Bachelor of Science in Computer Science
October 2017

Competitive coevolution for micromanagement in StarCraft: Brood War

Filip Bloom

Faculty of Computing
Blekinge Institute of Technology
SE - 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering. The thesis is equivalent to 10 weeks of full-time studies.

Contact Information
Author:
Filip Bloom
e-mail: filip.bloom@hotmail.se

University advisor:
Peng Cong
Department of Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE - 371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Interest in and research on neural networks and their capacity for finding solutions to nonlinear problems has increased greatly in recent years.

Objectives. This thesis attempts to compare competitive coevolution to traditional neuroevolution in the game StarCraft: Brood War.

Methods. Implementing and evolving AI-controlled players for the game StarCraft and evaluating their performance.

Results. Fitness values and win rates against the default StarCraft AI and between the networks were gathered.

Conclusions. The neural networks failed to improve under the given circumstances. The best networks performed on par with the default StarCraft AI.

Keywords: neural networks, NEAT, StarCraft

List of Figures

2.1 Simple Neural Network 4
2.2 Neural Network with one hidden layer 5
4.1 Neural Networks vs Default AI 13
4.2 Fitness Values 14
4.3 Traditional vs Coevolved 15
4.4 Traditional vs Traditional 15
4.5 Coevolved vs Coevolved 16

Contents

Abstract . . . . . **iii**

List of Figures . . . . . **iv**

1. Introduction **1**
1.1. Background 1
1.2. Related work 1
1.3. Aims 2
1.4. Objectives 2
1.5. Research questions 3

2. Neural Networks **4**
2.1. Neural networks 4
2.2. Neuroevolution 5
2.3. NEAT 6
2.4. Competitive coevolution 6

3. Method **8**
3.1. Motivation 8
3.2. StarCraft: Broodwar and BWAPI 8
3.3. Overview 9
3.4. Experiment 10
3.4.1. Implementation 10
3.4.2. Execution 12

4. Results **13**

5. Analysis and Discussion **17**
5.1. Evaluation against default StarCraft AI 17
5.2. Evaluation between neural networks 17
5.3. Research questions 17

6. Conclusion and Future Work **18**

References **19**

1 Introduction

1.1 Background

Interest in and research on neural networks and their capacity for finding solutions to nonlinear problems has increased greatly in recent years. While much has been discovered, the many different variations on neural networks and their uses leaves many research gaps. Modern computer games, in particular, is a field where neural networks have found several uses [1], yet is only partially explored.

An open problem regarding neural networks in games is the efficiency of competitive coevolution, i.e. the fitness of a network is evaluated by letting it play against another agent from the same population. This stands in contrast to what can be described as traditional neuroevolution, where the network is evaluated by playing against a preprogrammed AI agent. Competitive coevolution can in theory lead to open-ended evolution and thus improve indefinitely, although such behaviour has never been achieved. Risi and Togelius [1] suggest that more room for complexity, as exists in modern games, might allow for more sophisticated methods to be evolved.

In this thesis we intend to investigate the efficiency of competitive coevolution compared to traditional neuroevolution. We will accomplish this by evolving two AI-controlled players for the game StarCraft: Brood War and then comparing the resulting players. The reason for choosing StarCraft is twofold: first, it is a real-time strategy game where each unit can make any one of several decisions multiple times per second. This makes for an extremely large possibility space which neural networks are generally considered good at exploring. Second, there is a fair amount of both tools and precedence for developing AI for StarCraft, which should aid in the implementation compared to other games.

1.2 Related work

Both Lubberts et al and Runarsson et al have investigated competitive coevolution with regards to the board game Go [2], [3]. As shown by Runarsson et al, coevolution has the potential of achieving a higher level of play than self-play learning on small Go boards given the right configuration. However, standard setups of the evolutionary algorithms resulted in the self-play algorithm both learning faster and achieving a higher level of play [3].

Zhen and Watson evaluated NEAT and its real-time variant, rtNEAT, in controlling StarCraft units in matchups between different types of units. The matchups were 12 units vs. 12 units where each individual unit was controlled by its own neural network. The study showed the viability of using neuroevolution for unit micromanagement [4].

Shantia et al compared two different reinforcement learning methods in 3v3 and 6v6 matchups of Terran Marines. The networks in the 6v6 scenario proved to learn much slower due to the vastly increased number of possible game states and thus much larger problem space [5].

1.3 Aims

The aims of this thesis are:

- Explore and understand the difference between competitive coevolution and traditional benchmarked evolution when evolving AI-controlled players for games.
- Gain greater insight into the development and implementation of neural networks to play games.
- Explore the benefits of and problems with competitive coevolution in neural networks.

1.4 Objectives

In order to evaluate different learning approaches in StarCraft: Brood War, we need to do the following:

- Implement a player for StarCraft: Brood War that can be controlled by a neural network.
- Create an in-game scenario that allows the neural network to fight the default game AI.
- Create a version of the same scenario where two neural networks fight each other.
- Let the two approaches evolve over a number of generations.
- Record the fitness values in order to track progress of the two approaches, and compare the results.
- Evaluate the two different neural networks by letting them fight both the default AI and each other.

1.5 Research question

Is there a difference in evolution speed between competitive coevolution and traditional neuroevolution when evolving AI-controlled players for StarCraft: Broodwar? If so, what is the difference?

