NTNU – Trondheim
Norwegian University of
Science and Technology

The Effects of Supervised Learning on Neuro-evolution in StarCraft

Tobias Laupsa Nilsen

Master of Science in Computer Science
Submission date: Januar 2013
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Tobias Laupsa Nilsen

The Effects of Supervised Learning on Neuro-evolution in StarCraft

Master thesis, Spring 2013

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

This thesis explores the use of supervised learning in combination with evolutionary algorithms. The two techniques are used alone and in combination to train an artificial neural network to solve a small scale combat scenario in the real time strategy game StarCraft. The thesis focuses on whether or not it is indeed beneficial to use the two in combination and how injecting human knowledge through logged examples influences the results of the evolutionary algorithm. In the small scale combat scenario a number of agents must cooperate to defeat an equal number number of enemies. The different approaches to training the network are tested and it is found that using human knowledge to create an initial population for the evolutionary algorithm dramatically improves performance compared to the other approaches, and is able to produce solutions to the scenario of high quality.

Preface

This Master thesis is a part of the requirements for the master of technology in computer science at the department of Computer and Information Science at NTNU. The supervisor for this thesis is Keith Downing.

Tobias Laupsa Nilsen

Trondheim, January 11, 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the work which will be done in this thesis. Section 1.1 briefly introduces background for the problem and the authors motivation. Section 1.2 introduces the goal of the thesis and the underlying research questions. Section 1.3 introduces how the research questions will be investigated, and what experiments will be carried out. Section 1.4 outlines what this thesis will contribute to the scientific community, and finally section 1.5 presents the structure of the rest of the thesis.

1.1 Background and Motivation

StarCraft is a computer game in the real time strategy(RTS) genre. Released in 1998 it was a massive success, popular with gamers and critics alike, it sold more than 11 million copies making it one of the best selling computer games of all time. Its popularity was such that it spawned a professional league of StarCraft players, world championships with considerable prize-money, and was not really overtaken until the release of its sequel StarCraft 2 in 2010.

Some of the games popularity can no doubt be credited to its complexity and balanced gameplay, meaning that to date no single strategy has been found that cannot be countered by a skilled player. To play the game successfully the player must solve a number of difficult problems in a dynamic multi-agent environment in real time. These problems range from finding the best strategy and production plan to path finding and low level control of troops.

In this thesis we will focus on the management of troops referred to by players as micro-management, and in the rest of this thesis as small scale combat. The computer will be given a number of military units and tasked with destroying an equivalent force placed nearby, the second force will be controlled by StarCrafts default AI. Good solutions to such a problem would not just benefit the games community, but could potentially be of use in other similar environments, as will be discussed in section 2.1.

1.2 Goals and Research Questions

In this thesis we will explore different ways of finding good solutions to a small scale combat scenario in the RTS game StarCraft. Two different methods will be used, both alone and in combination, to train an artificial neural network(ANN) which will function as a controller for individual agents in the scenario. The two methods are an evolutionary algorithm(EA) and learning using the back-propagation(BP) method. The goal of the thesis can be summarized as follows:

Goal To determine whether or not BP learning used in conjunction with EAs is advantageous compared to EAs or BP learning used alone.

Both BP learning and EAs have been used successfully to solve complex problems in agent control, this thesis will explore if a combination of these two techniques is better suited for the purpose of finding the weights of an ANN agent controller than each of them used in isolation.

Research question 1 Is it advantageous to use BP learning prior to EAs?

Used prior to the evolutionary process BP learning can functions as a guide or seed, avoiding the proliferation of many individuals with very low fitness values, but it can potentially steer the evolution into a local optima.

Research question 2 Is it advantageous to use BP learning after EAs?

Using BP learning after applying some form of evolution could function as a fine tuning of the network, refining the findings of the global search of evolution with the local gradient descent based back-propagation algorithm. It has been found that this combination can be more effective than either one used independently due to EAs perceived weakness in fine tuning and BPs sesitivity to initial conditions.[Yao, 1999]

BP is however as a supervised learning method entirely dependent on its teaching examples, and in a complex environment such as StarCraft these examples can

be hard to accurately capture or find and very hard and time consuming to hand author. Furthermore even if the examples themselves are very good they may reflect a different strategy from the one found by the evolutionary process and may therefore lead to worse performance.

Research question 3 Is an EA better suited to determine the weights of an ANN controller than BP learning?

Which of these methods, BP, or an EA will be better suited to make a controller for the chosen scenario when used on its own.

1.3 Research Method

To find answers to the questions asked in the previous section I have built a system capable of logging actions taken by a StarCraft player, making, training and using ANNs as controllers for StarCraft and an EA capable of searching for optimal weights for the ANN. Using this system i will investigate whether or not using BP learning before or after the evolutionary process, confers advantages over BP learning or an EA used alone.

The system will be used to conduct a series of experiments. First an EA will be run 20 times with a population of randomly created individuals.

Secondly a set of training examples will be created, by logging the actions of the author in the scenario to be solved. These examples will be used to train 20 sets of weights for the ANN controller using BP learning.

Then 20 experiments will be run with the EA using one or more of the best performing solution found by BP as seeds to create the initial population.

The best solutions found in each of the EA experiments will then be subjected to a round of BP learning using the same examples as in the previous experiment.

The solutions found will be compared on how successfully they solve the problem i.e. how many percent of the games they play they are able to win.

By comparing the performance of the solutions found by the different approaches used on the problem it will be possible to comment on whether or not using BP learning in conjunction with an EA is more effective at finding good solutions than any of the techniques used in isolation.

1.4 Contributions

The contributions of this thesis can be outlined as follows:

1. Showing whether or not BP learning used in conjunction with EAs is advantageous compared to EAs or learning used alone on this problem.
2. Determining if ANNs are a suitable choice for agent control in StarCraft and similar environments.
3. Discussion of the suitability of StarCraft and similar games for research in bio-inspired AI.

The thesis will explore the suitability of using Neural networks as controllers for unit behaviour in the real time strategy game StarCraft, and how best to train these networks. The focus of the thesis is on the use of EAs and BP learning and how these methods can best be used to solve a complex problem.

The thesis will focus on whether or not combining the two methods have advantages over on or the other used on its own. Based on the results it will also be possible to comment on whether or not ANNs are suitable for agent control in complex environments such as StarCraft.

Finally based on the experiments it will be possible to have a discussion about whether or not StarCraft and games like it are suitable domains for research in the field of biologically inspired artificial intelligence.

1.5 Thesis Structure

The rest of this Thesis is structured as follows:

Chapter 2 will start with a brief introduction to StarCraft, then the specific problem to be solved will be presented alongside a discussion on the features of the problem and its relation to other problems and fields of AI. This is followed by a brief introduction to the techniques used in this thesis, feed-forward neural networks, genetic algorithms, and neuro-evolution. Systems solving similar problems will be presented and discussed in relation to the work done in the thesis.

Chapter 3 explains the system that has been built, what parts it comprises, and how they work.

Chapter 4 begins by presenting how the experiments outlined in section 1.3, has been performed using the system described in chapter 3. Following this the results of the experiments are presented and discussed in relation to the comparative

performance of the different approaches and try to explain why some approaches are performing better than others.

Chapter 5 begins with a discussion of what the results obtained in chapter 4 suggest about the research questions posed in section 1.3. Following this there is a discussion of the validity of these results, and a summary of the contributions of the thesis. The chapter ends with a brief description of possible directions for future research.

Chapter 2

Background Theory and Motivation

This chapter presents background theory necessary to understand the rest of this thesis and related work. Section 2.1 presents information about the game StarCraft, the problem chosen for this thesis and why StarCraft and games like it are well suited for AI research. Section 2.2 presents the techniques used in this thesis i.e. feedforward neural networks and genetic algorithms. Section 2.3 presents work done by others on problems similar to the one used in this thesis. Section 2.4 discusses how previous works relate to the work of this thesis.

