University of Nevada, Reno

Neuroevolution for Realtime Strategy Game Micromanagement

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

by

Aavaas Gajurel

Dr. Sushil J. Louis/Thesis Advisor

May, 2019

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

AAVAAS GAJUREL

Entitled

Neuroevolution for Realtime Strategy Game Micromanagement

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Sushil J. Louis, Ph.D., Advisor

David Feil-Seifer, Ph.D., Committee Member

Binod Guragai, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2019

Abstract

We explore the use of neuroevolution to evolve control tactics for groups of units in real-time strategy (RTS) games. RTS games have units with different characteristics in weaponry, movement and abilities, which makes effective coordination of units in real time a challenging problem. We focus on micro control, which requires quick planning and decision making during skirmishes between two groups of opposing units. We created a new representation that can effectively map game state to the neural input domain and evolved the network using neuroevolution of augmenting topologies (NEAT) to control movement and attack commands for each unit. We applied this approach to a group of ranged units skirmishing with a group of melee units in StarCraft II, an RTS game. The evolved neural networks performed well and lead to kiting behavior for the ranged units, which is a common tactic used by professional players in ranged versus melee skirmishes in RTS games. The evolved neural network also generalized well to other starting positions and numbers of units. We then move to a more natural two-objective fitness evaluation that maximizes damage done and minimizes damage received and show that we can use NEAT to evolve a variety of high-performance micro tactics along the Pareto front. Finally, we show that we can generate even more robust micro tactics by evolving against the best performing networks trained using normal evolution. We believe these results indicate that our neuroevolutionary approach can generate effective coordinated distributed control for agents in RTS games and in other distributed control environments.

Acknowledgements

I would like to offer my sincere thanks to my advisor, Dr. Sushil J. Louis. Without his support, this thesis would not have been possible. He has always provided me with insightful suggestions and guidance throughout the journey of my Masters' education.

I would also like to thank my other committee members, Dr. David Feil-Seifer and Dr. Binod Guragai for giving me valuable feedback and support.

Lastly, thanks to all of my friends and family who have been a constant support in my life.

This work was supported by grant number N00014 − 17 − 1 − 2558 from the Office of Naval Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research. Dr. Louis was supported in part by the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center (http://inspire-utc.mst.edu) at Missouri University of Science and Technology. The views, opinions, findings and conclusions reflected in this publication are solely those of the authors and do not represent the official policy or position of the USDOT/OST-R, or any State or other entity.

Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction **1**

1.1 Thesis Organization 5

2 Related Work **8**

3 Background **12**

3.1 Genetic Algorithms 12

3.2 Artificial Neural Networks 15

3.3 Neuroevolution and NEAT 17

3.4 Multi-objective GA and NSGAII 19

3.5 RTS games and Starcraft 21

4 Single objective Neuroevolution **24**

4.1 Methodology 24

4.1.1 Neural Network Representation 24

Input Representation 26

Output representation 28
4.1.2 Experimental setup 28
Map configurations 31
Fitness Function 31
4.2 Results and discussion 33
4.2.1 Simulation Results 33
4.2.2 Starcraft II Results 38
4.2.3 Comparison of Results 41

5 Multiobjective Neuroevoluton for Heterogeneous Group 44
5.1 Methodology 44
Network representation 44
5.1.1 Experimental setup 46
Map configurations 47
Fitness Function 48
5.2 Results and discussion 49
5.2.1 Training 50
5.2.2 Testing 52
5.2.3 Manual Coevolution 53

6 Conclusion and Future Work 60

Bibliography 62

List of Tables

4.1 Neural Network Inputs 27
4.2 Properties of Vultures, Zealots and Hellions 30
4.3 NEAT Evolution parameters 33

5.1 Neural Network Inputs for multiunit groups 45
5.2 Properties of Marine, Marauder, Medivac, Stalkers and Zealots . . . 46
5.3 Multiobjective NEAT Evolution parameters 50
5.4 Comparison of robustness between best individuals from normal evolution and Manual Coevolution (MCOE) runs. 55

List of Figures

3.1 Canonical genetic algorithm 13
3.2 An artificial neuron 15
3.3 Pareto fronts shown in green and red where green is a higher ranking front as it dominates the red front. 20

4.1 Spatial representation of inputs and outputs used in the neural network. Blue circle is the unit being controlled, red circles are enemy units, green circles are friendly units, black circle is the range of controlled unit and green and red arrows are desired displacement in each axis. 26
4.2 Architecture Diagram showing information flow and relation between each component in the experiment. 30
4.3 Map configurations for friendly (green) and enemy (red) positions used for training 35
4.4 NSC2: Remaining Vultures corresponding to increasing Zealot numbers. Vultures' performance decreases smoothly as the number of Zealots increases and never goes below two, a good indication of the robustness of evolved network. 37

4.5 NSC2: Remaining Zealots corresponding to increasing Zealot numbers. Zealots are completely destroyed by Vultures till the number of starting Zealots rises above 13. The number of remaining Zealots never rise above six, a good indicator of micro quality and robustness. 38

4.6 SC2: Remaining Hellions corresponding to increasing Zealot numbers. Number of remaining hellions show a gradual decrease which is expected as the Hellions get overwhelmed by the increasing number of Zealots. 41

4.7 SC2: Remaining Zealots corresponding to increasing Zealot numbers. Number of remaining Zealots for each scenario gradually increases in previously unseen scenarios which indicates generalizability of evolved networks. 42

5.1 Map configurations for friendly (green) and enemy (red) positions used for training 47

5.2 Pareto front progress for 2 Type vs 2 Type over $1, 5, 10, 15$ and 20^{th} generations (Average of 10 runs). Good progress is shown in the damage done axis compared to the 1^{st} generation as we see that the 20^{th} generation had individuals which can destroy all their opponents. 51

5.3 Pareto fronts for 3 Type vs 3 Type over $1, 5, 10, 15$ and 20^{th} generations (Average of 10 runs). We see the progress from first to the 20^{th} generation and notice that there is less variation between different generations, signifying that this configuration was easy to optimize for our network 52

5.4 2 Type vs 2 Type testing results on 100 randomly generated scenarios. Our evolved network generally kills all the opponents and in the worst case deals 50% of the maximum damage. 53

5.5 3 Type vs 3 Type testing results on 100 randomly generated scenarios. Here we see that the majority of points lie in the region greater than 50% but the remaining health lie below the 50% mark, which can be attributed to increase in group complexity. 54

5.6 2Types vs 2Types manual coevolution Pareto fronts for 1^{st} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best from normal evolution. 55

5.7 2Types vs 2Types manual coevolution Pareto fronts for 2^{nd} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best form 1^{st} MCOE iteration. 56

5.8 2Types vs 2Types manual coevolution Pareto fronts for 3^{rd} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best form 2^{nd} MCOE iteration. 57

5.9 3Types vs 3Types manual coevolution Pareto fronts for 1^{st} iteration showing significant improvement of the Pareto front in the damage done objective when manually coevolving against the best from normal evolution. 58

5.10 3Types vs 3Types manual coevolution Pareto fronts for 2^{nd} iteration. Here we see that this iteration of manual coevolution shows minuscule improvement over the best individual form 1^{st} MCOE iteration. 58

5.11 3Types vs 3Types manual coevolution Pareto fronts for 3^{rd} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best form 2^{nd} MCOE iteration. 59

Chapter 1

Introduction

Real Time Strategy (RTS) games are a genre of multi-player video games where players take actions concurrently and the underlying game world dynamically changes over time. The overarching objective of the game is to establish a position capable of defending against and destroying opponents. Actions in the game can be largely divided into two modes: "macro" and "micro". Macro management relates to long term strategic decisions and is concerned with resource gathering, spending those resources on research, deciding on the type and number of units to build, and in building those units. Micro management is concerned with quick and short term tactical control of units usually during a skirmish between a group of friendly units against an opponent's group. RTS game environments are a partially observable and imperfect information environment due to a restricted view through the camera on a part of the whole map and a "fog of war" which hides information form parts that have not been explored. Players have to control numbers of units ranging from tens to hundreds while simultaneously moving the camera around, deciding which units or unit factories to build, selecting units, scouting,

and exploring. The state space of typical RTS games like Starcraft is estimated to be more than $10^{50^{36000}}$ using a conservative branching factor of 10^{50} for each frame in a 25 minute game [1]. Consequently, RTS games provide a challenging platform for testing machine learning approaches [2].

