unnecessary.

Ants place pheromone in sets of 5 units at a time. But to prevent the environment from being entirely cluttered with old pheromone, pheromone is evaporated at a rate of $4$ units every $400$ time steps, which leaves about enough time for an ant to get to a food source and back on the same trail at a leisurely pace.

Typically, this experiment will start out looking like the graphic shown in Figure 5. In that figure, the five green squares represent food sources. The home base is the uncolored rectangle near the bottom center of the graphic.

After the scout ants have learned to find and return food, the experiment begins to look like the graphic in Figure 6.

*3) Experiment Type 2: Hardcoded Paths:* The first type of experiment requires the scavenger ants learning both how to

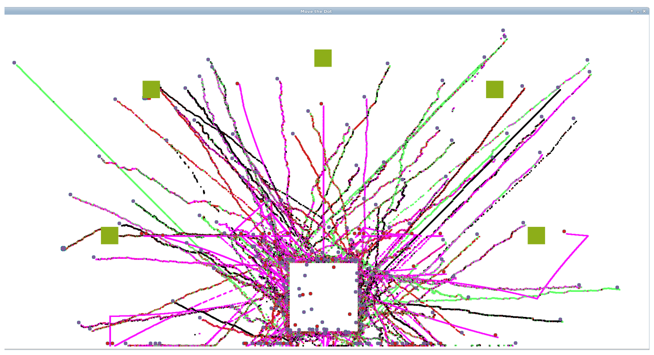

Figure 5. Experiment 1 Start

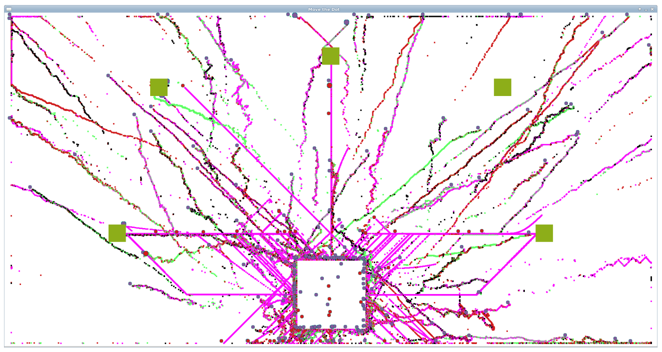

Figure 6. Experiment 1 Middle

follow and how to place pheromone. To be comprehensive, this second type of experiment tests only whether scavenger ants can learn to follow pheromone. To accomplish this, scout ants are removed, and the scavenger ants are not given the ability to place pheromone. Instead, five pheromone trails have been hardcoded into the simulation, each pointing from the center of the home to one of the food sources. Fitness is still calculated regularly, and the inputs to the scavenger ant neural networks remain the same, but now the five outputs from the previous experiment that had been used to determine which pheromone the ants should place have been removed. Instead, only the two outputs used to determine direction are used. The experiment quickly begins to look like the graphic of Figure 7.

ive

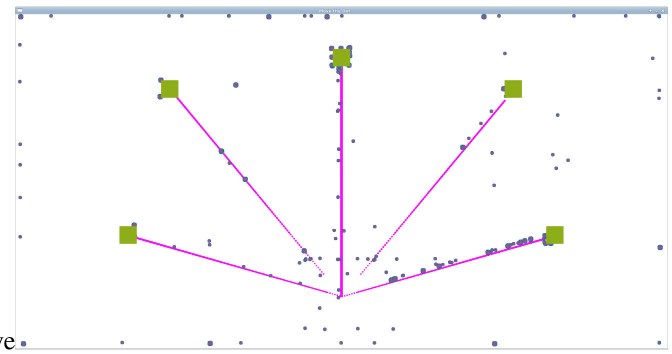

Figure 7. Experiment 2

## C. *Hypothesis*

For both our experiments, since the environment of the simulation does not change significantly over time, the learning technique need not be continuous, so we expect that the performance of NEAT and rtNEAT to be roughly the same.