2.1 StarCraft

As mentioned in section 1.1 StarCraft is a computer game in the real time strategy(RTS) genre. The game is set in the far future in a distant galaxy and the story revolves around a war between three different races, all of which have distinctly different buildings and units at their disposal necessitating quite different play-styles and strategies. The Protoss race are quite humanoid and focuses on powerful but expensive units, the Zerg are quite diverse but usually insectoid in appearance and focuses on cheap and plentiful but weak units. The final race the Terran are humans and fall somewhere in between the two other in relation to power vs cost. StarCraft updates the game state and draws visuals to screen, roughly 25 times per second, each of these updates are in the community and in the rest of this thesis referred to as a frame.

RTS games are characterised by the player being put in charge an initially small detachment of military units and must use them to collect resources, defend his position and expand to obtain more resources and ultimately destroy his enemies. To successfully do this the player must solve a number of problems on two levels of abstraction commonly referred to as macro- and micro-management.

Macro-management consists of high level strategic decisions which include but are not limited to:

- Choosing which buildings to build and when to build them.
- Choosing which units to train and when to train them.
- Choosing when, where and with what units to attack the enemy.

Micro-management on the other hand is concerned with carrying out parts of the larger macro plans and can include but are not limited to:

- Choosing where to place buildings and what workers to use.
- Moving units from one place to the other with minimal casualties.
- Controlling units in battle as effectively as possible.

This thesis focuses on one specific facet of RTS game-play mainly the control of units in combat situations. To simulate this I have created a custom scenario for StarCraft, see figure 2.1, containing five units controlled by the player, on the left, and five enemies controlled by the default AI of StarCraft.

Figure 2.1: The scenario used in this thesis.

The unit chosen for this scenario is the terran marine, the most versatile of the basic units available in StarCraft. The marine is fills the role of general purpose infantry being a man with a gun he has a ranged attack able to hit both ground and air targets from afar.

As a point of clarification in this thesis the word player refers to one of the two entities which are in control of a number of units, the word agent is used to refer to the units themselves i.e. the marines. This choice is based on the scope of the work as it explores only the small scale combat part of StarCraft and not the larger strategic parts of the game, even though the players can be considered agents in their own right.

RTS games are good venues for AI research because they are quite detailed simulations of reality[Buro and Furtak, 2003]. While the combat of StarCraft is a simplification over actual tactical combat the agents in this thesis must operate in a complex environment with the following properties:

- **Partial observability**: Positions on the map which are not currently occupied by one or more of the units are not visible to the player. In this thesis which focuses solely on small scale combat this has been relaxed so that the agents have access to the enemies position before they are directly visible. This is done because the development of a scouting/searching strategy is outside the scope of the problem. This also eases implementation of the system.
- **Deterministic(Strategic)**: The game is largely deterministic with the exception that bullets fired at a target has some small chance of missing their target. The chance for a bullet to miss is almost 50% when units are firing at enemies occupying high ground. In this thesis the map used is flat containing no high ground as such the only source of real uncertainty is the actions of the enemy agents.
- **Sequential**: An action taken in one state effects all subsequent states.
- **Dynamic**: The environment changes due to the actions of other agents.
- **Continuous**: The environment is continuous both with respect to time and state.
- **Multi-Agent**: In this thesis the ANN player controlling five agents is opposed by one other player controlling five identical agents. The environment contains both hostile and friendly agents necessitating that the agents cooperate to be able to destroy their enemies.

As we can see these characteristics are quite similar to the real world, with the obvious exception that the real world contains a lot more uncertainty. Making it plausible that lessons learned by solving problems in StarCraft could potentially be used on real world problems such as autonomous driving or navigation by other autonomous agents, such as robots, operating in dynamic real time environments.

2.2 Background Theory

This section will describe the solution techniques used in this thesis.

2.2.1 Artificial Neural Networks

Artificial neural networks(ANNs) is a name given to a broad class of networks consisting of simple interconnected processing units called neurons connected to each other via weighted connections.

The neurons typically function by summing its inputs and then applying some simple mathematical function to the sum, this is inspired by the functioning of neurons in the brains of humans and higher life forms where neurons function as threshold detectors. The weighted connections are meant to simulate the axons in the nervous system where neurons can both inhibit and excite the neurons they are connected to in varying degrees.[Floreano and Mattiussi, 2008; Purves, 2012]

These different networks have, in spite of being at best crude approximations to actual nervous systems, been shown to have remarkable properties. Capable of learning to solve complex problems like classification, clustering, time series prediction, and function approximation.[Kohonen, 1990; Hornik et al., 1989] ANNs have been employed in robotics as controllers, and a field known as computational neuroscience uses them to glean insights into the functioning of the human brain.[Lewis et al., 1996; Churchland et al., 1993]

The neural networks considered in this thesis belong to a class of networks referred to as multilayer perceptrons or feedforward neural networks.[Floreano and Mattiussi, 2008] Feedforward neural networks can, given the right topology and weights approximate any function to an arbitrary precision.[Hornik et al., 1989] These networks have three distinguishing characteristics:

1. They are organized into layers consisting of one or more neurons each.
2. The networks have directionality, connections go only in one direction from layer N to layer N+1.
3. The networks are fully connected. All neurons in layer N receive input from all neurons in layer N-1 and send their output to all neurons in layer N+1.

Figure 2.2 shows a feedforward neural network with three layers consisting of a total of seven neurons propagating signals from left to right through the network.

The training method used in this thesis is known as back-propagation(BP). A supervised learning algorithm which uses examples of input and appropriate output to train the network. The BP algorithm can be described as follows:

1. Read The first input-output example.

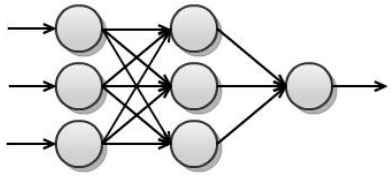

Figure 2.2: Feedforward neural network with three layers.

2. Use the input as input to the ANN.
3. Calculate the error between the desired output and the actual output.
4. Calculate the error of each individual neuron in the network and use it to change the weights.
5. If there are more examples read the next one and go to step 2.
6. If the error is is greater than a user defined threshold go to step 1.
7. If the error is below the given threshold the algorithm is finished.

The above algorithm is adapted from Callan [1998] page 38.

Learning using this method can, like all gradient descent based methods, potentially get stuck in local error minimum rather than finding the optimal solution to the problem, and the convergence can be very slow particularly on larger networks.[Hinton, 1989; Floreano and Mattiussi, 2008]

2.2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) refers to a class of algorithms that use concepts and operations known from evolutionary biology to search the best solution to a given problem. EAs differ from actual evolution in a number of ways, but most drastically in the fact that while evolution is an open ended unguided process with no end point, EAs need well defined fitness functions which means that the search is guided and will end when a suitable individual is found or after a predetermined number of generations.

Like ANNs EAs have proven to be useful in solving many difficult problems,

and excel at global optimisation problems. EAs have also had success designing digital and electrical circuits, antennas and numerous other applications.[Weile and Michielssen, 1997; Floreano and Mattiussi, 2008]

Genetic algorithms(GA) are a subset of the larger category of EAs, they are a way of searching for solutions in a way which is meant to mimic important aspects of natural evolution. They are characterised by their representation of the genome as binary strings or vectors of real numbers, and their emphasis on using the crossover operator.[Whitley, 2001] The basic GA can be expressed as:

1. Initialize a population of individuals
2. Test the individuals on the problem to be solved and assign a performance measure
3. Create a new population by using genetic operators on the population based on the performance measure.
4. Let the new population replace the old one.
5. Unless a stopping criterion is met, such as a good enough individual found or the maximum number of generations performed, go to step 2.

The population consist of a number of individuals whose genes are often expressed as vectors of binary or real valued numbers. The performance measure is commonly referred to as a fitness value and is meant to represent the individuals suitability to its environment.