This thesis focuses on generating artificial agents capable of good micro control in RTS games. Micro requires quick decision making and fast successive actions to control both movement and attack commands for units in a group. There are multiple unit types, each with its own advantages and disadvantages. Each unit has unique, well-defined characteristics regarding their capabilities, like weaponry, range, speed, maneuverability and others. Good micro can be a deciding factor in a skirmish between two groups with similar characteristics and the player has to consider the attributes of both friendly and enemy units to choose an effective tactic for the particular scenario. The complexity of the different ways in which any unit group can be controlled is as a result challenging, particularly since directives have to be provided in quick reactive time-frames. Multiple unit type leads to different optimal strategy in skirmishes for groups depending on factors like types of units in one's group, types of units in opposing group and also on the quantity of different unit types in each group. Research on RTS games and especially micro has various applications to distributed control in robotics, games and in scenarios like mining and agriculture where multiple agents are distributed but work towards a common goal.

RTS games have been used as an environment for AI research and various approaches towards automation of different aspects of RTS game playing have

been explored [1]. Approaches like reinforcement learning, scripting, and search, among others, have been used with the end goal of creating a fully automated, human-comparable RTS player [3]. Previous work has explored using Genetic Algorithms (GA) to search for an optimal combination of parameters, which are then used in Potential Fields (PF) and Influence Maps (IM) equations to control the tactical actions of skirmishing units [4]. Our research builds on this previous work in RTS game AI, but takes a different approach. Rather then having a set of parameterized control algorithms, or potential fields, for controlling movement, we explore the feasibility of evolving a neural network to perform good micro. In particular, we explore using Neuro-Evolution of Augmented Topologies (NEAT) [5] to evolve neural networks to effectively and autonomously control units for skirmishes in RTS games. NEAT evolves both the structure and connection weights of a neural network by utilizing genetic algorithm principles [6] and applying them to a population whose chromosomes represent different instances of networks being explored.

We devised a new representation which maps the game state to neural network inputs and then uses the output from the network as commands for units in the game. The neural network being evolved by NEAT is provided with information about relative position of all units in the map and the particular unit's internal state and type information. In a skirmish, there are two factors that determine the effectiveness of the outcome - damage dealt to the opponent and damage received from the opponent. Both aspects cannot be optimized independently of each other

because of their interconnected nature, merging them into one objective is sub-optimal. It is preferred to be able to choose agents on the basis of tactics which priorities any of the two different domains or balances both. Thus, we created a new multiobjective formulation for groups of multiple types of units which considered both increasing damage done and decreasing damage received. We combine NSGAII (one of the multiobjective GA approaches) with NEAT so that the evolution is guided based on multiobjective fitness of an individual. We also co-evolved units against evolved individuals to generate more robust individuals in a process which we call manual coevolution. We used our faster simulation of RTS game for quick research iterations and for hyperparameter tuning, which we applied to Starcraft II evolution.

Results on an underlying RTS-physics implementing simulation (named NSC2), show that NEAT can evolve networks for micro control of ranged units against a group of melee units. The evolved network generated kiting behaviour for ranged units (copied from vultures in Starcraft) which allowed five vultures to eliminate twenty-five zealots (a strong melee unit) without suffering any damage. Our results indicate that evolved networks generalize well to different starting configuration and varying numbers of vultures. We then moved to Starcraft II (SCII) game [7] and were able to show that NEAT can evolve good micro on a simple, flat Starcraft II map with no obstacles. We again see that hellions learn to kite and on average destroy close to 90% of zealots. [1] As before, the NEAT network generalized to different numbers of zealots and to different starting locations. We

[1] Like vultures in Starcraft, hellions are also relatively fragile, longer ranged, and fast Starcraft II units.

also show that our extended multiobjective approach controlling a heterogeneous group of units can generalize to different numbers of units in each group, and is robust against different random starting positions. With manual cooevolution, we show that the best individuals of later iterations performed better than the ones from previous iteration.

The major contribution of our work is a new neural network representation which allows for generalizable and robust micro control in RTS games. Our representation compactly represents a large amount of information in the RTS game state such that a modest neural network can be trained without requiring huge resources for training. We also combined a multiobjective fitness with NEAT and applied it to the control of multiple types of units in each group. We show that multiobjective neuroevolution, with our representation, can be used to evolve coordinated behaviour which is useful in distributed control of heterogeneous agents in multiple domains.

1.1 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 describes approaches related to our current work. We show that RTS is a popular research environment with different approaches used to tackle different aspects of the game. We see that both search and machine learning techniques are heavily used, and have been shown to work well. We also show that genetic algorithms and neural networks have been used in similar domains.

Chapter 3 details approaches which are the basis of this thesis. We give an overview of genetic algorithms and how it is different from conventional search techniques. We detail the working of artificial neural networks and how neuroevolution and NEAT combine genetic algorithm and neural network to create a particularly interesting approach to solve machine learning problems. We also show how multiobjective formulation of genetic algorithms work and how they are more powerful than the single objective formulation. Lastly we give an overview of RTS games and glimpse the complexity and variety in those games and why they are a great environment for research.

Chapter 4 describes our neural network representation, evolution configurations and experimental setup for single objective approach. We detail the simulation configuration and our fitness formulation used for evolution. We discuss the results for both NSC2 simulation and Starcraft II and compare the findings. This research was published in the 2018 IEEE Conference on Computational Intelligence in Games(CIG) [8].

Chapter 5 describes our multiobjective approach and results. We have experimented with groups consisting of two and three types of units skirmishing against another identical group. We detail our formulation of a fitness function to maximize damage done and minimize damage received, and how we incorporate the NSGAII with the NEAT approach to create a multiobjective NEAT formulation. We also describe our manual coevolution framework and then describe the results. The research described in this chapter was accepted for publication in the 2019 Genetic and Evolutionary Computation Conference (GECCO) [9] and in the

2019 IEEE Congress on Evolutionary Computation [10]. Lastly, in Chapter 6 we draw our conclusions and explain possible future approaches which can be used to extended this work.

Chapter 2

Related Work

Significant work has been done over the years in the field of designing effective RTS AI players using different techniques [1]. Buckland *et al.* described a rule based approach in his book [11] and Rabin and Steve [12] explained scripted agents which is a general approach used by bots that play in Starcraft AI ladder matches. Weber and Mateas [13] explored using data mining on gameplay logs to predict the opponent's strategy. A tree based search approach was used by Churchill, Saffidine and Buro who utilized transition tables to generate trees of actions and performed alpha beta pruning to create agents for eight vs eight unit skirmish [14]. While these approaches are concerned with rule based agents which require manual scripting to a degree, our approach learns a strategy purely based on input and output representation. Rule based approaches are harder to generalize to new scenarios, but it can be easier to explain their actions because of their explicit nature.

Others have also tried reinforcement Learning: Wender and Watson [15] used

Q learning and Sarsa to develop a fight or retreat agent. Shantia, Begue and Wiering [16] applied reinforcement learning on neural networks by using neural-fitted and online versions of the Sarsa Algorithm where they implemented a state space representation similar to [15]. Deepmind recently demonstrated an agent named AlphaStar which can beat pro-gamers in a game of Starcraft II [17]. They applied deep reinforcement learning with population based learning and imitation learning in a Starcraft II environment to train long-term short-term (LSTM) [18] neural networks using game-play data from expert players and self-play. Currently AlphaStar is limited to one map and plays only one race against the same race but exhibits professional human level micro within those constraints. These approaches are similar to our work such that they try to optimize the internal model based on some fitness given an input and output representation. AlpaStar, which uses reinforcement learning, uses fixed models (which can be quite large) as their control mechanism. Our approach uses a dynamic model which starts minimally and can grow to match the complexity of the problem, which mitigates the problem of designing a model to match the problem at hand.