Null hypothesis: The evolution speed of competitive coevolution and traditional neuroevolution is equal.

Hypothesis 1: Competitive coevolution has a higher evolution speed than traditional neuroevolution.

Hypothesis 2: Traditional neuroevolution has a higher evolution speed than competitive coevolution.

2 Neural Networks

2.1 Neural networks

Artificial neural networks (ANN) as a computational model were first presented by McCulloch and Pitts in 1943. The model was loosely modeled after axons and synapses, and simulates the "all-or-none" process of neurons through threshold logic in order to propagate signals through the system. In other words, if the sum of a neuron's inputs is above a given threshold, the neuron's output is 1. If the sum of the inputs is below the threshold, the output is 0 [6].

Author Kevin Gurney gives the following definition for neural networks, including emphasis on key terms:

> "A neural network is an interconnected assembly of simple processing elements, *units* or *nodes*, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or *weights*, obtained by a process of adaptation to, or *learning* from, a set of training patterns." [7]

The weight values of the connections between neurons determine the resulting data, and these weights can be changed iteratively in order to "train" the network to give a desired result from a given input. *Figure 2.1* illustrates a neural network where the circles represent nodes and the arrows represent the weighted connections.

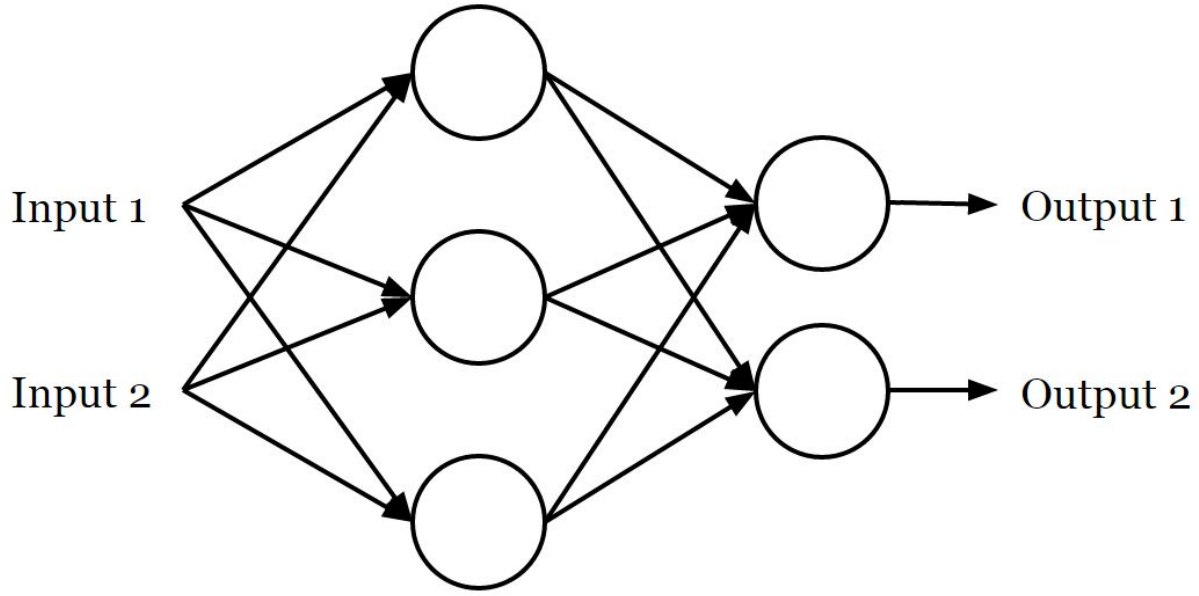

Figure 2.1: *Simple neural network*

More complexity can be introduced to the network by adding one or more layers of *hidden* nodes between the input and the output, as seen in *figure 2.2*. The extra connections between nodes allow for more advanced behavior and a larger number of viable solutions.