The genetic operators used in GAs are crossover and mutation. Crossover is meant to mimic the operation of genetic recombination when parents reproduce by blending the genetic material of the parents to produce the children.

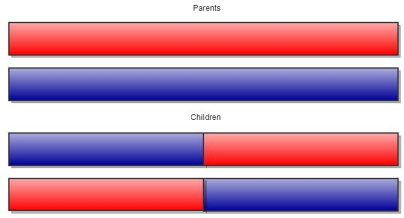

Figure 2.3: Example of one point crossover.

Figure 2.3 illustrates a simple case of one point crossover where the genes of the parents are recombined to form the children. The crossover can occur at multiple points and the points are usually chosen at random.

Mutation is meant to mimic the natural deviation in the genetic material during reproduction. When the genome is represented by a binary string mutation simply takes the form of flipping a one to a zero and vice versa, when the genome is represented by real numbers, mutation often take the form of adding random numbers drawn from some distribution.[Yao, 1999]

2.2.3 Neuro-Evolution

The combination of these techniques, ANNS and EAs, is called neuro-evolution(NE), and the resultant networks are often referred to as evolutionary artificial neural networks(EANNs). Evolution has been applied to ANNs in a number of different ways from directly finding the best weights, to finding optimal learning rules or activation functions for the network.[Yao, 1999] EANNs have, like its constituent parts shown itself to be a viable solution to a wide range of problems, like controlling legged robots, playing checkers and as controllers for video game characters.[Chellapilla and Fogel, 1999; Clune et al., 2009; Stanley et al., 2005]

In this thesis a genetic algorithm will be used to search for weights for an ANN which will be used to control the marines in the scenario described in section 2.1 and seen in figure 2.1.

2.3 Related Work

Hagelbäck and Johansson [2008a,b] details what the authors call a multi-agent potential field(MAPF) bot. A bot being a name given to computer programs takes the place of the human player in a computer game. The principle to is allow all units and objects in the map to be surrounded by fields which are meant to mimic electrical fields attracting or repelling the units under the players control. A matrix is created which details the perceived value of each position on the map. This allows for the abstraction of spatial information because the agents do not themselves have to reason about the exact or relative positions of their allies or enemies, rather they can simply move towards favourable positions on the map and be repelled from unfavourable ones.

The charges of these fields were set with trial and error. The method was put to the test in the Open Real Time Strategy(ORTS) competition of 2007 and achieved below average results. The authors identified a number of weaknesses with their solution and developed an improved version capable of decisively beating all the top contenders of the competition. Showing both that the implementation and

sophistication the potential fields are very important for the overall performance, and that artificial potential fields can be very effective in an environment very similar to the one in StarCraft.

Sandberg and Togelius [2011]; Rathe and Svendsen [2012] both uses genetic algorithms to tune the parameters of potential fields for use in combat situations in StarCraft. Sandberg and Togelius are able to show a clear improvement in the performance of evolved solution and find solutions which perform very well, concluding that EAs are indeed effective ways of training MAPF bots for play in StarCraft. Rathe and Svendsen also uses an EA to tune the charge values of the different fields differentiating themselves by using multi objective optimization in lieu of single fitness values. Their results are weaker than those achieved by Sandberg and Togelius something they attribute to the implementation of their potential fields, again showing that potential fields are a powerful technique but very dependent on its design and sophistication to achieve high performance.

Shantia et al. [2011] uses several neural networks to approximate the value functions of an agent performing an action in its current situation. The networks are trained in two different scenarios very similar to the one used in this thesis. Like the scenario used in this thesis both scenarios consists of equal forces of marines fighting. In the first scenario each team must coordinate three marines, in the second each team consists of six marines.

The networks are trained using two variants of a reinforcement learning algorithm called sarsa, awarding the neural networks rewards or punishments online by the effects of their actions on the game world every few game frames. In the 6 versus 6 scenario incremental learning starting with the best performing networks from the 3 versus 3 scenario is contrasted with starting the networks of with randomized weights. The networks are provided complete information of the game world and uses 9 different vision grids reminiscent of artificial potential fields to abstract information about the game world such as the firing ranges of enemies.

The learning algorithms are able to successfully solve the 3 versus 3 scenario but had considerable difficulty finding good solutions to the more difficult 6 versus 6 scenario. The results showed that to find good solutions to the 6 versus 6 problem incremental learning was necessary. The results indicate that neural networks can be used successfully to evaluate state information and the values of action in a problem very similar to the one used in this thesis. The results also suggest that reinforcement learning is better able to solve difficult problems when starting from some semi functional solution.

Ki et al. [2006] uses real time NE to tune the weights of an ANN to imitate the actions of a human player in a RTS. The actions of a human player is logged during play and used to train the networks in real time. The networks are taught to imitate a simple strategy of retreating when health gets low to survive. The results demonstrate that even very simple neural networks without hidden nodes are able to learn strategies and function well in a problem very similar to the one used in this thesis. It also shows that it is possible to use an ANN to imitate human actions in this environment.

Fan et al. [2003] proposes a method called rule-based enforced sub-populations (RESP) building on the enforced sub-population(ESP) method proposed by Gomez and Miikkulainen [1997]. ESP is a method of evolving ANNs where each individual represent a single hidden node in the network rather than the full network itself. The network topology is decided and for each hidden node in the network a sub-population of possible hidden nodes are initialized. These sub-population are closed so that crossover is only performed between members of the same sub-population. The individuals are evaluated by randomly picking one member of each sub-population, making up a complete network, and evaluating the network, this is done many enough times that each individual is likely to have been tested a sufficient number of times.

RESP is enhanced by creating the initial network by translating a rule-base into an ANN and using this as a starting point for ESP evolution. The method is shown to outperform ESP on a task where multiple predators must cooperate to catch a prey, even if rules are randomly removed from the rule-base. The result suggest that using a rule base to inject human knowledge into the evolutionary process allows for solving more difficult problems even if the rule base itself is incomplete or damaged.

Gabriel et al. [2012] describes their work creating a multi-agent small scale combat bot for StarCraft using rtNeat a real time variant of neuro-evolution of augmenting topologies(NEAT).[Stanley et al., 2005]

NEAT is an NE method which evolves both the weights and the topology of neural networks. NEAT starts from a collection of minimal networks and add complexity during evolution, using genetic markers to ensure that crossover is applied between similar individuals, and uses speciation to protect innovation which may not be immediately beneficial. [Stanley and Miikkulainen, 2002]

RtNeat uses the same method as NEAT but does it in real time running evaluations of the individuals after a specified number of frames. This was devised

and used by Stanley et al. [2005] in the NERO video game in which the player instructs robots, who each represent an individual with its own ANN, which learn by rtNEAT and then pit them against robots trained by other players.

Gabriel et al. [2012] trains their agents by running 12 vs 12 matches against both the default AI of StarCraft and two of the best performing bots of the 2010 AIIDE StraCraft AI competition. The method is tried in four different scenarios where the sides switch between being made up of units which can attack from range and units using melee attacks for a total of four combinations:

- Ranged vs melee
- Ranged vs ranged
- Melee vs ranged
- Melee vs melee

Each game is run with each side starting with 12 individuals and 100 reinforcements, when a unit is killed another is created subtracting from the remaining reinforcements until they are depleted. Every 500 frames in the game the units are evaluated and the worst performing units are replaced. The system is able to learn to beat the default AI in all the scenarios quite convincingly, but has a much harder time defeating the more advanced AIs. It still performs quite well and is able to win or tie 7 out of 8 scenarios against the 2 advanced AIs, even if some of the victories are very narrow.

The results show that neural networks using evolution can learn to perform very well in the domain of StarCraft even outperforming very advanced AI implementations. One of the advanced AIs tested is the Overmind winner of the 2010 AIIDE Stracraft competition which uses potential fields tuned with reinforcement learning to control its small scale combat behaviour.