Potential Fields (PFs), which have been widely used for robot navigation and obstacle avoidance [21][22] [19], have also been used for micro. Hagelback and Johansson [20] presented a multi-agent potential fields based bot architecture for the RTS game ORTS [21] which incorporated PFs into their unit AI. More recent work has focused on combining PFs with Influence Maps (IM) to represent unit and terrain information. In this context, an influence map is a grid superimposed on the virtual world where each cell is assigned a value by an IM function, which is

used by an AI to determine desired actions [22]. PFs together with IMs have been shown to be good at RTS micro, but require scripted agents with tuning parameters in case of meta search [4] and in case of pure potential fields, require larger numbers of parameters with increasing number of unit types [23]. Our approach requires no manual scripting and also makes extending to multiple units easy with a small increase in input representation.

NEAT has been applied to dynamic control tasks like double pole balancing without velocity information [5] where it could evolve a robust control policy. It has also been applied to evolving walking gaits for virtual creatures [24] and steering control for driving agents [25] [26]. NEAT has also been applied to evolve video game playing agents for games like Ms. Pac-Man [27] and Tetris [28] and has been shown to be applicable to general Atari gameplaying [29]. Board games like 2048 [30] and Go [31] have also been shown to be within reach.

NEAT and its realtime variant rtNEAT have been used to tackle different aspects of RTS games. Olesen *et al.* [32] used NEAT and rtNEAT to control the macro aspects of the game to match the difficulty of the opponent. Gabriel *et al.* [33] used rtNEAT to evolve a multi-agent system for Brood war agents based on ontological templates, where they show that their hierarchical method could be used to evolve good micromanagement tactics. RtNEAT for RTS micro control was applied in [34] where the network controlled activation of hardcoded flight or flee response. These application of NEAT to RTS require some kind of hard coded behaviour written by humans, which is then activated by an evolved NEAT network. In our case, we map the game state to network inputs and use the outputs

to control individual units directly without any scripts or intermediary mapping required.

Multiobjective evolutionary approach was also used with pure potential fields approach to find diverse strategies for RTS environments [23]. Coevolution was used by Avery and Louis in [35] to develop micro behaviours by coevolving influence maps for team tactics, and in [36] where they cooevolved influence map trees(IMT) and showed that evolved IMTs displayed similar behaviours to hand coded strategies. Here, we combine multiobjective approach with NEAT and also use manual coevolution to increase the robustness of evolved strategies.

In the next chapter, we describe the background for our work.

Chapter 3

Background

This chapter starts with an overview of Genetic Algorithms and Neural Networks, then we explain Neuroevolution and particularly NEAT, then we describe Multi-objective evolution and lastly we touch on RTS games and how they are interesting as a research platform.

3.1 Genetic Algorithms

Genetic Algorithms are a class of search algorithms inspired by natural selection, first introduced by John Holland in 1970 [6]. GAs can be categorized as a heuristic search algorithm that try to model natural selection and thus guide a population of prospective solutions toward a desired goal. A canonical framework for a genetic algorithm is given in Figure 3.1.

A canonical GA starts with a population of prospective individuals with a problem solution encoded in its representation, also called a chromosome. In every iteration, each individual's fitness is evaluated using a fitness function that gauges

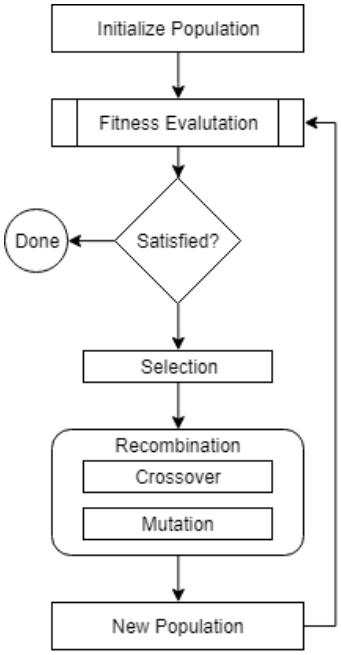

FIGURE 3.1: Canonical genetic algorithm

the quality of a solution it represents. Then, for next iteration, also known as next generation, a new batch of individuals are generated using selection and various recombination methods like crossover and mutation. Selection is used to choose individuals from previous generation based on their fitness and with some random chance. Crossover is performed between the selected parents by combining their chromosomes based on a particular crossover method and new off-spring are created that inherit some aspects of their parents. Mutation randomly changes a particular unit value in a chromosome. Selection and recombination is repeatedly performed till we get a new batch of individuals for the next generation. This process is repeated for multiple generations till we reach a satisfactory solution. Here, we see that GA is an iterative, non deterministic and parallel approach as it takes

multiple iteration to reach a solution, incorporates some randomness, and uses a population of candidate solutions to search for the solution in parallel.

The chromosome representation is hand crafted and encodes all the aspects of the solution we want to optimize or search. There can be different kinds of representation like: binary representation where all values are encoded as bits, real valued representation where values are represented as real numbers or a network representation where a chromosome is a graph structure that encodes a neural network which can solve a problem. In our research, we use this network representation which encodes a neural network. Crossover and mutation operations are directly affected by the representation of the chromosome, as crossover recombines parts of chromosome to generate new individuals and mutation changes a unit of representation. Crossover helps to recombine useful findings from different individuals and mutation introduces new variety.

We can tune the parameters of a GA to favour exploration over exploitation or vice-versa during an evolutionary run. Selection mechanisms define exploitation while crossover and mutation define exploration. Genetic Algorithms are generally used for optimization, machine learning and search, and although they are not guaranteed to find an optimum solution, they can be easily applied to complex and hard to define problems.

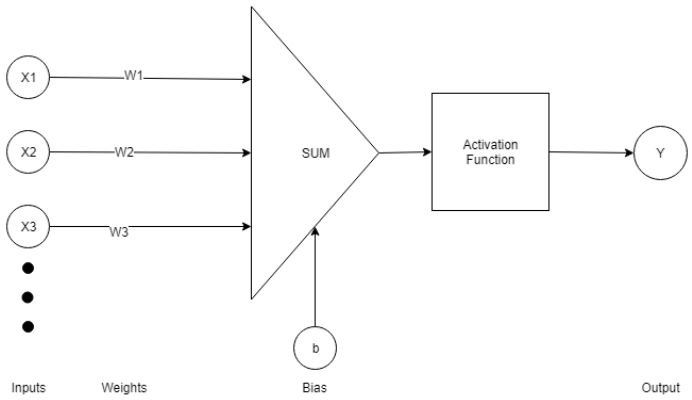

FIGURE 3.2: An artificial neuron

3.2 Artificial Neural Networks

Artificial Neural networks (ANN) are a class of machine learning models which are inspired by biological neurons. McCulloh and Pitts proposed a mathematical simplification of neural processing in biological brains in 1943 in their seminal work "A logical calculus of the ideas immanent in nervous activity" [37]. A single unit of ANN is called a neuron which takes in various inputs, scales those individual inputs independently with some weights and adds them before mapping it with a non linear function which is called an activation function. A neuron is represented diagrammatically in 3.2. A fixed input without a weight multiplier can also be added as a bias.

The output of a neuron can be represented mathematically as Equation 3.1. Where y is the output, f is an activation function, x_i are inputs values and w_i are corresponding weights for n inputs. This neuron can represent any linear function and thus can be used to classify a linearly separable problem. To extend this concept to non linear problems, we use multiple neurons interconnected in a network,

which is called an Artificial Neural Network.

$$y = f(\sum_{i=0}^{n} w_i x_i) \tag{3.1}$$

Artificial neural networks(ANN) have interconnected neurons that are separated into three types - inputs, outputs and hidden nodes. Inputs nodes are sensor nodes that take in values from outside the system, output nodes are the nodes that produce the answers from the network and hidden nodes are those nodes which lie in the information propagation path of the neural network. Each node is activated based on the outputs of the nodes from which it has incoming connections. The universal approximation theorem for ANNs [38], states that a sufficiently large and finite network with a single hidden layer can approximate any real valued continuous function. Thus, we see that multi-layer ANNs can be quite flexible.