## III. Results

To see all the figures with our results, refer to pages 9 (rtNEAT results) and 10 (NEAT results) of this paper. We had twelve different variants of our experiments, each of which had four different variables that we recorded: the average fitness of the ants, the maximum fitness of the ants, the amount of food the ants obtained, and the amount of food the ants returned. The twelve variants of our experiment consist of the following:

- rtNEAT Experiments
  - Experiments of Type 1
    1) 0 Scouts, 4 Starting Hidden Neurons
    2) 0 Scouts, 10 Starting Hidden Neurons
    3) 50 Scouts, 4 Starting Hidden Neurons
    4) 50 Scouts, 10 Starting Hidden Neurons
  - Experiments of Type 2
    5) 2 Hidden Neurons
    6) 4 Hidden Neurons
- NEAT Experiments
  - Experiments of Type 1
    7) 0 Scouts, 4 Starting Hidden Neurons
    8) 0 Scouts, 10 Starting Hidden Neurons
    9) 50 Scouts, 4 Starting Hidden Neurons
    10) 50 Scouts, 10 Starting Hidden Neurons
  - Experiments of Type 2
    11) 2 Hidden Neurons
    12) 4 Hidden Neurons

We graph each type of experiment for each brain on the same plot for each variable, and plotted both the food obtained and food returned on the same graph, yielding 2 different neural networks $\times$ 2 different experiment types $\times$ 3 different plotted variables $=$ 12 different graphs. Videos of these experiments can also be found at https://www.youtube.com/channel/UC6BHeoJBVHBpEOYsc6VqkVw.

As our results show, ants exhibited almost no learning during the first type of experiment, with an almost linear rate of food obtaining, implying basically random encounters with the food squares, and very little food returning, implying no ability of consistent path creating or following. In these types of experiments, rtNEAT and NEAT performed very similarly, as per our expectations.

However, the results of the second type of experiment are very different. Ants controlled by rtNEAT displayed significant learning, with an early exponential increase in the amount of food obtained closely followed by food returned, implying that ants were able to follow the hardcoded paths to the food and back. We also see the benefit of starting with 2 hidden neurons as opposed to 4, with the latter beating the former by a wide margin with respect to all three measurements. But with standard NEAT, no such learning took place, with a linear increase in the amount of food obtained and minimal amount of food returned, very similar to performance with the first type of experiment. This implies, against our hypothesis, that rtNEAT is more suitable for learning the task than regular NEAT.

## IV. Conclusion

### A. Discussion

Originally, we believed that the difficulty of the task would be so great that rtNEAT and NEAT would not perform too well. In addition, we thought that rtNEAT and NEAT would both perform equally. That is, online learning did not seem to give any advantages in this domain. However, this was not the case.

We have found that for the simpler task, where the path was hardcoded, rtNEAT outperformed NEAT by a large margin. There are many possible reasons for this difference. The main one is that NEAT has many discontinuities in its evolution. At one time there are many ants out, then at another, they all vanish and get replaced by a new population. This might add the difficulty of the task. Whereas, when using rtNEAT the evolution is continuous. This allows newborn ants to take advantage of the pheromone trails left by its predecessors.

When it comes to the difficult task, it is true that they both perform equally as badly. Thus, we have found the border at which the task becomes difficult. Learning to follow pheromones is easily achieved by rtNEAT agents. However, learning to place pheromone trails for other is the difficult task. This communication between ants is the difficult task.

### B. Future Work

We have many ideas for the improvement of the project. First, we believe that using a more powerful neuroevolutionary algorithm such as hyperNEAT or Evolving-Substrate hyper-NEAT will achieve higher results. These algorithms utilizes a network known as a Compositional Pattern Producing Network (CPPN), which is known for its high level of symmetry, to generate a symmetric neural network. This would be useful for two reasons. One, the high symmetry will be beneficial for this project since the behavior we seek is symmetrical. That is, when ants are in similar situations, they should perform similarly. Second, CPPN based neural networks allow for a large input size and output size. Due to our large number of inputs, these networks will learn better when the CPPNs evolve insteadimprove our results of the neural networks [12].