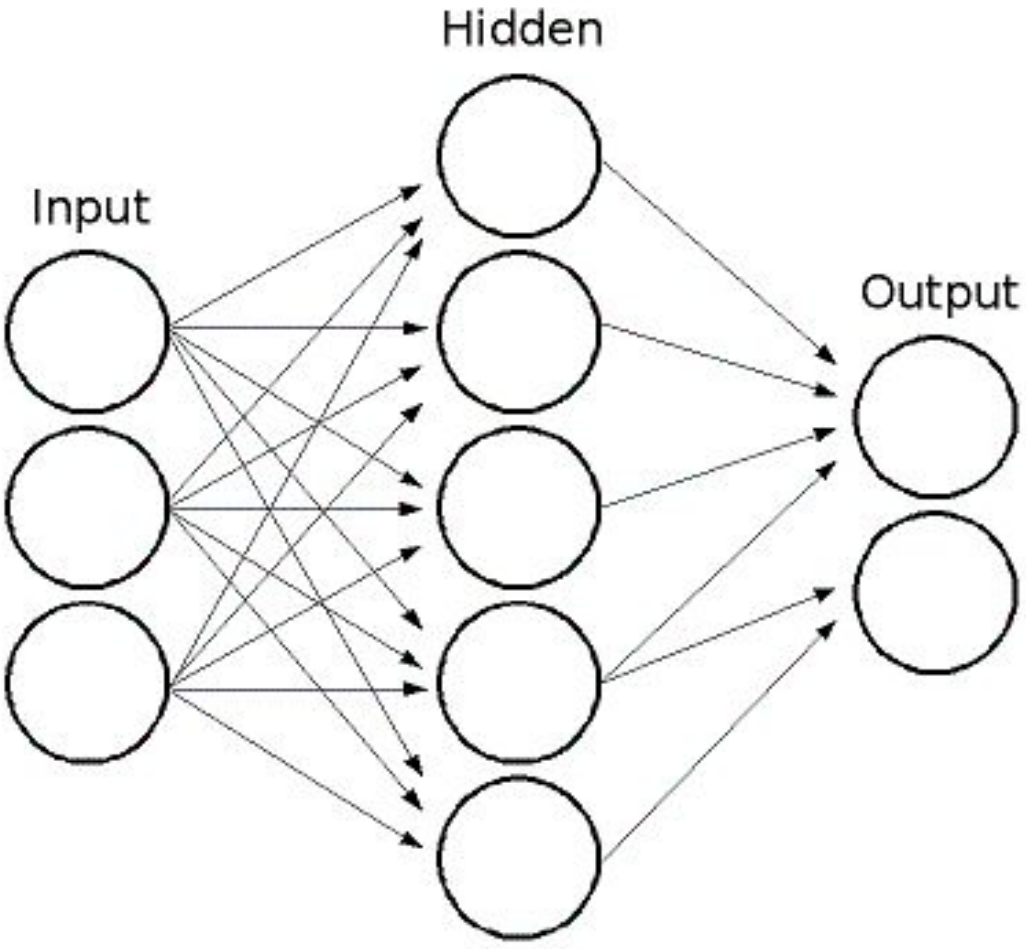

Figure 2.2: *Neural network with one hidden layer*

2.2 Neuroevolution

Genetic algorithms, like neural networks, are inspired by natural processes in their design. All living organisms have genetic material containing information about their features and attributes, which they carry on to the next generation. The offspring carries genetic features in chromosomes from both of its parents, the combination of which might manifest in new and possibly beneficial ways. Genes more suited to the surroundings make the organism more likely to survive and reproduce, which allows it to transfer the beneficial genes to the next generation in a process known as natural selection.

Genetic algorithms encode their features in a way analogous to the natural process, where

each solution is represented as a chromosome. Each generation, the viability of each solution is evaluated by an appropriately defined *fitness function* and represented as a single *fitness value*. The best performing solutions are allowed to reproduce by combining their chromosomes, while the worst performing solutions are discarded from the population. This allows for an environment where only the best solutions "survive", and the viability of the solutions increase with each generation.

Neuroevolution is a method for training neural networks which makes use of genetic algorithms to improve itself.

Neuroevolution refers to the use of genetic algorithms to solve the following tasks when developing neural networks [8]:

- learning of weights when training the network,
- determining the network architecture, i.e. the number of nodes and which links should exist between nodes,
- simultaneously determining the weights and the structure of the network.

In the past couple of years neural networks have been used to demonstrate means of developing solutions to a number of problems, including agents capable of learning how to play computer games [1].

2.3 NEAT

One common variation of a neural network implementation is NeuroEvolution of Augmenting Topologies (NEAT). NEAT is a method of training neural networks which simultaneously evolves both the weights and the structure of the network, thus both optimizing and complexifying solutions. NEAT also allows for speciation of the neural networks to make the process more efficient. Changing the structure of a neural network usually initially reduces its fitness, since the weights are not calibrated for the new structure. By grouping similar networks into species which primarily compete within their own niche, innovation is protected and new topologies get a chance to improve [9].

The original NEAT implementation in C++, along with several variations in other languages, is available online [10].

2.4 Competitive coevolution

In *competitive coevolution*, the fitness of each network is evaluated by letting it compete with other networks in the same or in a different population. This means the strength of a network is only relative to other networks as improvement of one means decrease of another. The idea

is to let competing solutions outdo each other in order to create an ‘arms race’ [11]. Ideally speaking, the evolution is open-ended and the solutions improve with each generation indefinitely. In reality, however, there are several problems to address. True, open-ended “arms race” coevolution has never been achieved [1].

3 Method

3.1 Motivation

In order to answer the research question we will implement and evolve two neural networks for StarCraft: one for the traditionally evolved network, and one for the competitively coevolved version. We can then evaluate them by analysing the two networks' progress as they evolve, how well they perform against the default game AI and against each other.

We chose StarCraft as the game for two main reasons:

- There are several tools available which help with the implementation, primarily BWAPI (see Sec. 3.2) which allows us to easily create a bot to interact with the game.
- We believed the sheer possibility space offered by a real-time strategy game would allow for more complex behaviour to develop.

3.2 StarCraft: Brood War and BWAPI

StarCraft: Brood War is a real-time strategy game released by Blizzard Entertainment in 1998 and is one of the most successful games of its genre. The game takes place in a science fiction-based setting where the player takes the role of an army commander. In a real-time strategy game, the player controls multiple units and buildings. Gameplay involves unit production, base building, resource management, upgrading units, buildings and technology, and direct unit control. The larger scale strategic choices such as unit and resource production and base development - the "big picture" decisions - are referred to as *macro*. The direct control of units, i.e. deciding where they should go and which action they should perform, is referred to in the StarCraft community as *micro*.