ANNs can be trained on a given problem using variety of methods. Backpropagation [39] is one of such ways where errors are propagated back from the output layer using gradient descent. Backpropagation is a supervised technique, which requires signals to determine errors, which are then used to make changes to the network. We used neuroevolution, which is another method to get an ANN which can solve a problem. Neuroevolution and especially the NEAT algorithm is described in the next section.

3.3 Neuroevolution and NEAT

Neuroevolution is a method of training neural networks using evolutionary algorithms where the parameters of the network are tuned in each iteration of evolution. Neuroevolution methods can be differentiated based on whether they evolve structure and weights or just the weights. For the later kind, network topology is pre-defined, and just the weights are tuned. One example of the class of neuroevolutionary algorithms that evolve both topology and Weights is Neuroevolution of Augmenting Topologies (NEAT) which was devised by Stanley and Miikkulainen in 2002 [5].

NEAT is a robust algorithm for evolving neural networks based on genetic algorithm principles. NEAT attributes its robustness to three aspects, specifically that it starts complexifing from minimal structure, it leverages speciation and it uses historical markings in the genome for crossover and speciation [5]. NEAT allows for continuous complexification by allowing crossover together with a number of different kinds of mutations which allows for good network variation [5]. Examples of NEAT mutations are:

- slight perturbation of connection weight
- large change of connection weight
- addition or deletion of nodes
- addition or deletion of connections
- enable or disable connections

Starting from minimal complexity keeps the algorithm from searching a bigger space which is not strictly required to solve a particular problem and speciation protects innovation by disallowing competition between different viable strategies. Additionally, fitness sharing is also done to keep one species from dominating the whole population. NEAT's chromosome representation has two structures to store the network: one for nodes and another for connections. Finally, historical marking is a novel idea introduced by the authors that solves the problem of competing conventions, where networks may model very similar function and still have very different representations in a neural network representation. For instance, two networks, while still computing exactly the same function, may have their hidden nodes appear in a different order and thus are represented by different chromosomes, making them incompatible for crossover.

With historical marking, each new structural addition is marked with a new sequentially increasing identification marker. As a result, NEAT knows when a new innovation was introduced in an individual. Historical markings are also used for speciation and for mating between network representations. To calculate the distance for speciation, we calculate the total mismatch between the genomes of two individuals based on their innovation numbers and weights. And to mate two individuals, the genomes are lined up such that genes with the same innovation number are organized together and the genes from the fitter parent are copied to the child.

NEAT evolves a neural network from inputs and outputs specified by the problem domain. By using NEAT, we can also do away with the process of having to

encode the problem domain into a genome; but the focus instead shifts towards designing the representation for inputs and outputs of the network. For NEAT to be able to evolve good networks, we need to carefully decide on the representation used by the neural network , as a good representation makes it possible for NEAT to accurately model the problem domain.

3.4 Multi-objective GA and NSGAII

Multiobjective (MO) problems are a class of problems which have more than one fitness criteria, and such that, one criteria cannot be optimized independently of another. Combining multiple fitness into a single optimization criteria is non trivial, and is very hard to get right. It is considerably easy to tackle such problems each of the objectives can be considered separately during optimization, which a MO approach allows. Non-dominance and Pareto fronts are key concepts in any MO approach: when a particular solution has better fitness for all objectives compared to another solution, the former is a non-dominated individual which dominates the later. A set of all such non-dominated individuals form a Pareto front. Figure 3.3 shows a depiction of multiple Pareto fronts made up of non-dominated individuals. Here, the green front is better in every way compared to the red front as the individuals in green front dominate all individuals in the red front. We also see that Pareto fronts can be useful to select individuals from variety of tactics based on their position in a Pareto front. An aggressive agent deals more damage but cares less about the damage it is receives, a fleeing agent values health compared to damage, and a balanced agent falls between one of those extremes.

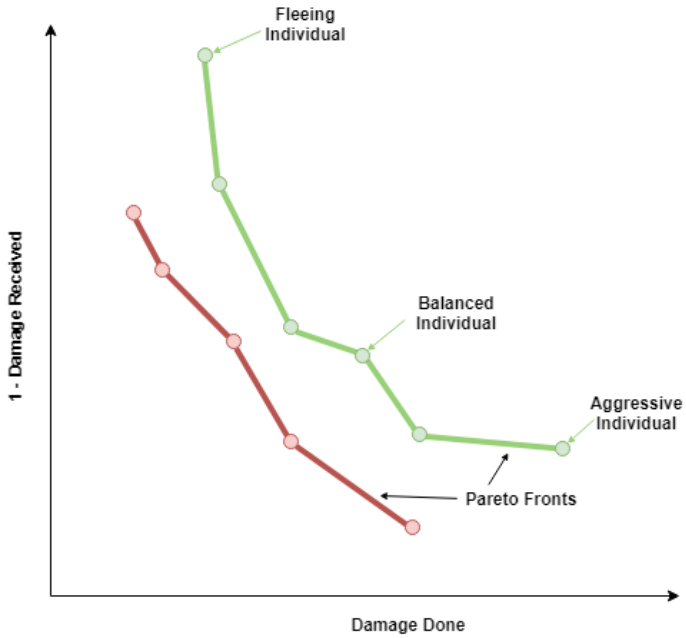

FIGURE 3.3: Pareto fronts shown in green and red where green is a higher ranking front as it dominates the red front.

Genetic Algorithms discussed till now have used a single objective fitness function that they optimize. When a problem is multi objective like ours, we can employ a multiobjective GA to get a variety of solutions which span the whole Pareto front of possibilities. Then, we can choose a particular solution based on whether we want to prioritize one objective over another, or want a balanced solution which does best on all objectives. The Nondominated Sorting Genetic Algorithm II (NSGA) is one of the multiobjective evolutionary algorithm (MOEA) which can be used to optimize problems with more than one objective.

The NSGA algorithm has been shown to find a better spread of solutions and better convergence to the true Pareto-optimal front compared to other MOEA approaches [40]. NSGA uses Pareto dominance and the position of individuals in

a Pareto front to order the population into a ranked set of individuals. First, all nondominated Pareto fronts are calculated based on the fitness of individuals, then those individuals in better fronts are given higher rank than those in the next dominated Pareto front. Individuals are selected based on their rank to produce next generation of individuals after mutation and crossover. If required, intra-front ranking can also be done based on the crowding distance, which is the measure of similarity based on the clustering of individuals on the front. This is used to promote diverse individuals in next generation.

3.5 RTS games and Starcraft

Real Time Strategy (RTS) games are a genre of computer games which can be played by multiple players at once. The game comprises of numerous game-play features that makes it highly dynamic and attracts a lot of players and is also popular as an e-sport. The most popular format is where two players play against each other; where they have to mine for resources and use them in judicious ways to create armed offensive units or mining units to generate more resources. The objective is to destroy all units and structures of the opponent. The real-time and fast paced nature of the game requires significant skill to play competitively. A player has to keep track of resources and mine for more while concurrently planning for attack and defense by coordinating player units. A fog of war limits knowledge of game state and obscures areas of the map which have not been visited, thus resulting in a partially observable game. Players have to control a number of units ranging from tens to hundreds while simultaneously moving the camera around,

deciding which units or unit factories to build and where, selecting units, scouting, and exploring the map. These decisions can be largely divided into two categories: "macro" and "micro". Macro management relates to long term strategic decisions and is concerned with resource gathering, spending those resources on research, deciding on the type and number of units to build, and in building those units. Micro management is concerned with quick and short term tactical control of units usually during a skirmish between a group of friendly units against an opponent's group.