2.4 Motivation

The literature seem to suggest ANNs are indeed capable of producing good solutions to problems very similar to the one used in this thesis, and that they can indeed produce solutions that rival those of the most prevalent and successful method used on these kinds of problems namely artificial potential fields and static rules.[Gabriel et al., 2012]

The literature also show that the initial conditions of reinforcement learning techniques do effect the outcome. Both Fan et al. [2003] and Shantia et al. [2011] reports improved performance when starting their methods with some imperfect

solution. In the case of Fan et al. [2003] a manually created network and for Shantia et al. [2011] solutions found to a less complex problem.

This thesis aims to find out whether or not it is beneficial to include human knowledge into the evolutionary process in the search for good solutions to a very complex multi-agent problem which requires the agents to cooperate. As such the aims of this research are similar to those of Fan et al. [2003]. This work differs from that of Fan et al. [2003] in both the complexity of the problem to be solved and the methods used to solve it.

StarCraft is a more complex environment than the predator prey domain used in Fan et al. [2003], among other factors in that the actions of the enemy agents are far more unpredictable.

The method is different in that it does not require the manual creation of a rule-base but rather the logging of actions performed in game which is then learned by the network through BP. The method also differs from the other works in the field in that it does not use a variant of the ESP or NEAT method of evolution but a simpler more conventional GA.

This work also differentiates itself from other works in the field in the way the ANN will be used. All the above works, which operate in StarCraft and uses an ANN, uses the ANN as a selector of one of a number of preprogrammed behaviours whereas in this thesis the output of the ANN directly codes the action to be taken i.e. what coordinates to move to and what enemy to attack.[Shantia et al., 2011; Gabriel et al., 2012; Ki et al., 2006]

If successfully able to solve the problem presented in section 2.1 this work would suggest that while certainly effective in their own right, more advanced NE algorithms, such as ESP and rtNEAT, are not strictly necessary to solve the complex problem of small scale combat in StarCraft, and that combining human knowledge through BP with a GA can be a very effective strategy for finding solutions to this and similar problems.

Chapter 3

Implementation

This chapter presents a brief overview of the system that has been created to investigate the research questions of section 1.2, what it is capable of doing and what parts it is made up of. Section 3.1 outlines the capabilities of the system and the three different components the system comprises. Section 3.2 details how the different capabilities of the system are implemented.

3.1 Overview

The system consists of three main parts, the BWAPI client, the neural network, and the population. The client contains the main loop of the program and is responsible for getting information from, and sending commands to StarCraft, as well as using the two other components. The neural network is the controller which is consulted by the client to determine which action to take in a given situation. The population is used when running the genetic algorithm, it contains the individuals to be tested in StarCraft and uses genetic operators on the individuals based on the fitness scores it is given by the client.

The system can be used in the following ways:

- Create an ANN.
- Logging actions taken by a human user.
- Training ANNs with back-propagation.
- Running a genetic algorithm to find weights for ANNs.

- Testing the found solutions by running them as many times as deemed necessary on the problem.

3.1.1 The Client

The client is the main component of the system, it is a stand alone program which can be injected into StarCraft and has complete access to all information about the game, and can give all the same commands that a human player could using. The client uses BWAPI an open source api which allows for the creation of custom AIs for StarCraft, and is based on the example client which is part of BWAPI.

The client is responsible for updating and consulting the two other components it also writes to and reads results and individuals to and from text files.

3.1.2 The Network

The network component implements feedforward neural networks using a sigmoid activation function. This component can be used to create feedforward neural networks with any number of inputs, outputs, and hidden layers with an arbitrary amount of hidden nodes in each.

The network can be fed with information and activated, trained with back-propagation learning and all its weights can be retrieved and changed. A neural network can be initiated either with randomly chosen weight values or with an existing vector of weight values.

3.1.3 The Population

The population is a collection of individuals on which genetic operators can be used. Each individual is an object containing an id, a fitness score, and a vector of double precision floating point values(doubles) representing the weights of the neural network.

A population can be initiated, randomly or using on or more seeds. If initiated randomly each and every individual of the population will be initiated with its vector of doubles randomly picked from a Gaussian distribution with a given mean and deviation. If initiated with one or more seeds the population is made up of one copy of each seed unchanged and the rest of the population created

by adding mutated copies of each seed to the population until the population is full.

Reproduction is done by selecting two parents stochastically based on the fitness scores of the individuals, giving more successful individuals a greater chance to procreate than their less successful counterparts.

The genetic operators used on the population is one point crossover and mutation. Crossover is implemented by picking a random number N ranging from zero to the number of hidden nodes and then moving over all weights associated with the first N hidden nodes.

Crossover is handled in this way to avoid the potentially destructive effects of removing half of the weights associated with a hidden node. Hidden nodes functions as feature detectors in the data they are presented and having half of its weights randomly removed is almost certainly destructive.[Yao, 1999] Handling crossover in this way allows networks to switch feature detectors rather than destroying them.

Mutation is implemented by iterating through the vector of doubles and with a chosen probability adding a double picked from the Gaussian distribution with a specified mean and deviation.

3.2 The System

The three components briefly outlined above will be used to create an ANN and train it to solve the problem presented in section 2.1.

3.2.1 Game Play

When playing the game the client cycles through all five units collecting their state information, running it through the ANN and sending move or attack commands to the game every 15 frames. The 15 frame delay was chosen as a result experimentation during development. very short delays between orders was observed to lead to poorer performance. This can be attributed to the commands being issued faster than the agents were able to carry them out. This was observed to be particularly detrimental to the attack command as it takes several frames to carry out, leading to largely pacifistic agents. Even with this delay the client issues 5 commands approximately 100 times in a minute adding up to 500 commands issued per minute.

3.2.2 Logging Examples

When logging examples for use with the BP algorithm, the client collects the same information as it does when playing the game itself, and stores it together with the corresponding actions the units are carrying out. Figure 3.1 shows a sample of the training data recorded by the system. All the examples code the same action, in this case attacking the closest enemy. The first line is the game state and the following line is the action taken in that state. The choice of data corresponds to the inputs and outputs of the ANN used in this thesis and is explained in detail in section 3.2.3

```
0.898438 0 1 1 0.107143 0 0.924479 0.178571 0 0 0 0.107143 0 1
0 0 0 1 0 0
0.925781 0 1 0 0.428571 0.571429 0.924479 0.0357143 0 0.821429
0 0 0 1 0 0
0.949219 0 0 1 0.892857 1 0.924479 0.5 0.5 1 1 0.892857 1 1 0.
0 0 0 1 0 0
0.9375 0 1 1 0.107143 0 0.96875 0.178571 0 0 0 0.107143 0 1 0
0 0 0 1 0 0
0.972656 0 1 0 0.428571 0.571429 0.96875 0.0357143 0 0.821429
0 0 0 1 0 0
0.996094 0 0 1 0.892857 1 0.96875 0.5 0.5 1 1 0.892857 1 1 0.2
0 0 0 1 0 0
0.9375 0 1 1 0.107143 0 0.96875 0.178571 0 0 0 0.107143 0 1 0
0 0 0 1 0 0
0.972656 0 0 0 0.428571 0.571429 0.96875 0.0357143 0 0.821429
0 0 0 1 0 0
```

Figure 3.1: Example of the data logged by the system.

3.2.3 The Neural Network

The neural network used as a controller in this thesis, shown in figure 3.2, has 3 layers with 29 inputs, 5 hidden nodes, and 6 outputs, for a total of 175 weights, making it a quite complex network. The black dots in figure 3.2 indicates that 19 input neurons have been left out to simplify the figure.