There are three different races/factions in StarCraft: Brood War, each with its own playstyle:

- *Terran* consists of humans and favor versatility and average-costed units,
- *Protoss* are highly technologically advanced aliens and favor expensive, powerful units,
- *Zerg* are an insect-like alien race which favor large numbers of cheap units.

We have chosen to focus on two Terran units for our project: the Medic and the Marine. Using different races or many different units would make the problem space far too complex for the scope of this project.

In order to let a neural network manipulate units in the game, we will use the *Brood War API* (BWAPI). BWAPI is a third party open source C++ project which allows students, researchers and hobbyists to create Artificial Intelligence agents that play Brood War [12]. StarCraft: Brood War and BWAPI have previously been used to study NEAT and other approaches to machine learning [10], [11].

3.3 Overview

We implemented a version of NEAT capable of evolving a *bot*, i.e. an AI-controlled player of Brood War. BWAPI was used to interface with the game. We chose to focus on the micromanagement of a small number of units in the game, since the scope of evolving a comprehensive AI-controlled player is far too big. The same implementation of the bot was used for both the coevolution and the non-coevolved approach.

We used an existing implementation of NEAT written for Visual Studio [13] in order to simplify the integration with BWAPI. We used StarEdit to create a custom map with the attributes necessary for the experiment.

Each player controls four units (three marines and one medic) where the network determines the actions of each unit. The coevolved network plays against a player from another population, while the traditional network plays against the default Starcraft AI. During each game round, the players aim to kill each other's units. A game round ends when all marines on either side have been killed. When the round is over, the relevant data is summarized, all units are reset, and the next round begins.

Each unit has two possible outcomes that the bot chooses from: *action* and *evade*:

- If the unit evades, it calculates the average position of nearby enemies and moves in the opposite direction.
- If the unit acts and it is a marine, it attacks the lowest health enemy within weapon range. If no enemies are within range, it moves towards the average position of all enemies.
- If the unit acts and it is a medic, it heals the lowest health ally, prioritizing nearby allies.

The behaviour of these outcomes were explicitly defined. The output of the neural network was used to decide which action to take.

3.4 Experiment

3.4.1 Implementation

Design. We created a DLL implementing the virtual methods BWAPI uses to communicate with the game. Four of these functions were used to manage the game rounds:

- onStart() is called when the game starts. It was used to initialize some variables and generate the initial population if needed.
- onFrame() is called each logical frame. Most of the code was here: all units were updated by feeding input to the active network, if the round was over the network's performance was evaluated, and
- onUnitCreate(Unit) is called when a unit is created. This was used to check for new rounds: if the new unit is a medic owned by the bot, a new round has started since exactly one medic is created each round.
- onUnitDestroy(Unit) is called when a unit is destroyed. Was used to check the remaining number of marines and mark the round as ended or not, accordingly.

The bot loads the first population into memory when the game starts, and runs through them in turn. When all networks in the population are evaluated, the fitness values are calculated and the population of the next generation are created.

The network is only updated every 10 logical frames in order to give the previous action a chance to have a relevant effect.

Input. As network input, we used the following values:

- unit health,
- number of enemies within weapon range ("nearby enemies"),
- total number of enemies within twice the weapon range ("distant enemies"),
- number of allies within weapon range ("nearby allies"),
- total number of allies within twice the weapon range ("distant allies"),
- a value indicating whether the unit is currently attacking something (1 if it is, 0 otherwise),
- a value indicating whether the unit is currently being healed (1 if it is, 0 otherwise),
- a value indicating whether the unit is a medic (1 if it is, 0 otherwise).

Unit health serves to give the network information about how much danger the unit is in, and is expressed as a fraction of its maximum health. The two inputs for nearby and distant enemies serve a similar purpose, as do the inputs for the number of allies: they give the unit a rough idea of how dangerous the surrounding area is and how much support it has.

Whether the unit is being healed or is attacking something has direct implications for choices

in combat, as such that information is fed to the network. Making a distinction between medics and marines as input allows the network to behave differently for different units, but also allows for the specific differences to be optimised.

Distant enemies and distant allies are both scaled to the range [0..0.5], as their values should have less influence over the network compared to other inputs. All other input values are scaled to fall within the range [0..1].

Output. The inputs were calculated for each unit and entered into the network. The network's output then decided the action of that unit. The output consisted of two nodes: if Node 0 had an equal or larger value than Node 1, the unit would evade. Otherwise, the unit would act.

Fitness calculation. Calculating fitness in a way that accurately reflects the effectiveness of the network is very important for The fitness of a network is determined by the remaining health of the bot's units, the remaining health of the enemy's units, and the time taken to complete the game round.

The fitness was calculated as follows:

$$\frac{Squad\ health}{Enemy\ health + 1} - \frac{Frames}{1000} + 1 \qquad (1)$$

Squad health is the sum of the remaining health of the bot's units.
Enemy health is the sum of the remaining health of the enemy's units.
Frames is the duration of the game round in number of logical frames.

We determined experimentally that the duration of a game round lies at around 100-500 frames. We thus decided to use the number of frames divided by 1,000 in order to ensure that it has an appropriate effect on the fitness value. We also decided on a cutoff point of 1,000 frames where the game round would be considered over due to timeout. In order to discourage passive behavior that would cause a timeout, i.e. the bot tries to "hide" rather than defeat the enemy, a timeout resulted in a fitness of 0 regardless of health. NEAT does not work with negative fitness values, so 1 is added to ensure a positive fitness value.