A group can consist of multiple types of units, each with their own advantages and disadvantages. Each unit has unique, well-defined characteristics regarding their capabilities, like weaponry, range, speed, maneuverability and others. Good micro can be a deciding factor in a skirmish between two groups with similar characteristics and the player has to consider the attributes of both friendly and enemy units to choose an effective tactic for the particular scenario. The complexity of the different ways in which any unit group can be controlled is as a result challenging, particularly since directives have to be provided in quick reactive time-frames. Multiple unit type leads to different optimal strategy in skirmishes for groups depending on factors like types of units in one's group, types of units in opposing group and also on the quantity of different kinds of unit in each group.

Starcraft II is a popular RTS game played around the world, and has a good selection of races and units with different styles and capabilities. Choice of a particular race lends to a particular game style and compounded with multitude of unit combination, every match-up is different. Given such variety, Blizzard, the

maker of the game has put considerable effort into the balance of the game units to maintain the strategic value of the gameplay. And with the introduction of Starcraft II API [7], Starcraft II lends itself very well to automation, which makes it perfect for research and have been used by many as a platform for RTS research [2].

Chapter 4

Single objective Neuroevolution

There are different aspects of micro game-play that can be controlled for an entity, such as movement, whether to attack, when to flee, and other such unit specific actions. Controlling all aspects of a micro engagement is therefore a complex endeavor. In this chapter, we focus on entity movement and firing control for ranged units and try to evolve networks with good tactics which are effective against melee units. We next describe the neural network representation for NEAT, and the experimental setup used for evolution.

4.1 Methodology

4.1.1 Neural Network Representation

In this research, we have implemented a neural network representation that can directly control entity movement and the decision to fire or move based on the spatial position of other entities in the map and our entity's current state. With this approach, the movement decision is solely taken by the network without any

hard-coded maneuvering behaviours. This makes it possible for NEAT to come up with new control strategies that are not limited to hard-coded scripts. Our representation is entity localized, meaning, each entity uses the same evolved neural network and calculates inputs and outputs based on their current location. Individual inputs and outputs are described below.

Our neural network inputs can be divided into two classes according to the type of information they represent. The first class deals with spatial information and describes the relative position of all entities on the map. In order to be able to represent units consistently and uniformly, regardless of the number, we followed an approach whereby we divide the visible world into regions relative to the current position of the unit being controlled. Figure 4.1 shows the spatial information being fed into the neural network.

In our representation, the world around the entity is divided into 4 quadrants with the entity at the center. The four quadrants are further divided into eight regions separated by the attack range of the unit as shown by the labels r_1 to r_8 in Figure 4.1. Each region then corresponds to four inputs in the network as listed below.

1. the number of enemy units
2. average distance of enemy units
3. number of friendly units and
4. average distance of friendly units in the region

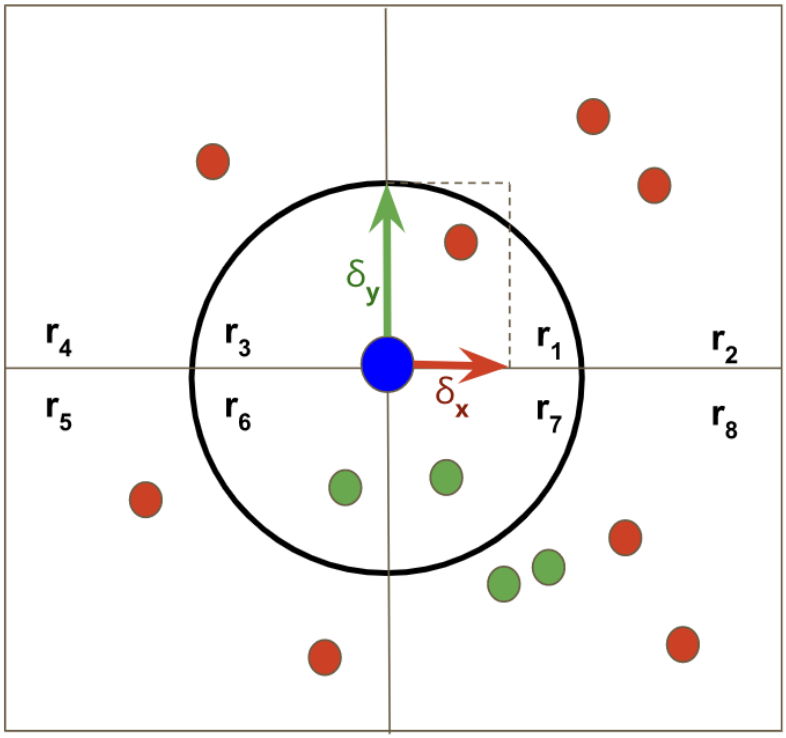

FIGURE 4.1: Spatial representation of inputs and outputs used in the neural network. Blue circle is the unit being controlled, red circles are enemy units, green circles are friendly units, black circle is the range of controlled unit and green and red arrows are desired displacement in each axis.

Input Representation

Next, we have four inputs indicating map boundaries. These inputs provide distance from the entity to north, south, east and west boundaries correspondingly. The second class of inputs feeds the entity with some of the entity's own internal state. The internal features that we have considered are:-

1. the current health

2. weapon cool-down

3. current fire or move state and

4. a recurrent input which is the previous attack/move output from the network

In total, the neural network that controls the movement of friendly units in our environment has 40 inputs, 16 that are used to represent the position of all friendly units, 16 that are used to represent the position of enemy units, 4 boundary sensors and 4 inputs for the internal state as shown in table 4.1. The representation is constructed such that it can capture essential information from different map configurations and number of entities without having to vary the number of inputs in the network. Once computed, all inputs are scaled between 0 to 1 by representing each value as percentage of a maximum possible value for that input.

TABLE 4.1: Neural Network Inputs

id	Type
1-8	enemy avg position per region
9-16	friendly avg position per region
17-24	enemy units per region
25-32	friendly units per region
33-36	boundary sensors for 4 directions
37	self cooldown
38	self hitpoints
39	current attack/move state
40	previous attack/move state

Output representation

Outputs of the network are represented by two scalars representing desired δx and δy coordinate displacement, and one Boolean to decide whether a unit should fire or move at that instant. δx and δy displacement output are scalar values from 0 to 1 from which we subtract 0.5 and then scale them to go the coordinate position relative to the unit's current position. This allows the output to represent any coordinate around the entity in the region corresponding to the scaling factor. The outputs are then fed into the movement mechanism of the simulation or SCII in order to generate movement.

The third output is a move or attack command which is a Boolean signal. If the output is greater than 0.5, the entity has to focus on attack and if the output is lower, the entity stops attacking and begins moving to the position signalled by δx and δy displacement output. To attack, the entity focuses on the nearest opponent in range and fires its main weapon if the weapon is not in cooldown.

4.1.2 Experimental setup

NEAT evolves the network across generations based on the fitness of the network. To evaluate the fitness of the neural network, we used two different environments: the Starcraft II game and NSC2 simulation of the Starcraft environment which is tailored to capture the micro combat aspects of Starcraft and can be run without graphics for significant speedup. Although our NSC2 simulation used for experimentation and to explore input and output representations runs fast, the simulated

physics and entity properties cannot be made identical to SCII easily as the mechanisms used by SCII are not open. This means, the micro evolved in our simulation may not transfer well to SCII. In addition, there are differences in the properties of vultures in our simulation, vultures in Starcraft Brood Wars, and hellions in SCII. However, our simulation runs much faster and we can experimentally try multiple representations, input configurations, and NEAT parameters far more quickly than when using the SCII API. We can then start long evolutionary runs within the SCII environment with more confidence.

For our experiments, we choose the zealot as a representative melee unit and either the vulture or hellion as a representative ranged unit. More specifically, for our simulation environment, we copied zealot and vulture properties from the Starcraft BW API [41]. When running in SCII, zealots and hellions use SCII properties. Vultures/hellions and zealots deal comparable damage in each attack but have different attack range and movement speeds. Table 4.2 shows the properties of the units considered for this thesis. When micro'd well, Hellions and vultures can be strong against zealots because of their greater speed and attack range, which makes it possible for a small number of vultures/hellions to kite a bigger group of zealots to death. We expect our approach to evolve good tactical control that can exploit this strength of vultures/hellions against zealots.