The inputs used are:

- Agent status
  - Agent heading
  - Agent hit points
  - Agent weapon status
  - Agent under attack
- Allies status
  - Centroid position relative to the agent
  - Centroid heading

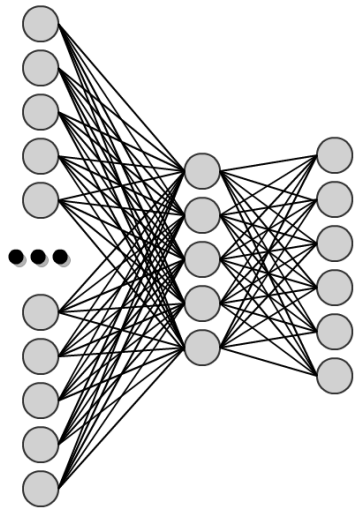

Figure 3.2: The neural network used as a controller in the thesis.

  - Relative position of the three closest allies

- Enemies status
  - Centroid position relative to the agent
  - Centroid heading
  - Relative position of the three closest enemies
  - Whether or not each of the three closest enemies are under attack
  - Hit points of each of the three closest enemies
- The ratio of ally to enemy hit points

The choice of inputs was taken partly based on the work done by Shantia et al. [2011], and partly based on the authors own understanding of the game.

The choice was also based on trying to make an ANN that could potentially scale to deal with a larger or smaller number of agents. If the number of agents increase each agent will still only consider the three closest enemies and allies plus the centroids of each group, when the number of agents decrease, as it does every game due to deaths, the corresponding inputs to the network is fed the position

of the centroid of the corresponding group. The choice to use relative coordinates was also taken in an effort to let the network generalize better.

The relative coordinates are calculated based on the position of the agent and the other unit and scaled by the sight range marines have in the game. In the game this sight range is 7. If the difference in the x- or y- coordinate between the agent and the other unit is in the range of (-7, 7) it is divided by 7 to get a number in the range (-1, 1). If the difference is greater the relative coordinate is set to -1 or 1. The coordinates are then scaled to be in the range of [0, 1].

The choice to use five hidden nodes was taken after experimentation with BP showed that while more hidden nodes led to better training results the actual performance on the problem did not improve.

Through experimentation it was found that five hidden nodes were as simple as the network could be made without sacrificing performance. Fewer hidden nodes have the advantages of shortening the time it takes to teach the network through BP, dramatically shrinks the solution space for the GA, and can lead to better generalization.[Sietsma and Dow, 1991; Fletcher et al., 1998]

The six outputs directly encode the action the agent should take in its current situation. The first output functions as a boolean determining whether the agent should move to a new position or attack an enemy. Outputs 2 and 3 are the relative x and y coordinates the agent should move to. The last three outputs functions as boolean values determining which of the three closest enemies should be attacked. As the network uses the sigmoid activation which only takes on the values one and zero as results of incredibly high activation and approximation by the computer, activations below 0.2 and above 0.8 will be considered as 0 and 1 respectively.

3.2.4 Back-propagation

The BP algorithm is implemented in the manner described in section 2.2.1 using examples to train the ANN described in section 3.2.3, with some modifications. First a momentum term changing the weights based not only on error but also on the previous weight change to hopefully avoid local minima.[Floreano and Mattiussi, 2008]

Secondly because the network ignores part of its output during operation based on the output of the first output-neuron, the error of the ignored output-neurons do not contribute to the error of the network or cause changes to their weights. This is done because they are irrelevant and there exists no right answer as to what value they should take.

3.2.5 The Genetic Algorithm

The GA works by initializing a population in one of the ways described in section 3.1.3 and each of them are tested as controllers a number of times. Each time one of the teams of agents are destroyed or the time runs out, the individual is awarded a fitness score based on its performance.

The fitness score is calculated as 100 points per destroyed enemy, 500 for victory and 100 points for each living agent. This adds up to a total of 1500 points for a perfect victory and is averaged over the number of trials. These values were chosen because they are very easy to obtain and was designed to favour winning individuals over good but losing individuals, awarding the closest of victories 1100 points, almost three times as many points as the closest of defeats which award the individual 400 points.

After all the individuals in the population have been tested the fitness scores are used to produce a new generation using the genetic operators described in section 3.1.3. The algorithm can also uses elitism to avoid losing good solutions when creating a new generation.

The algorithm runs for a given number of generations to find the best solutions, a fitness score is not used as a stopping criterion because the performance of each individual varies significantly between trials and generations. This is primarily due to the variations in how the default AI behaves which can change between trials, and generations. If a really good individual is discovered it should survive or at least be able to make a significant impact on the populations due to the elitism allowing it to survive multiple generations.

At the end of each generation the best performing individual is saved to file along with the best, and average fitness obtained during the generation.

3.2.6 Testing

Due to the variability the individuals show in the trials the fitness scores are not completely representative of the actual performance of the weights of the individual. To properly test the found solutions the solutions will be evaluated by running them a sufficient number of times to find their actual performance. The performance measure used will be the number of victories alongside the average kill score. The performance measures are not equally important as a high win percentage is obviously better than a high kill score.

It is technically possible to lose every game and yet get an average kill score of 400 points out of a maximum 500 points if every game is lost by the closest

possible margin, while winning half the games but losing the last half without destroying a single opponent would give a kill score of only 250. As shown by the quite unlikely examples the kill score can act as a measure of how stable the performance of the solution is, but is only meaningful as a secondary performance measure alongside the percentage of games won.

By experimentation it was found that when running the individuals 100 times the results were consistent to within a couple of percentage points. This usually takes at least 5 minutes to run even on the fastest possible setting, which would make it far to time consuming to use in the genetic algorithm.

Chapter 4

Experiments and Results

This chapter presents the experiments done to answer the research questions. Section 4.1 explains what experiments will be run and in what order. Section 4.2 details the experiments and the parameters used to allow for reproduction of the results. Section 4.3 presents the results of the experiments. Section 4.4 discusses what mau cause the differences in performance.

4.1 Experimental Plan

To answer the research questions of section 1.2, the following experiments will be carried out.

1. The genetic algorithm(GA) will be run 20 times with randomized starting weights for all individuals.
2. Back-propagation(BP) will be used to generate 20 sets of weights based on input-output pairs logged from human play.
3. The GA will be run 10 times using the one of the best performing sets of weights found through BP as a seed as explained in section 3.1.3.
4. The GA will be run 10 times using the five best performing sets of weights found through BP as seeds as explained in section 3.1.3.
5. BP will be applied to 15 of the best individuals found by the genetic algorithm, the individuals will be picked evenly from the experiments carried out in step 1, 3 and 4.

The result of these experiments will be compared by testing them in the way described in section 3.2.6 and noting win percentage and average kill score i.e. the average number of enemies destroyed per game. This will address the research questions presented in section 1.2, by showing the quality of the solutions obtained by the different approaches, and allows for analysis as to what may cause the differences in performance.

4.2 Experimental Setup

4.2.1 Experiment 1: GA only

The GA was run 20 times with 20 individuals in the population over 100 generations. In each generation each individual was tested 5 times consecutively. These parameters were chosen so as to limit the time the algorithm would take to run to completion while still giving it sufficient time to find good solutions. With 20 individuals being tested 5 times each per generation for 100 generations, a total of 10 000 games, the algorithm takes a minimum of 4 hours to run to completion.

The genes of the individuals were randomly instantiated by pulling each of the 175 real numbers from the Gaussian distribution with mean 0 and variance 1. All 20 trials used 30% elitism, preserving the 6 best individuals for the next generation.

The mutation rates and the variance of the mutation were varied, each combination of these parameters were tested 5 times. These parameters can be found in table 4.1.

4.2.2 Experiment 2: BP only

The BP algorithm was used 20 times to produce 20 individuals with a learning rate of 0.1 with stopping criteria lower cumulative squared error than 0.02 or 500 revolutions. These parameters were chosen because it was observed during testing that solutions with significantly higher errors performed very poorly. The networks were initialized with weights in picked from the Gaussian distribution with mean 0 and variance 0.3.

The 0.02 was rarely achieved and as such it was necessary to use a secondary stopping criterion, set at 500 because it was observed that at this point the algorithm had usually got stuck, fluctuating between very similar solutions.

The test set is a file containing 1167 examples of inputs and corresponding outputs captured from the author playing the scenario 3 times, using the system in the way described in section 3.2.2.