Secondly, currently, the ants only have a sight radius of one pixel around them. Increasing the size of this sight radius will allow our ants to see the trails more easily. It is possible that their sight radius is so small that they quickly lose track of the trails due to the randomness. Since they are completely lost without the trails, this leads to a lot of confused ants.

Thirdly, biological ants have a larger memory than our artificial ants. This might give them the ability to return to the nest more effectively. The problem is that we would need to use a new type of learning structure which has a longer memory than neural network. However, finding this should improve our results greatly.

### C. Acknowledgements

We would like to thank our professor Lisa Meeden for teaching us everything we know about adaptive robotics and artificial intelligence, Kenneth O. Stanley and Risto Miikulainen for their NEAT and rt-NEAT implementation, and our classmates for being supportive of our work.

## References

[1] Adrian Agogino, Kenneth Stanley, Risto Miikulainen, *Online Interactive Neuro-Evolution*. Neural Processing Letters, 1999.
[2] Mark A. Bedau, *Artificial Life*. Handbook of the Philosophy of Science, Volume 3: Philosophy of Biology. Elsevier BV, 2007.
[3] Eric Bonabeau, Guy Theraulaz, Marco Dorigo, *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, 1st Edition, 1999.
[4] Teo Gelles, Mario Sanchez, *Comparing Approaches to Online NeuroEvolution*. Swarthmore College, 2014.
[5] Kyung-Joong Kim, Sung-Bae Cho, *A Comprehensive Overview of the Applications of Artificial Life*. Artificial Life 12. Massachusetts Institute of Technology, 2006.
[6] R.S. Parpinelli, H.S. Lopes, A.A. Freitas, *Data mining with an ant colony optimization algorithm*. IEEE Transactions on Evolutionary Computation, 2002.
[7] Kenneth O. Stanley, Risto Miikkulainen, *Evolving Neural Networks through Augmenting Topologies*. Evolutionary Computation. Massachusetts Institute of Technology, 2002.
[8] Kenneth O. Stanley, Risto Miikkulainen, *A Taxonomy for Artificial Embryogeny*. Artificial Life Journal, 2003.
[9] Kenneth O. Stanley, Risto Miikkulainen, *Competitive Coevolution through Evolutionary Complexification*. Journal of Artificial Intelligence Research 21. AI Access Foundation and Morgan Kaufmann Publishers, 2004.
[10] Kenneth O. Stanley, Bobby D. Bryant, Risto Miikulainen, *Evolving Neural Network Agents in the NERO Video Game*. Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games, 2005.
[11] Kenneth O. Stanley, Bobby D. Bryant, Igor Karpov, Risto Miikkulainen, *Real-Time Evolution of Neural Networks in the NERO Video Game*. The Proceedings of the Twenty-First National Conference on Artificial Intelligence, 2006.
[12] Kenneth O. Stanley, Risi, Sebastian, Lehman, John, *Evolving the Placement and Density of Neurons in the HyperNEAT Substrate*. Proceedings of the Genetic and Evolutionary Computation Conference, 2012