The total health of the four units is 180, which means the range of possible fitness values theoretically lies within [0..181]. However, since game rounds typically end with medics at full health - the algorithm for finding the lowest health unit will almost always select a marine since they have lower starting health - the range will in practice lie within [0.8..3.7].

3.4.2 Execution

Automated evolution. Each approach was evolved for 20,000 generations. This was to ensure that the networks achieve significant results, since it can take thousands of generations before the network improves due to the complexity of the problem. The population size was 100 and each network was evaluated for one game round, for a total of 2,000,000 rounds. By manipulating the local speed of the game, we processed around 12,000 logical frames per second which meant that the experiment took in total 9-12 hours to complete. The experiment was repeated 5 times with different random seeds in order to improve the validity of the results.

The experiment was automated through the use of a custom StarCraft map which sets up the next round as soon as the previous round ends, through the use of the built-in trigger system. When either player had 0 marines, all units were removed and units for the next round were placed at predetermined locations. Chaoslauncher [14], a third party piece of software which adds functionality to Brood War, was used to automate menu navigation and inject the AI DLL into the game.

Benchmarks. The networks were benchmarked by letting them play 20 game rounds against the default StarCraft AI and examining both the win rate and the average fitness values. The win rate is the primary indicator: once the bot can consistently defeat the default AI, it can be considered better.

This benchmark was performed after the experiment as well as every 500 generations in order to track progress. The results from the benchmark were not used to train the networks.

Evaluation. After the coevolved and traditional networks were evolved for the same number of generations, they were evaluated by letting them play in six configurations:

- coevolved network vs traditional network,
- coevolved network vs default StarCraft AI,
- coevolved network vs coevolved network,
- traditional network vs traditional network,
- traditional network vs default StarCraft AI.

The win rate of the two networks in each configuration can then be compared and we can determine which one yielded a better result given the same learning time.

Recording data. As the simulation was being run, the population of neural networks was written to a file every 500 generations so that they could be evaluated in the benchmark later. Furthermore, the best, worst, and average fitness values of each generation were recorded continuously in the same way.

4 Results

Here are the results from the benchmarks and evaluation as described in section 3.4.2.

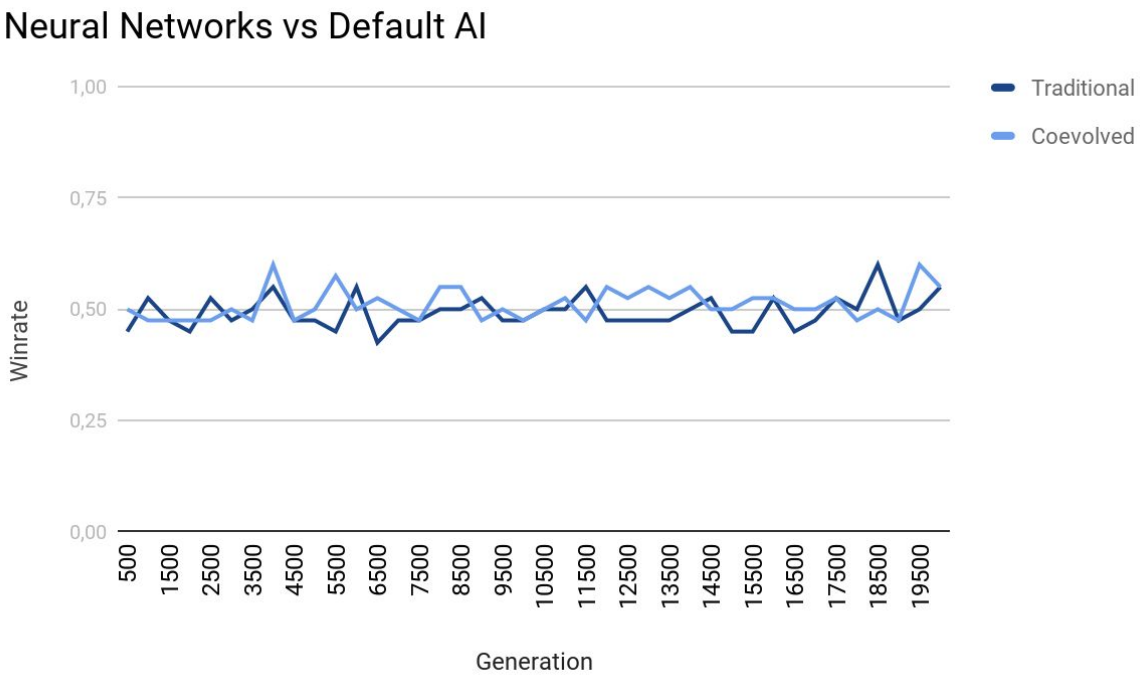

Figure 4.1: *Win rate of the best neural network of each benchmark generation versus the default StarCraft AI.*