4.2.3 Experiment 3: BP then GA 1 seed

The GA was run 10 times using one of the five best sets of weights found in experiment 2,using only BP, as a seed to initialize the first generation. The number of generations, individuals per generation and trials per individual were identical to experiment 1.

The the set of weights used was entry number 19 in 4.4 the overall fourth best performing set of weights found winning 14% of its matches. This seed was chosen among the five best seeds because it seemed to be one of the best at retaining its formation before attacking the enemy. The other individuals in the first generation are made by mutating the seed as described in section 3.1.3 with a mutation rate of 0.5 and a variance of 1.

All 10 trials used 30% elitism like experiment 1 and varying rates and variance of mutation each combination was tested 2 times, these parameters can be found in table 4.2.

4.2.4 Experiment 4: BP then GA 5 seeds

The setup of this experiment is identical to the one used in experiment 3, with the exception of the initial population being made up of the 5 best performing solutions found in experiment 2, using BP only, and mutated copies of them.

4.2.5 Experiment 5: GA then BP

The parameters and examples used in experiment 5 are the same as the ones used in experiment 2. The difference being that BP will be used on networks loaded with the 5 best performing solutions of experiment 1, 3 and 4, for a total of 15 tests.

4.3 Experimental Results

4.3.1 Experiment 1: GA only

The detailed results of experiment 1 can be found in table 4.3, and will be summarised here.

When tested it was found that only one of the 20 trials performed produced a set of weights capable of beating the opposition often enough to be considered a good solution to the problem. This set of weights, entry number 6 in table 4.3, won 69% of its matches. The other 19 solutions did significantly worse, the second best performing solution won only 27% of its matches, and the performance averaged over 20 solutions was 18.75%, as can be seen in the last entry of Table 4.3.

During testing it was observed that the best performing solution was not only quantitatively better performing than the other solutions but indeed qualitatively different in its functioning. It was the only solution where the agents used the move command to position themselves effectively before attacking the enemies.

The 19 other solutions did not position themselves effectively before they attacked but simply ran of to attack the enemies. This sub optimal behaviour was picked up early and is usually observed in the very first generation. This very simple strategy was so successful relative to the other early strategies that it dominated the evolution completely.

4.3.2 Experiment 2: BP only

The complete results of the solutions found by using BP 20 times can be found in table Table 4.4.

Overall the resulting performance was poor, on average the solutions won only 8.6% of their matches. The best performing solution found performed about on par with the average performance of the solutions found in experiment 1 i.e. winning 19% of its matches. The difference in average performance between experiment 1 and 2 is statistically significant at a confidence level of 95%, showing that the GA is better able to find solutions to this problem than using BP with the logged examples.

These solutions did however share a characteristic with the best performing solution found in experiment 1, using GA only, that is they used both the move and the attack commands to position themselves before and under the attack.

Trials	Mutation rate	Variance
1-5	0.3	0.5
6-10	0.4	0.8
11-15	0.5	0.8
15-20	0.5	1

Table 4.1: The mutation rate and mutation variance of experiment 1.

Trials	Mutation rate	Variance
1-2	0.3	0.4
3-4	0.3	0.5
5-6	0.4	0.8
7-8	0.5	0.8
9-10	0.5	1

Table 4.2: The mutation rate and mutation variance of experiment 3 and 4, BP then GA 1 and 5 seeds.

Trial	Victories	Kill score
1	13	284
2	19	315
3	22	360
4	21	322
5	15	287
6	69	410
7	11	290
8	15	284
9	15	302
10	11	291
11	11	290
12	12	288
13	16	298
14	26	337
15	15	293
16	13	298
17	27	326
18	14	283
19	10	283
20	20	291
Average	18.75	306.6

Table 4.3: Results of experiment 1, GA only.

Trial	Victories	Kill score
1	16	249
2	4	111
3	4	139
4	4	140
5	5	186
6	4	169
7	2	116
8	8	149
9	3	119
10	10	152
11	11	197
12	2	130
13	10	160
14	14	243
15	13	163
16	5	206
17	17	225
18	7	120
19	14	221
20	19	250
Average	8.6	172.25

Table 4.4: Results of experiment 2, BP only.

4.3.3 Experiment 3: BP then GA 1 seed

The complete results of experiment 3 can be found in table 4.5. Half of the solutions clearly outperform even the best solution found in experiment 1. The average performance is as high as 63.7% clearly outperforming both the approaches of experiment 1 and 2. The difference in average performance between experiment 1, and experiment 2 and 3 is, despite the relatively low number of trials performed, statistically significant at a confidence level of 95%.

The best performing solution, entry 8 in table 4.5, achieves a win percentage of 96% being an almost perfect solution to the problem.

The two worst performing solutions, achieving win percentages of only 25% and 16%, were observed to be similar to the ones generally found in experiment 1. Early in the evolutionary process a strategy of attacking the opposition immediately was found, and through the stochastic nature of the algorithm and the evaluation, these were able to outperform all others leading the algorithm to get stuck in a local optima.

All other solutions employed strategies where they positioned themselves before attacking, and the best performing solution has the agents moving towards the enemy in an almost perfect line formation.

4.3.4 Experiment 4: BP then GA 5 seeds

The complete results of experiment 4 can be found in table 4.6. The average win percentage of the found solutions is 60.5% slightly lower than the average win percentage achieved by the solutions found in experiment 3.

Again a qualitative similarity was seen between the solutions generally found in experiment 1 and the worst performing solution found in this experiment. This solution employs the strategy of immediate attack, all other solutions found position themselves before attacking.

The performance achieved is more even than the one observed in experiment 3, having fewer highs and lows than the results of experiment 3. Its best performing member achieves a win percentage of 75%, while experiment 3 produced 4 superior solutions one achieving a win percentage of 96%. The average performance is better than the ones achieved in experiment 1 and 2, using only the GA or BP respectively, at a significance level of 95%. The difference in average performance between experiment 3 and 4 is not statistically significant.

Trial	Victories	Kill score
1	63	381
2	78	446
3	72	449
4	87	479
5	87	472
6	57	405
7	56	392
8	96	494
9	25	337
10	16	312
Average	63.7	416.7

Table 4.5: Results of experiment 3, BP then GA 1 seed.

Trial	Victories	Kill score
1	69	429
2	67	414
3	69	428
4	75	441
5	73	441
6	25	336
7	62	392
8	43	352
9	57	392
10	65	424
Average	60.5	405.2

Table 4.6: Results of experiment 4, BP then GA 5 seeds.

4.3.5 Experiment 5: GA then BP

As can be clearly seen in table 4.7 The application of BP universally degrades the performance of the solutions subjected to it. The average win percentage of the solutions subjected to BP tumbles from 62.53% down to only 10.33%. When BP was applied it was observed that the initial error was often very high, indicating that most of the solutions had found very different input-output mappings than the ones encoded in the examples.

4.4 Discussion

A overview of the results of all 5 experiments can be found in table 4.8 presenting the mean number of victories, the number of victories achieved by the best individual and the 95% confidence interval of the mean for each experiment.

The solutions found in experiment 1, using the GA with randomly initialized individuals, was observed to almost exclusively use the very simple strategy of using only the attack action. Only one solution was found where the agents were able to position themselves effectively to achieve a good performance. It was observed during the early generations that the individuals generally used either the move command or the attack command exclusively, and because only one of these actions by themselves provide a direct boost to the fitness score, the individuals using attack only are able to dominate the early and usually subsequent generations. This seems to suggest that it is very difficult to find solutions able to both position themselves and attack capable of performing on par with this very simple solution through random search.

The problem of getting stuck in local optima could perhaps be remedied by altering the fitness function, which only awards successfully destroying enemies or winning the game, to also show some form of preference to movement. This was considered and rejected for two reasons, the first is that this would be more difficult to implement as all simple strategies, such as awarding fitness points for the number of move commands, could potentially lead to the proliferation of individuals which simply walked around taking the most circuitous route possible.