Figure 4.2 shows the three main components of our experimental setup; the NEAT evolution module, the evaluation adapter and the game itself, which could either be SCII or NSC2. The evolution module is concerned with running the evolution by assimilating the fitness results and generating networks. We used the

TABLE 4.2: Properties of Vultures, Zealots and Hellions

Property	Vulture	Hellion	Zealot
Hit-points	80	90	100
Damage	20	13	16
Attack Range	5	5	0.1
Speed	4.96	5.95	3.15
Cool-down	1.26	1.78	0.857

SharpNeat implementation of NEAT by Colin Green [42] for the evolution module and adapted it for our purpose. An evaluation adapter is the mediator between evolutionary algorithm and the game which we implemented using sockets to be able to run the game simulations in parallel. It gets the configuration from the NEAT module and sets up the game, it also gets a neural network configuration from the evolution module and feeds inputs with the current game state into the network and uses the output from the network to feed the game and move entities. The adapter returns the final fitness after running the simulation which ends when one of the players has no units left or after a set number of frames.

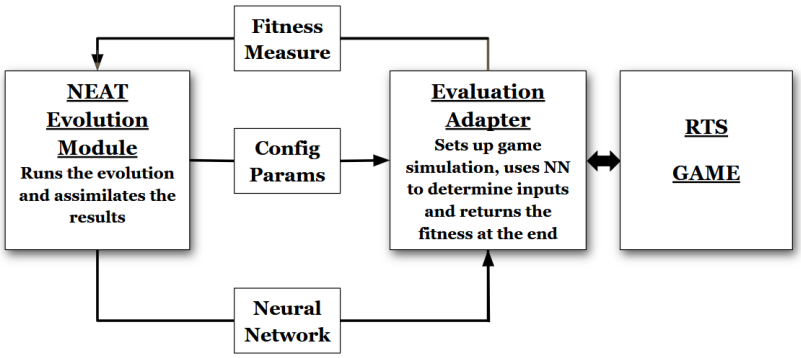

FIGURE 4.2: Architecture Diagram showing information flow and relation between each component in the experiment.

Map configurations

We choose five different unit spawn configurations that determine entity starting positions. These are diagonal, side by side, surround, surrounded and random. In diagonal, opposing sides spawn in groups diagonally on a square map. Similarly, in the side by side configuration, entities appear along the same y-coordinate separated by a fixed distance of 10% of the map size. In surrounded, vultures or hellions appear at the center in a group surrounded by number of zealots and the opposite is true for surround - hellions or vultures surround zealots. We also experimented on random spawning locations for all units of both players. We kept the number of vultures or hellions constant at five but randomly varied the number for zealots for different configurations. Number of units for different map configurations are arbitrary but, few ranged units are pitted against large number of melee units as it forces the ranged units to develop a strategy to win. Constant number of ranged In the rest of the thesis, we mean vultures or hellions when we use the term ranged units.

Fitness Function

Genetic Algorithms require each individual in a population to be assigned a fitness value which they can work towards optimizing. Fitness function is used to quantify desirability of each individual and should give higher fitness to more desirable solutions. Here, we are using the single objective formulation of genetic algorithms which require a single fitness value to be assigned to each individual. Thus, we used a fitness function that considers both the damage received and the

damage dealt by our evolving ranged units (Vultures and Hellions). Over evolutionary time, fitness gradually grows as the ranged units get better at damaging zealots and at evading attacks. At the end of each game run, we sum the remaining hitpoints for both zealots (Hz) and ranged units (Hh) and subtract the remaining hitpoints of zealots from the remaining hitpoints of the ranged units. We add the maximum hitpoints for all zealot units so that the fitness function is always positive.

For number of starting zealots Nz, remaining zealots Rz, remaining hellions Rh, and maximum hitpoint of zealot $Hzmax$, fitness F is calculated as:

$$F = \sum_{n=1}^{Nz} Hzmax_n + \sum_{n=1}^{Rh} Hh_n - \sum_{n=1}^{Rz} Hz_n$$

We should note that as the hit points of both zealots and ranged units are similar, with increasing numbers of zealots and low numbers of ranged units, this fitness function leans towards giving more weight to damage done than damage received. This could be better balanced in various ways. For example, by multiplying the sum of hitpoints of hellions by a balancing factor. Nevertheless, we found that the current configuration performed well during our experimentation phase.

4.2 Results and discussion

We experimented with two different RTS game environments: the Starcraft II game, and our simulation NSC2. Below, we describe our experiments and analyze the results for each.

4.2.1 Simulation Results

TABLE 4.3: NEAT Evolution parameters

Property	Simulation	Starcraft II
Population	50	50
Generations	100	100
Species	5	5
Initial Conn Probability	0.2	0.1
Elitism Proportion	0.2	0.2
Selection Proportion	0.2	0.2
Asexual Offspring Proportion	0.5	0.8
Sexual Offspring Proportion	0.5	0.2
Inter-species Mating	0.01	0.01
Connection Weight Range	5	7
Probability Weight Mutation	0.95	0.95
Probability Add Node	0.01	0.02
Probability Add Connection	0.025	0.04
Probability Delete Connection	0.025	0.025

In our first set of experiments using NSC2, we evolved vultures against a larger group of zealots. Zealots in our simulation, use a hand coded AI which controls each unit as follows: pursue the nearest vulture and attack when it is in range. Both zealots and vultures were given complete map vision, i.e. there was no fog of

war and thus they did not have to explore the map and could start pursuing their enemy right away. Note that this is a significant difference from SCII.

We ran NEAT on a population size of 50 individuals for 100 generations. The following results are average of 10 different runs of a complete evolutionary epoch, started with different random seeds. Various parameters that we used for NEAT evolution are noted in table 4.3. Each genome was evaluated based on a complete run of a test configuration, which consisted of 10 different spawning locations with different number of zealots and vultures. We sum the fitness for each of the 10 different training configurations to get the final fitness, which is then forwarded to the NEAT module. We run each scenario until one of the player loses all his units or for a maximum number of frames. We had the option to run NSC2 without the graphics rendering which significantly decreased running time compared to SCII.

Initially, we tried to evolve agents only based on a single test configuration of the map, but results showed that they did not generalize well to new scenarios. Evolving with only one configuration, results in much less robust networks whose performance might jump form high to low or low to high when changing the number of zealots even by as little as one zealot. In one case removing a single zealot significantly **decreased** vulture performance. This variability is also shown by other neural network based approaches for Starcraft AI [34]. We found that we must provide a number of different training configurations in order to evolve robustness. Using the sum of different fitness from different configurations led to good generalization across different map configurations and different numbers of units. The 10 different test cases are a sample from the the possible configuration

space of different number of zealots and 5 different starting configuration. The training scenarios are listed below and are depicted in Figure 4.3.