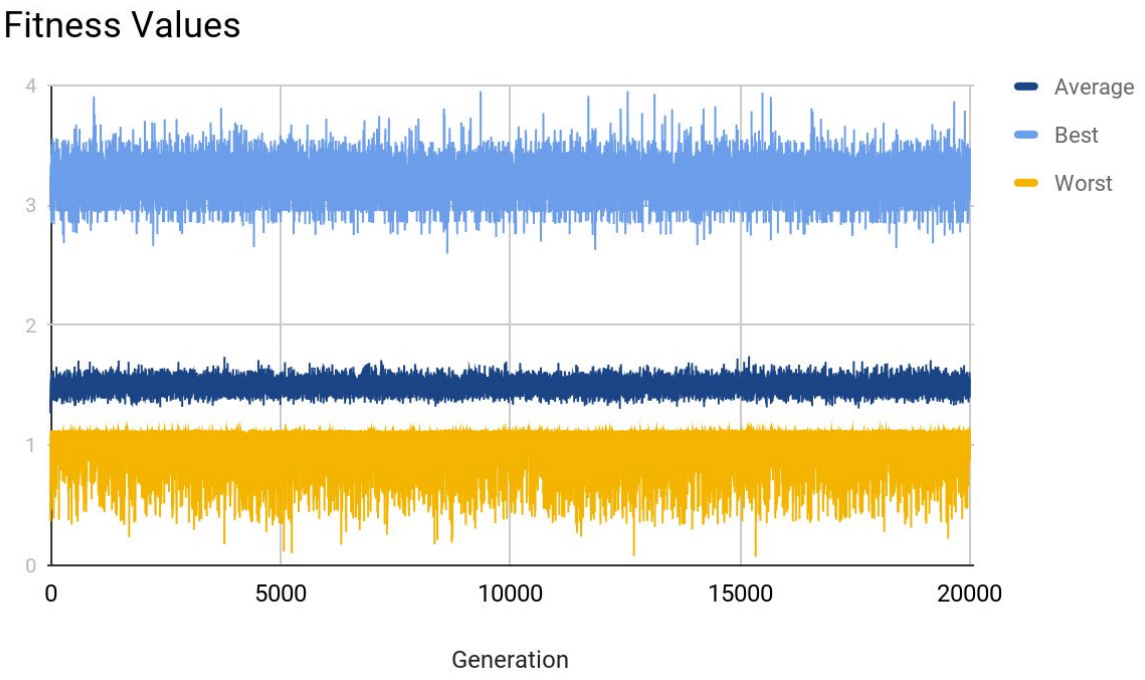

Figure 4.2: *Best, worst, and average fitness values from each generation of the traditionally evolved network.*

As shown in *figure 4.1*, both types of neural network displayed approximately the same win rate against the default AI, with no improvement between generations. Furthermore, it shows that there was no point where either network consistently outperformed the default AI.

Figure 4.2 shows how the fitness of the traditionally evolved network did not change significantly across generations. The fitness values mostly fall within the expected range [0.8..3.7] with some values even lower.