Furthermore movement is not really useful in this problem unless it is coordinated, that is the agents move in some formation, this would be very hard to award points for without human inspection during the operation of the GA. As mentioned in section 4.2 a single run of this GA takes at least 4 hours making human inspection during operation very impractical.

Experiment	Solution	Victories before	Victories after
1 GA only	6	69	5
1 GA only	17	27	0
1 GA only	14	26	12
1 GA only	3	22	5
1 GA only	4	21	2
3 BP then GA 1 seed	8	96	44
3 BP then GA 1 seed	4	87	40
3 BP then GA 1 seed	5	87	6
3 BP then GA 1 seed	2	78	0
3 BP then GA 1 seed	3	72	7
4 BP then GA 5 seeds	4	75	14
4 BP then GA 5 seeds	5	73	0
4 BP then GA 5 seeds	1	69	4
4 BP then GA 5 seeds	3	69	4
4 BP then GA 5 seeds	2	67	12
Average		62.53	10.33

Table 4.7: Results of experiment 5, GA then BP.

Experiment	Mean	Best	95% confidence interval
1 GA only	18.75	69	14.30 - 23.20
2 BP only	8.6	19	4.15 - 13.05
3 BP then GA 1 seed	63.7	96	49.32 - 78.08
4 BP then GA 5 seeds	60.5	75	46.12 - 74.88
5 GA then BP	10.33	44	2.66 - 18.00

Table 4.8: Overview of the results. Showing the mean number of victories, the number of victories for the best solution, and the 95% confidence interval of the mean

The second reason why this was rejected was that this thesis investigates how human knowledge can be incorporated into the evolutionary process by BP and the effects of this. It was therefore decided that the fitness function would only be a measure of how well each individual did on the problem, and not itself inject constraints on how the problem should be solved.

The solutions found in experiment 2, using the BP algorithm exclusively, showed poor performance. Not a single truly good solutions was found, and the best solution was only able to win 19 out of 100 games, only slightly better than the average from experiment 1. The difference in mean performance between experiment 1 and 2 was found to be statistically significant. The solutions were however observed to be able to use both the move and the attack actions although not particularly effectively. The poor performance can probably be attributed to the examples captured being of poor quality. This can potentially be attributed to one or more of three main factors:

- The logging method might capture some idiosyncratic or nonsensical states.
- The examples might include contradictory actions being recorded in similar states.
- The strategy recorded might be poor.

All of these can potentially explain why the networks which were trained to conform to the examples fared poorly. Suggesting that the capturing of examples during game play has not been particularly successful.

The performance of the solutions found in experiment 3 and 4, using one or more of the solutions found in experiment 2 to initialize the population of the GA, clearly outperforms all other combinations of these two techniques. The overall good performance is interesting in light of the poor performance of the solutions used to initialize the populations. It was hypothesized in section 1.3 that previously found solutions to could trap the GA in a local optima. Instead it is observed that using previously found solutions as seeds dramatically improves performance and results in much more diverse strategies than the ones generally found by the GA alone. While the GA can still be trapped in the local optima of only attacking it happens much rarer than with randomly instantiated individuals.

The author believes that the difference observed is due to the fact that the solutions used as seeds while rather poor, are just good enough to be competitive with the simple attack strategy early on, while employing more advanced movement strategies. This prevents the population from being dominated by the simple attack strategy and allows for the evolution of more advanced strategies.

Using more than one seed to initialize the GA was tried for fear of the populations being to homogeneous and steering into a local optima, this leads to more even but slightly worse results. It is possible that this approach is less susceptible to local optima but needs more time to find truly good solutions. This is suggested by experiment 3 producing 4 solutions superior to the best one found in experiment 4. Even so the difference in mean performance is very similar and not statistically significant.

The performance of the solutions found in experiment 5 showed a universal degradation of performance by training the solutions found with BP. This can be explained by the same factors that explain the performance of the solutions found in experiment 2, using BP only. It is interesting to note that the initial ancestors of entries 6-15 in table 4.7, having used solutions found by BP as seeds, once complied quite well with the test set used have now removed themselves so substantially from their origins, that trying to make them comply is very destructive to their performance. Suggesting that they have been able to develop strategies significantly different from the one encoded by the examples.

The best performing solution found was found in experiment 3 using a single solution found through BP as a seed to initialize the GA. This solution was able to win 96 out of 100 games an almost perfect solution to the problem. The strategy it employs is to form up into a tightly packed line facing the enemy and moving slowly towards the enemy usually correcting if they loose formation.

The slow movement towards the enemy allows them to provoke some of the enemy agents into attacking without backup from the rest of the enemy force, allowing them to usually fight with their full force against only parts of the opposition. The tightly packed line maximises the collective shooting range of the group allowing them to output the maximum damage as early as possible on approaching enemies. The tightly packed formation also makes it very likely that the agents input of 3 closest enemies will have significant overlap, this together with the input of which of the 3 closest enemies are currently under attack it seems allows them to focus their fire reasonably well.

In summary: As can be seen in table 4.8 using previously found solutions to initialize the population of a GA clearly outperforms the other methods tested, even if the previously found solutions shows rather poor performance. The GA used alone is likely to get stuck in local optima and the difficulty with capturing appropriate training examples leads to poor performance for both BP used alone and to tune solutions found by the GA.

Chapter 5

Evaluation and Conclusion

This chapter concludes the report. Section 5.1 evaluates the results presented in chapter 4 in relation to what they can tell us about the research questions presented in section 1.2. Section 5.2 discusses the merits and limitation of the thesis. Section 5.3 lists the contributions of the work. Section 5.4 concludes the thesis with suggestions on possible directions for further research.

5.1 Evaluation

In section 1.2 the goal of the thesis was defined as:

Goal To determine whether or not learning used in conjunction with EAs is advantageous compared to EAs or learning used alone.

From this broad goal formulation the research questions were defined as follows:

Research question 1 Is it advantageous to use learning prior to EAs?

Research question 2 Is it advantageous to use learning after EAs?

Research question 3 Is an EA better suited to determine the weights of an ANN controller than BP?

5.1.1 Research Question 1

The results of experiment 3 and 4, presented in section 4.3 clearly shows that using BP learning prior to the EA results in drastically better performance on the problem. It was observed that using BP prior to the EA prevented the EA from getting stuck in local optima and find better solutions. This despite the poor performance achieved by the BP algorithm in experiment 2.

The difference in mean performance is statistically significant from the mean performance obtained in experiment 1 and 2, using an EA and BP in isolation respectively. One solution found successfully solves the problem 94 out of 100 times being an almost perfect solution to the problem, whereas the best performing solution found using the EA in isolation solves the problem 69 out of 100 times. Suggesting that injecting human knowledge through supervised learning prior to an EA can indeed be advantageous.

5.1.2 Research Question 2

The results of experiment 5, presented in section 4.3 shows that the performance of the solutions subjected to BP after being found by the EA fell drastically. This is probably caused by poor quality of the examples, and that the solutions found by the EA are using different more effective strategies than the one encoded by the examples. On this and other problems where it can be very difficult to capture or manufacture training examples for supervised learning the results indicate that it is not advantageous to use supervised learning on the solutions found by the EA.

5.1.3 Research Question 3

The result of experiment 1 and 2, using an EA and BP alone respectively, presented in section 4.3 suggest that when training examples are hard to obtain an EA is better suited to find weights for an ANN than BP.

5.1.4 Conclusion

The experimental results suggest that it is indeed advantageous to use learning together with an EA on this problem, but only prior to the application of the EA. Using supervised learning in the form of BP prior to the EA can potentially prevent the EA from getting stuck in local optima and thereby being able to

find better solutions to the problem. Interestingly using BP after the EA to fine tune the found solution was found to not result in better performance despite numerous findings indicating this methods efficacy.[Yao, 1999] This is probably due to the difficulty of obtaining good training examples and the multitude of different strategies available. Finally the results indicate that on a problem where training examples can be hard to obtain EAs are better suited than supervised learning to determine the weights of an ANN.