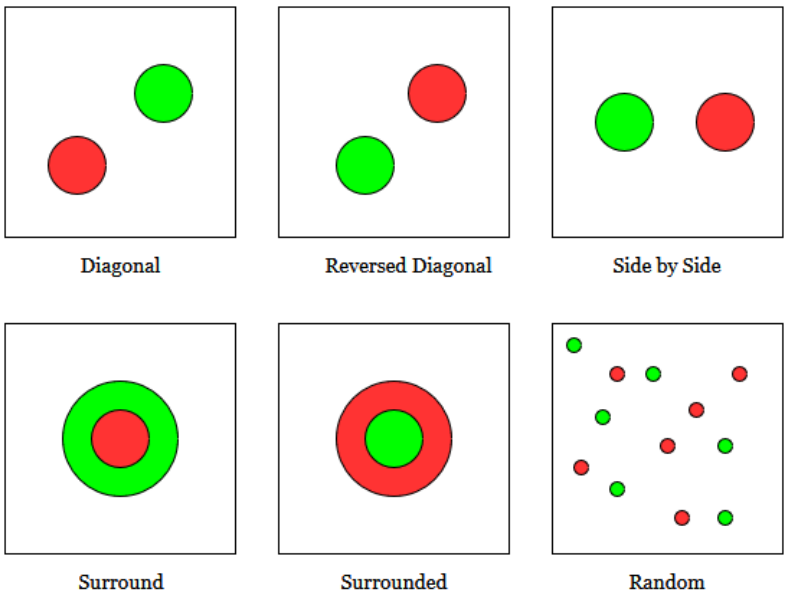

FIGURE 4.3: Map configurations for friendly (green) and enemy (red) positions used for training

1. Diagonal, 25 zealots
2. Reversed Diagonal, 20 zealots
3. Side by Side, 10 zealots
4. Reversed Side by Side, 15 zealots
5. Surround, 20 zealots
6. Surround, 10 zealots

7. Surrounded, 20 zealots
8. Surrounded, 25 zealots
9. Random, 15 zealots
10. Random, 25 zealots

In NSC2, the average number of generations needed to find the best individual was 80 and the average best fitness was 96% of the maximum fitness possible. We found that the the evolved vultures learned kiting or to hit and run, against the group of zealots. Kiting is an effective micro strategy that is used by speedy ranged units against slower melee units, where the ranged units fire, run out of range, turn back, fire, and run back out of range again and again avoiding damage to themselves while damaging the enemy.

After evolving neural networks to control vultures with kiting ability against groups of zealots, we tested for the generalizability of our result against scenarios that the Vultures did not encounter during the training phase. For each possible starting configuration, we varied the number of starting zealots from 1 to 30 while the number of Vultures was always constant at 5. Here, we note that Vultures were only evolved against the group of 10, 15, 20 and 25 zealots thus, their performance against different number of zealots shows the robustness of the evolved network.

Results of our generalizability tests are shown in Figures 4.4 and 4.5. The vertical axis represents the number of units remaining at the end of each game run and the horizontal axis represents the number of zealots against which the five vultures

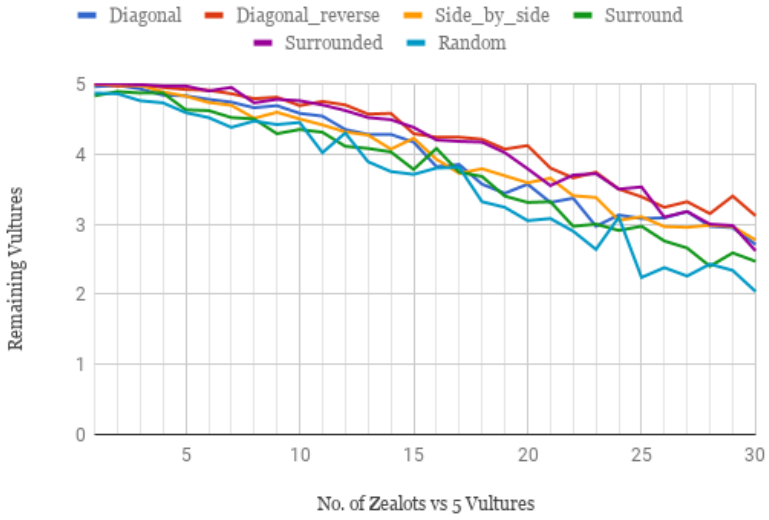

FIGURE 4.4: NSC2: Remaining Vultures corresponding to increasing Zealot numbers. Vultures' performance decreases smoothly as the number of Zealots increases and never goes below two, a good indication of the robustness of evolved network.

skirmish. We present the results for 6 different starting positions, each starting position averaged over ten runs. Figure 4.4 shows the number of vultures remaining when times runs out while Figure 4.5 shows the number of zealots remaining. In Figure 4.4, we see that vultures' performance decreases smoothly as the number of zealots increases. The number of vultures never goes below two, a good indication of the robustness of evolved networks.

Generalization with respect to damage done is shown in figure 4.5 where we note that the zealots are completely destroyed by vultures till the number of starting zealots rises above 13. The number of surviving zealots then gradually increases across all our scenarios. The number of remaining zealots never rise above six, another good indicator of micro quality and robustness. As the number of

Zealots increase, the Vultures start getting swamped and start taking more damage and are unable to kill all the Zealots for large number of Zealots. But we note that trend lines for all starting scenario depict similar gradual slope signifying the robustness against different configurations. Our NEAT agent kiting 5 Vultures against 25 Zealots in NSC2 for different map configurations is shown in Video 5 at https://www.cse.unr.edu/~aavaas/Micro.html.

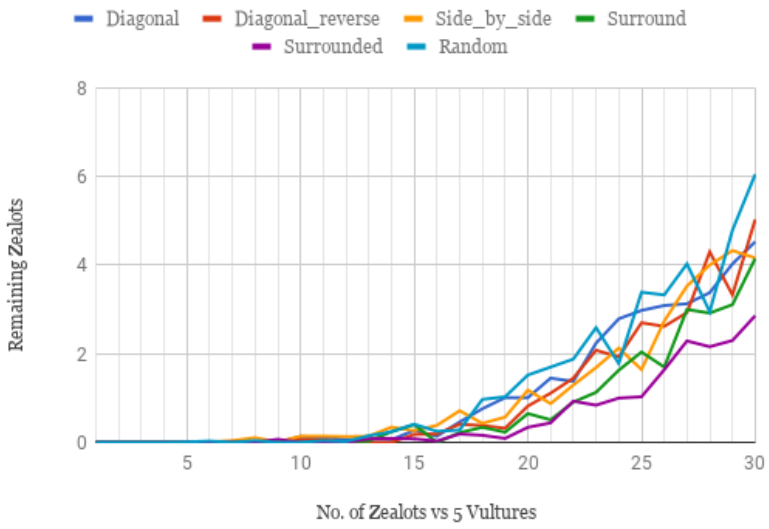

FIGURE 4.5: NSC2: Remaining Zealots corresponding to increasing Zealot numbers. Zealots are completely destroyed by Vultures till the number of starting Zealots rises above 13. The number of remaining Zealots never rise above six, a good indicator of micro quality and robustness.

4.2.2 Starcraft II Results

In Starcraft II, we evolved hellions which are ranged units that can do splash damage against zealots which are strong melee units. We control the movement and

attack commands for the hellions after translating the outputs from NEAT evolved neural networks to the commands for Starcraft II using the API. We ran the game at the top speed of 16. As Starcraft II runs relatively slow even at top speed, we ran on 15 machines for 24 hours.

For Starcraft II experiments, we ran on a population size of 50 for 100 generations and the results averaged over 10 runs for the final results. We use the same NEAT parameters as for our simulation as given in Table 4.3. Unlike our simulation, we only used the three different configurations to get the total fitness. The three configurations and corresponding number of zealots are listed below; the number of hellions is always five.

1. Diagonal, 25 zealots
2. Random, 20 zealots
3. Side by Side, 15 zealots

Over 10 runs, the average number of generations needed to find the best individual was 85 and the average best fitness was 88% of the maximum fitness possible. The evolved network also displayed kiting behaviour against the zealots - similar to our findings from the simulation approach. Videos of our agent controlling Hellions against Zealots in multiple scenarios and performing kiting in SC II is depicted in Videos 1 - 4 at https://www.cse.unr.edu/~aavaas/Micro.html. We see that for lower number of Zealots, our network kills all of them without losing any friendly units, and could still save most of the friendly units with considerable increase in the Zealot count. We also see that our units have evolved to

get out of corners and handle being at the map boundary, which are undesirable for match-up like ours as Hellions are more vulnerable to surround attack in such situations.