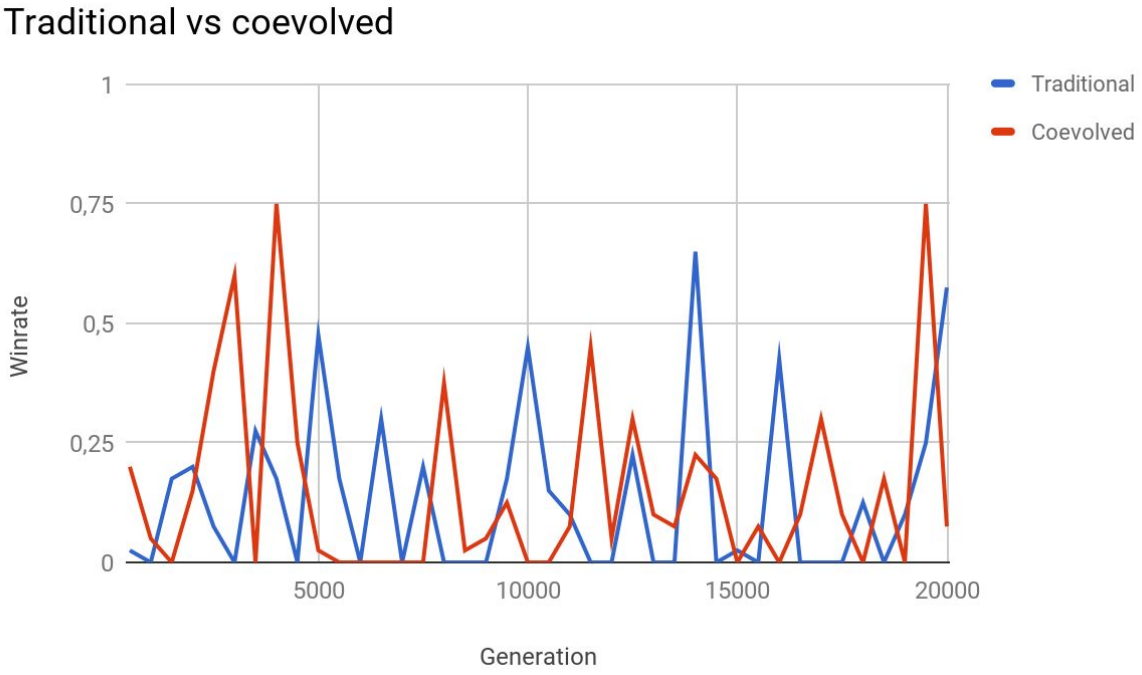

Figure 4.3: *Win rates between traditional and coevolved networks.*

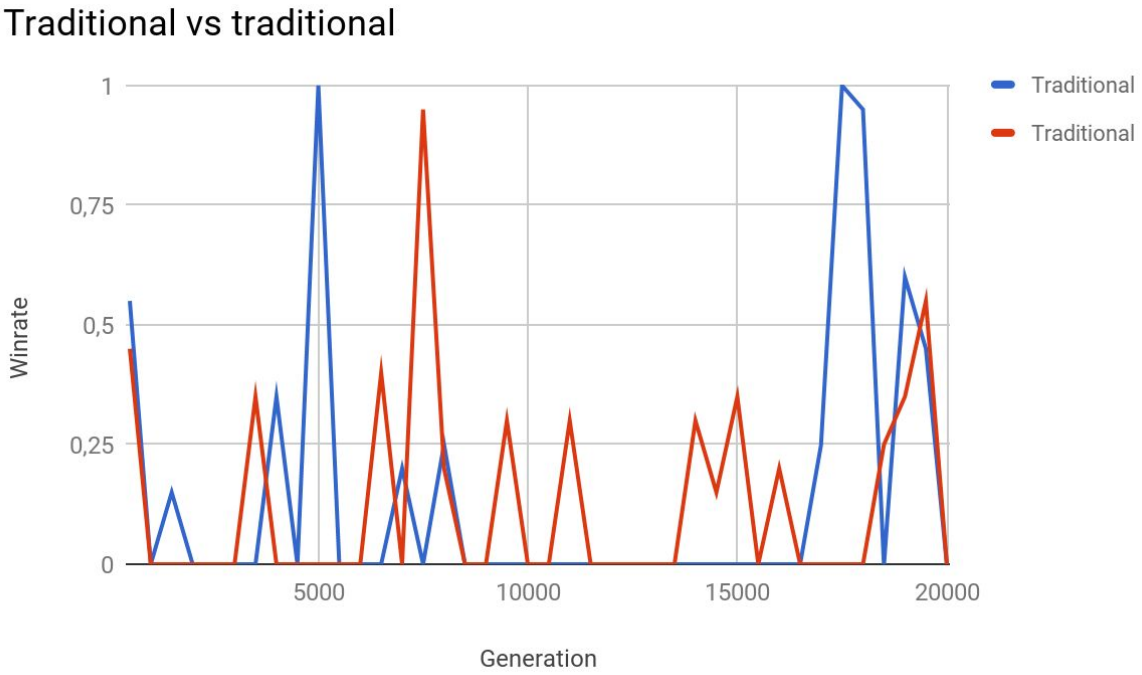

Figure 4.4: *Win rates of traditional network playing against itself.*

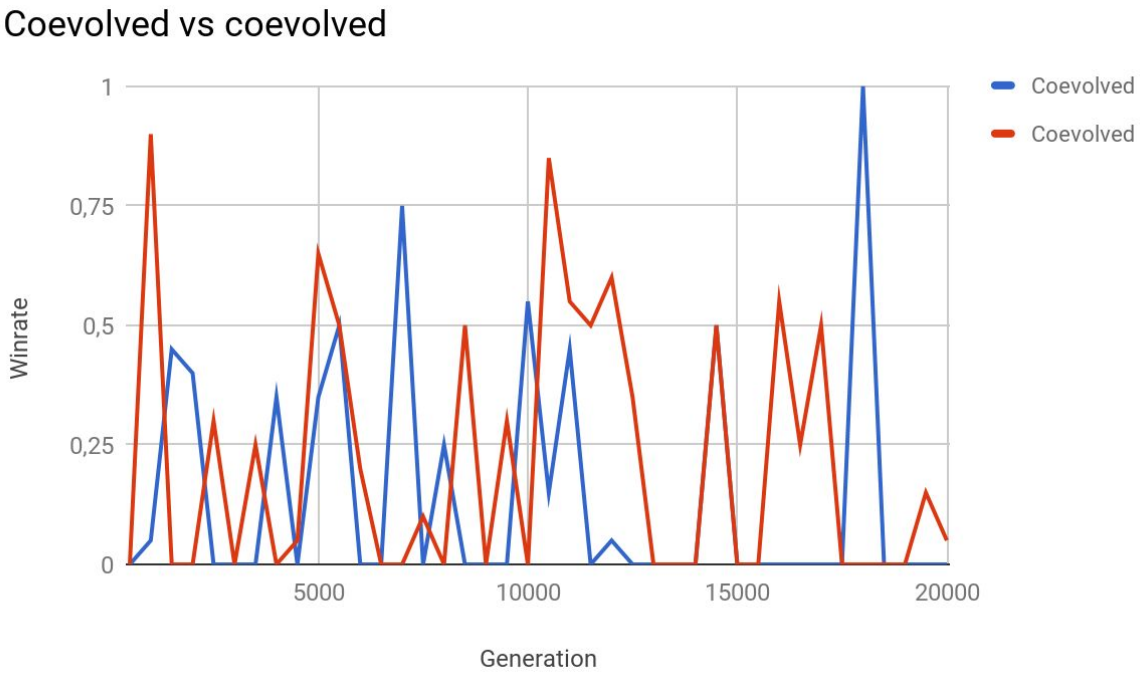

Figure 4.5: *Win rates of coevolved network playing against itself.*

As seen in *figure 4.3*, neither network type appears dominant when pitched against each other. The values are very inconsistent throughout generations.