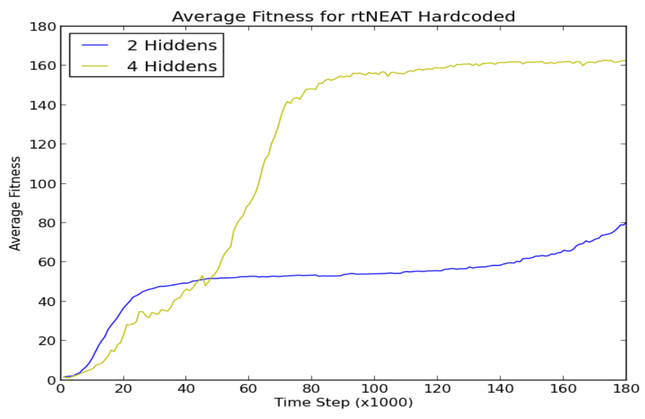

Figure 8. Average Fitness for rtNEAT Hardcoded

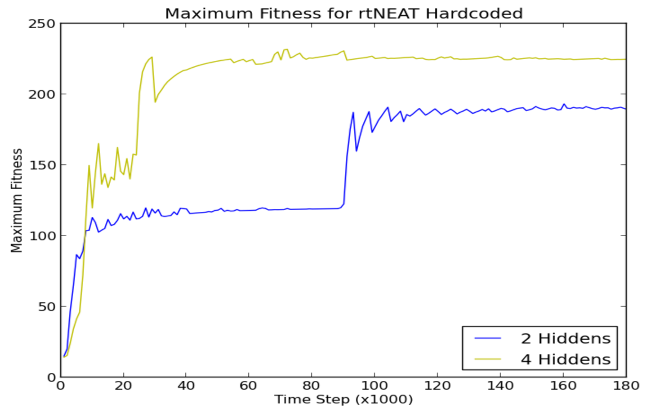

Figure 9. Maximum Fitness for rtNEAT Hardcoded

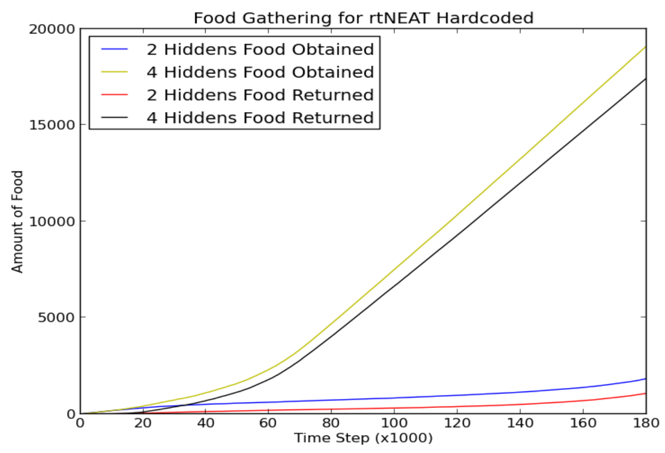

Figure 10. Food Gathering for rtNEAT Hardcoded

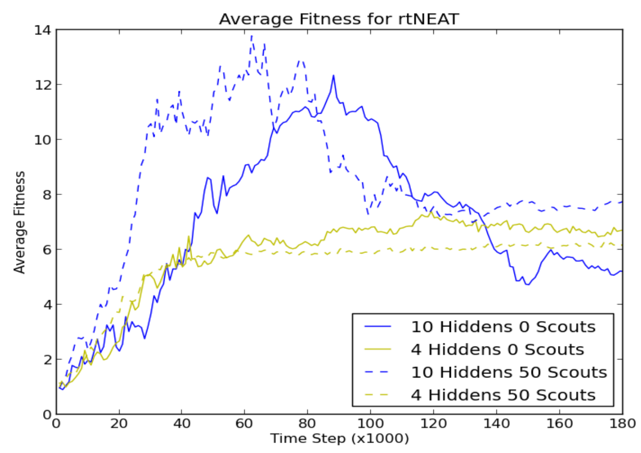

Figure 11. Average Fitness for rtNEAT