5.2 Discussion

The experimental results obtained in this thesis suggest that injecting human knowledge through supervised learning into an EA has significant advantages over using randomly instantiated individuals. The result show that this approach clearly outperforms all other combinations of these two methods and is capable of finding very good solutions to the complex problem used in this thesis.

The results are very unlikely to be an aberration as they are statistically significant at a confidence level of 95%. The results are also strengthened by the fact that they suggest the same conclusion as the work of Fan et al. [2003] and to a certain extent Shantia et al. [2011] namely that the performance of EAs are dependent on their initial conditions. How this work relates to other works was discussed in section2.3.

The results of the work is limited by the fact that it has only been subjected to a single problem in a single domain and as such it can not be guaranteed that the results are transferable to other domains or problems. As discussed in section 2.1 RTS games are very complex environments which presents simplified but still very complex problems to be solved. The author argues that the properties of the environment and the nature of the problems are complex enough that solutions and methods found to work in StarCraft are likely transferable to other domains.

Another limitation of the work is the relatively low number of experiments run, this is a direct consequence of the properties of StarCraft. StarCraft is a commercial game not meant for research purposes, as such the only way to access it is through the open source project BWAPI, which as mentioned in section 3.1 is an open source project. Because of it being a commercial game, it can only be sped up so far, meaning that a single run of the GA used in this thesis takes at least 4 hours, which over 40 runs adds up to at least 160 hours of pure simulation time. This has limited the number of runs and the number of parameters which were tried out.

5.3 Contributions

The contributions of this work were briefly presented in section 1.4 as

1. Showing whether or not learning used in conjunction with EAs is advantageous compared to EAs or learning used alone on this problem.
2. Determining if ANNs are a suitable choice for Agent control in StarCraft and similar environments.
3. Discussion of the suitability of StarCraft and similar games for research in bio-inspired AI.

This work has showed that using BP prior to create better initial individuals for an EA can be a simple and effective way of training ANNs to be effective controllers for agents in StarCraft, and likely other similar environments.

The performance of the best solutions obtained in this thesis and the work of Gabriel et al. [2012], Shantia et al. [2011] and Ki et al. [2006] shows that ANNs can function effectively as controllers in StarCraft.

As discussed in section 2.1 StarCraft is a very complex environment containing a multitude of complex problems good solutions to which can be of use in other arenas. The main drawback of StarCraft is the time it takes to run the simulations necessary when working with EAs. The author has also frequently read that there is a limitation on the number of games that can be run without incident, but has himself run over 20 000 games consecutively without incident, and has had none of the 40 experiments stopped due to technical difficulties.[Sandberg and Togelius, 2011; Rathe and Svendsen, 2012; Gabriel et al., 2012] In the authors estimation StarCraft is a suitable but not perfect environment for bio-inspired AI research, with the drawback being long simulation times.

5.4 Future Work

This section suggests possible directions in which to extend the work of this thesis.

5.4.1 Generality

The solutions found in this thesis are optimised to perform well on a single problem it would be interesting to see if it would be possible to train ANNs which

performed well in more than one scenario. Varying the layout of the map the units in them and so on to see if it would be possible to find networks with good performance in all situations. Showing ANNs capable of being a more general solution of the problem rather than a single solution to a single problem.

This could take the form of successive incremental evolution using good solutions from simpler task as a starting point to solve more complex problems as in Shantia et al. [2011].

5.4.2 Coevolution

The ANN in this thesis only plays against the default AI of StarCraft it would be interesting to see what kind of solutions could be found using coevolution. Coevolution can be used in two different ways competitively or cooperatively.

Competitive coevolution would pit two teams of agents against each other allowing them to potentially develop much more sophisticated and effective strategies than the current setup where the opposition is more static.

Cooperative coevolution could evolve each agent individually on the to see if different behaviours would arise for each agent potentially allowing them to carry out more advanced and effective tactics. Even more interestingly if the team was made up of different units it could be observed whether this led to them taking distinctly different roles in the team.

5.4.3 Integration with Artificial Potential Fields

Artificial potential fields have been used with success by Hagelbäck and Johansson [2008a], Sandberg and Togelius [2011], and UC Berkleys entry in the 2010 AIIDE Starcraft competition as described by Buro and Churchill [2012]. They provide a powerful way of abstracting several key features of the environment such as the shooting ranges of enemies, high ground, impassable terrain and so on. The technique itself does not specify how the agents should make decisions based on the information that is usually implemented with simple means. It would be interesting to see how an ANN could use this information and what strategies could be found.

Bibliography

Buro, M. and Churchill, D. (2012). Real-time strategy game competitions. *AI Magazine*, 33(3):106.

Buro, M. and Furtak, T. (2003). Rts games as test-bed for real-time ai research. In *Proceedings of the 7th Joint Conference on Information Science (JCIS 2003)*, pages 481–484.

Callan, R. (1998). *The Essence of Neural Networks*. Prentice Hall PTR.

Chellapilla, K. and Fogel, D. (1999). Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87(9):1471 –1496.

Churchland, P., Koch, C., and Sejnowski, T. (1993). What is computational neuroscience? In *Computational neuroscience*, pages 46–55. MIT Press.

Clune, J., Beckmann, B., Ofria, C., and Pennock, R. (2009). Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2764 –2771.

Fan, J., Lau, R., and Miikkulainen, R. (2003). Utilizing domain knowledge in neuroevolution.

Fletcher, L., Katkovnik, V., Steffens, F., and Engelbrecht, A. (1998). Optimizing the number of hidden nodes of a feedforward artificial neural network. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 2, pages 1608 –1612 vol.2.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press.

Gabriel, I., Negru, V., and Zaharie, D. (2012). Neuroevolution based multi-agent system for micromanagement in real-time strategy games. In *Proceedings of the Fifth Balkan Conference in Informatics*, pages 32–39. ACM.

Gomez, F. and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342.

Hagelbäck, J. and Johansson, S. (2008a). The rise of potential fields in real time strategy bots. *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Hagelbäck, J. and Johansson, S. (2008b). Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 631–638. International Foundation for Autonomous Agents and Multiagent Systems.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40(1-3):185 – 234.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

Ki, H.-w., Lyu, J.-h., and Oh, K.-s. (2006). Real-time neuroevolution to imitate a game player. In Pan, Z., Aylett, R., Diener, H., Jin, X., Göbel, S., and Li, L., editors, *Technologies for E-Learning and Digital Entertainment*, volume 3942 of *Lecture Notes in Computer Science*, pages 658–668. Springer Berlin Heidelberg.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464 –1480.

Lewis, F., Yesildirek, A., and Liu, K. (1996). Multilayer neural-net robot controller with guaranteed tracking performance. *Neural Networks, IEEE Transactions on*, 7(2):388 –399.

Purves, D. (2012). *Neuroscience*. Sinauer Associates.

Rathe, E. and Svendsen, J. (2012). *Micromanagement in StarCraft using Potential Fields tuned with a Multi-Objective Genetic Algorithm*. PhD thesis, Norwegian University of Science and Technology.

Sandberg, T. and Togelius, J. (2011). Evolutionary multi-agent potential field based ai approach for ssc scenarios in rts games.

Shantia, A., Begue, E., and Wiering, M. (2011). Connectionist reinforcement learning for intelligent unit micro management in starcraft. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1794–1801. IEEE.

Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67 – 79.

Stanley, K., Bryant, B., and Miikkulainen, R. (2005). Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on*, 9(6):653 – 668.

Stanley, K. and Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1757–1762. IEEE.

Weile, D. and Michielssen, E. (1997). Genetic algorithm optimization applied to electromagnetics: a review. *Antennas and Propagation, IEEE Transactions on*, 45(3):343 –353.

Whitley, D. (2001). An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817 – 831.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423 –1447.