Figure 4.4 and *figure 4.5* show that the traditional and coevolved network, respectively, also have very little consistency between generations when pitched against itself.

5 Analysis and Discussion

5.1 Evaluation against default StarCraft AI

Figure 4.1 shows that neither network improved throughout the experiment and that both networks were, at best, equally matched with the default AI. This could be due to one or more of the following:

- *Bad fitness function.* If the fitness value does not accurately represent the performance of the neural network, it will not be able to improve. The fitness values for most networks fell within the expected range of [0.8..3.7]. However, a fitness score above 3 indicates that the brain greatly outperforms the default AI, which was not the case.
- *Not enough evaluation.* Each member of each population was evaluated once, which might not have been enough to properly evaluate the effectiveness of a genome.
- *Output not sophisticated enough.* The output from each brain determined which action the network takes: either attack or retreat. It is possible that just having these two options was not enough for the network to develop any complex behavior.

The best fitness value from each generation was in the range of [3.0..3.5] as seen in *figure 4.2*, which implies the traditional network defeated the default AI with a lot of health remaining on those cases. However, these networks were evenly matched with the default AI as seen in *figure 4.1*, which shows a discrepancy in the evaluation process.

5.2 Evaluation between neural networks

As seen in *figure 4.3, figure 4.4* and *figure 4.5*, there is no observable trend when evaluating the networks against each other. This is to be expected, since neither type of network improved throughout the experiment. As there is no consistent strategy to improve upon, the winner of each matchup is essentially determined at random based on how the two brain's behaviors happen to interact.

5.3 Research question

Is there a difference in evolution speed between competitive coevolution and traditional neuroevolution when evolving AI-controlled players for StarCraft: Broodwar? If so, what is the difference?

No difference in evolution speed was found, as both types of neuroevolution failed to evolve in the given conditions.

6 Conclusion and Future Work

Conclusion

From the experiment we conclude that this NEAT implementation was incapable of evolving a bot under the given circumstances and thus unable to reject the null hypothesis. It is unclear what the specific problem with the implementation was. It could be bad fitness evaluation, not enough evaluation, or the output might not have been impactful enough to allow for a difference.

As the best networks throughout the generations performed on par with the default StarCraft AI, it seems likely that the fight-or-flight output of the neural network did not allow for complex enough strategies. Additionally, the consistently high fitness values of the best networks suggest that the fitness evaluation did not accurately reflect the neural network's performance.

The related work suggests that with the right configuration, competitive coevolution can potentially outperform traditional neuroevolution. We were however unable to achieve that in this thesis. Given more time, we might have been able to make the neuroevolution more effective and achieve better results.

Future work

In future work we would like to investigate different approaches to neuroevolution in order to successfully evolve a network and answer the research question. Different squad sizes and unit combinations could also be investigated.

References

[1] S. Risi and J. Togelius, "Neuroevolution in Games: State of the Art and Open Challenges," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 1, pp. 25–41, Mar. 2017.

[2] A. Lubberts and R. Miikkulainen, "Co-evolving a go-playing neural network," in *Coevolution: Turning Adaptive Algorithms Upon Themselves*, Birds-of-a-Feather Workshop, Genetic and Evol. Computation Conf. (GECCO-2001), 2001, p. 6.

[3] T. P. Runarsson and S. M. Lucas, "Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 628–640, Dec. 2005.

[4] J. S. Zhen and I. Watson, "Neuroevolution for Micromanagement in the Real-Time Strategy Game StarCraft: Brood War," in *AI 2013: Advances in Artificial Intelligence*, Dunedin, New Zealand, 2013, pp. 259–270.

[5] A. Shantia, E. Begue, and M. Wiering, "Connectionist reinforcement learning for intelligent unit micro management in StarCraft," in *The 2011 International Joint Conference on Neural Networks*, 2011, pp. 1794–1801.

[6] W.S. McCulloch and W. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, 1943, pp. 115–133

[7] K. Gurney, "Neural networks - an overview" in *An introduction to neural networks*. Boca Raton, FL: CRC Press, 1997, p. 1.

[8] L. Rutkowski, "Evolutionary algorithms" in *Computational Intelligence: methods and techniques*. Warszawa, Poland: Polish Scientific Publishers PWN, 2005, pp. 265–323

[9] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," in *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC '02*, 2002, vol. 2, pp. 1757–1762.

[10] K. O. Stanley (2015, May 5). *The NeuroEvolution of Augmenting Topologies (NEAT) Users Page* [Online]. Available: https://www.cs.ucf.edu/~kstanley/neat.html

[11] K. O. Stanley and R. Miikkulainen, “Competitive Coevolution through Evolutionary Complexification,” in *Journal of Artificial Intelligence Research*, 2004, vol. 21, pp. 63–100

[12] A. Heinermann (2017, April 19). *Brood War API* [Online]. Available: https://github.com/bwapi/bwapi

[13] M. Buckland (2002, September 7). *NEAT C++ for Microsoft Windows* [Online]. Available: http://nn.cs.utexas.edu/soft-view.php?SoftID=6

[14] Unknown author (2011, November 21). *Chaoslauncher* [Online]. Available: https://github.com/masterofchaos/chaoslauncher