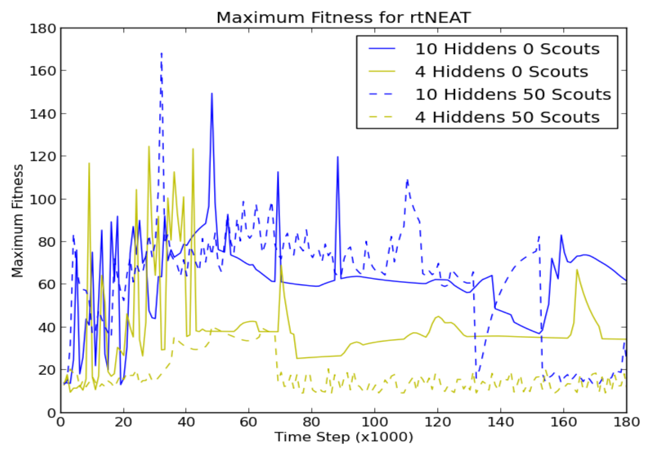

Figure 12. Maximum Fitness for rtNEAT

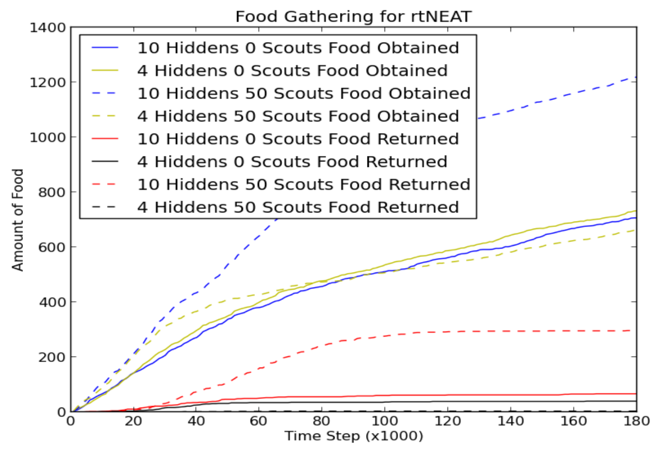

Figure 13. Food Gathering for rtNEAT

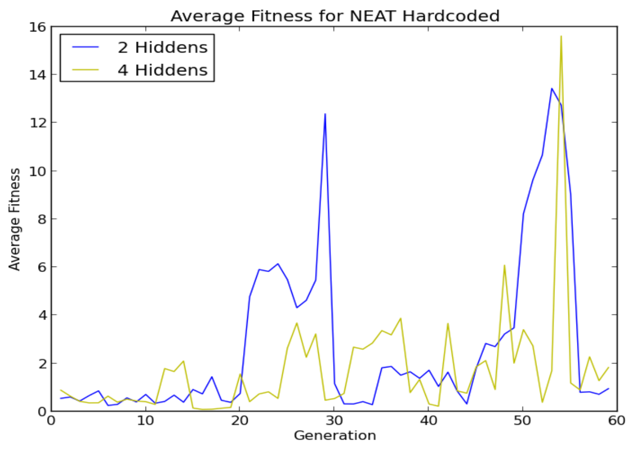

Figure 14. Average Fitness for NEAT Hardcoded

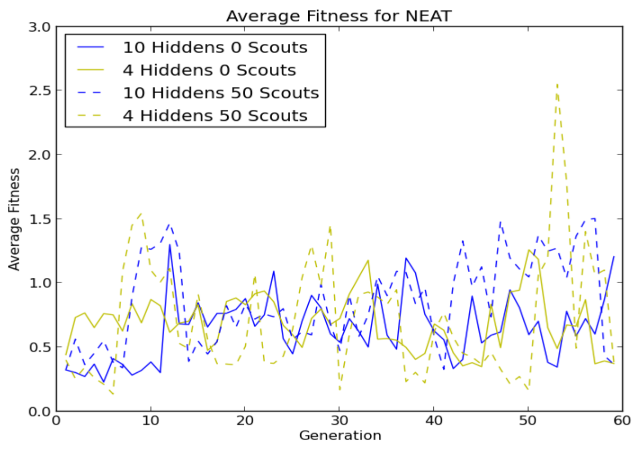

Figure 17. Average Fitness for NEAT

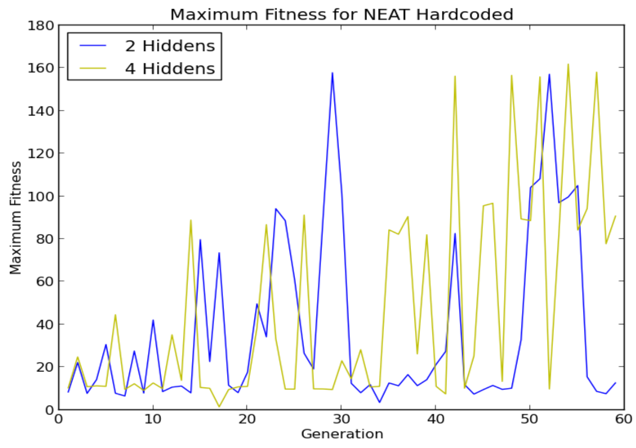

Figure 15. Maximum Fitness for NEAT Hardcoded

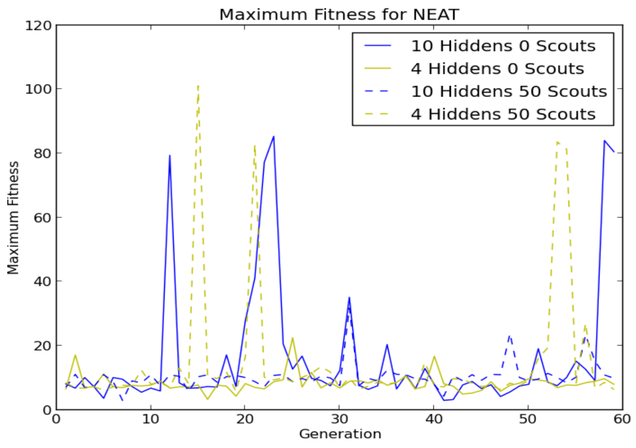

Figure 18. Maximum Fitness for NEAT

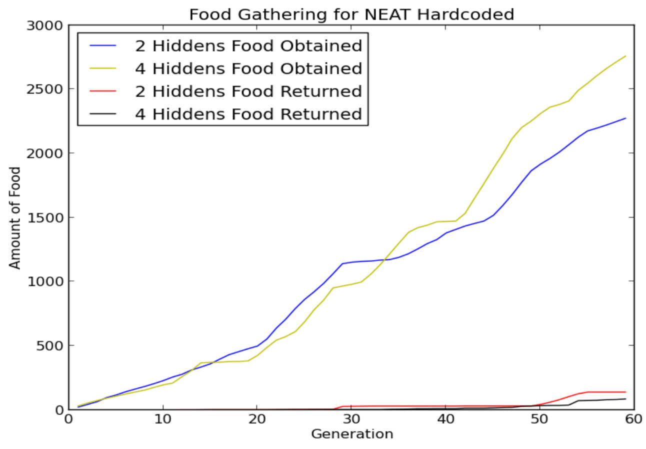

Figure 16. Food Gathering for NEAT Hardcoded

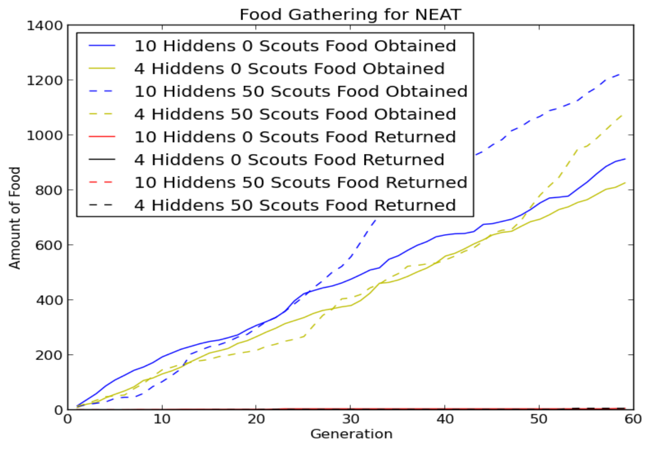

Figure 19. Food Gathering for NEAT