We tested for the generalizability of the evolved networks in similar fashion to the tests for the simulation environment. That is, we tested the best evolved network against new configurations and with varying number of zealots. Here, we note that hellions only evolved against groups of 15, 20 and 25 zealots, and on only three configurations. Generalization results are shown in Figure 4.6 and 4.7. The vertical axis represents the number of units remaining at the end of each game run and the horizontal axis represents the number of zealots remaining when skirmishing with five hellions. We present the results for six different starting positions with the number of zealots varying from 1 to 30. We ran the simulation for five runs on the same starting configuration to get the average number of remaining units.

As shown in figure 4.6, we see a downward trend for the number of remaining hellions starting from 5. However, unlike in our simulation, the trends are different for different starting configurations. Diagonal and side by side show good performance across different numbers of zealots while random and surrounded perform comparatively lower. The gradual decrease is expected as the hellions get overwhelmed by the increasing number of zealots. Still, we can see from the graph that hellions are generalizing well with respect to damage received against different number of zealots and different starting positions.

Generalization with respect to damage done is shown in figure 4.7 where we

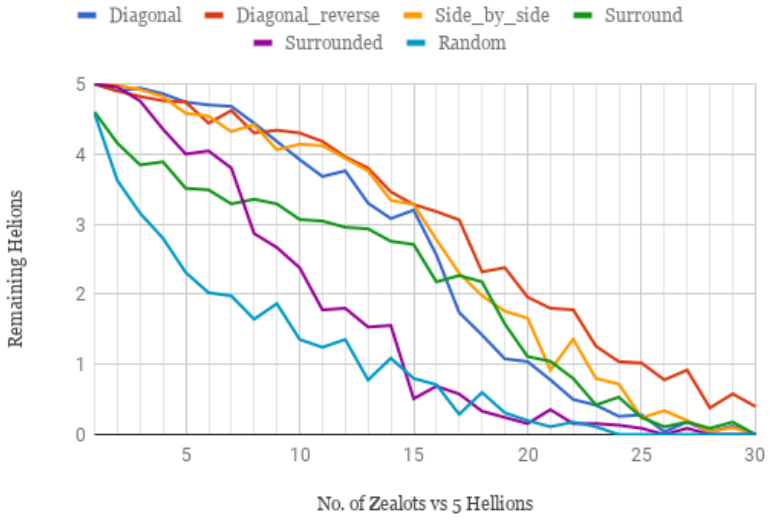

FIGURE 4.6: SC2: Remaining Hellions corresponding to increasing Zealot numbers. Number of remaining hellions show a gradual decrease which is expected as the Hellions get overwhelmed by the increasing number of Zealots.

again note that the hellions perform well for diagonal and side by side scenarios while performing comparatively lower in random and surrounded scenarios. The number of remaining zealots for each scenario gradually increases in these previously unseen scenarios, and indicates generalizability of our result.

4.2.3 Comparison of Results

We have shown that the NEAT generated neural networks from from Starcraft II and our simulation were able to generalize with respect to different starting positions and different numbers of zealots. Ranged units performed well against

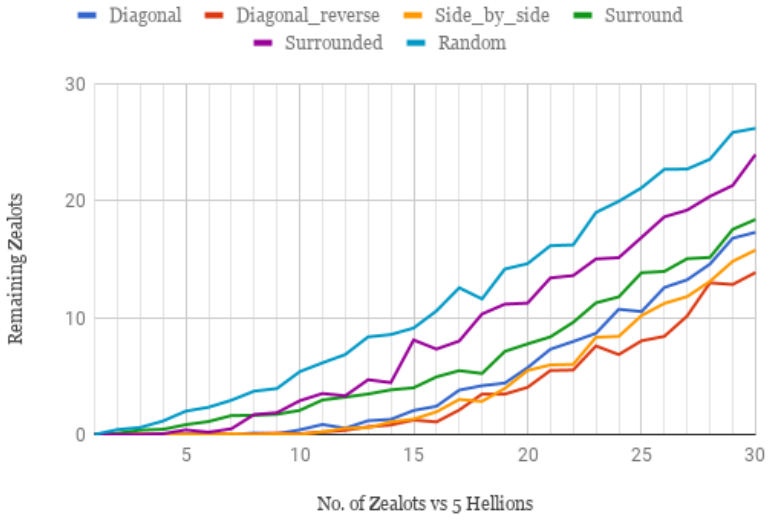

FIGURE 4.7: SC2: Remaining Zealots corresponding to increasing Zealot numbers. Number of remaining Zealots for each scenario gradually increases in previously unseen scenarios which indicates generalizability of evolved networks.

smaller numbers of zealots and performance decreased with increase in number of zealots for both environments. The gradual progression of values for different series without major deviance indicates that the evolved network is robust against changes in both starting position and number of zealots. Videos at https://www.cse.unr.edu/~aavaas/Micro.html serve well to indicate the quality and robustness of the evolved micro.

NEAT was able to evolve kiting behaviour in both our simulation and in the Starcraft II environments but there are some differences between the results from two environments. The evolved network in Starcraft II seemed to perform comparatively worse than our simulation. We believe fewer training configurations,

increased complexity of SCII, and differences in AI that we evolved against, account for these differences.

Till now, we have been evolving against a single objective fitness function that combines damage done and damage received factors of the skirmish into a single function. As those entities are co-dependent, it is hard to conjure a function that combines these qualities into one in the best possible way. In the next chapter, we see how multiobjective evolution solves this problem and extend our approach to scenarios with more than one type of units in a group.

Chapter 5

Multiobjective Neuroevoluton for Heterogeneous Group

In this chapter, we extend the approach used in Chapter 4, and evolve micro tactics for groups with more than one kind of units. We also employ multiobjective evolution and manual coevolution to generate effective tactics. We focus on coordinated entity movement and firing control which cover the most complex aspect of the micro game-play. We next describe the neural network representation, and the experimental setup used for this approach.

5.1 Methodology

Network representation

We employ representation similar to the representation used in previously in Chapter 4, where the space around a unit is divided and the positional information of

other entities are aggregated. We extend the representation such that a single neural network can incorporate tactics for more than one unit in group with heterogeneous units.

In this new representation, unit type is signalled by a set of inputs using a one hot representation to denote a particular type. This translates to n inputs required to signify unit types in a groups consisting of n different units. Thus, in total the neural network that controls the movement of friendly units in our environment with groups of n types has $40 + n$ inputs, 16 that are used to represent the position of all friendly units, 16 that are used to represent the position of enemy units, Four boundary sensors. Four inputs for the internal state and n for unit type as shown in Table 5.1. The representation is constructed such that it can capture essential information from different map configurations and number of entities. Once computed, all inputs are scaled between $0to1$ by representing each value as percentage of a maximum possible value for that input.

TABLE 5.1: Neural Network Inputs for multiunit groups

id	Type
1-8	enemy avg position per region
9-16	friendly avg position per region
17-24	enemy units per region
25-32	friendly units per region
33-36	boundary sensors for 4 directions
37	self cooldown
38	self hitpoints
39	current attack/move state
40	previous attack/move state
n	unit type

5.1.1 Experimental setup

As we are concerned with groups of different types of units, we selected units with unique abilities so that the group could benefit from the composition. In this thesis, we have considered scenarios with two different types of units and three different types of units in a group. For groups with two types of units, we selected Zealots and Stalkers from the Protoss race. The Zealot is a strong melee unit and the Stalker is a quicker ranged unit. For groups with three types in a group, we selected Marine, Marauder and Medivac from the Terran Race. Each unit differs in their abilities, range, speed, damage capacity and life points. We used different number of units of each type in a group, but keep the composition of opposing group the same as that of the friendly group. Table 5.2 shows the properties of the units considered in this thesis. We expect our approach to evolve good tactical control that can exploit the strength of each unit types in a group.

TABLE 5.2: Properties of Marine, Marauder, Medivac, Stalkers and Zealots

Property	Marine	Marauder	Medivac	Stalker	Zealot
Hit-points	45	125	150	80	100
Damage	6	10	0	13	16
Range	5	6	4	6	0.1
Speed	3.15	3.15	43.15	4.13	3.15
Cool-down	0.61	1.07	-	1.54	0.86

Similar to single objective NEAT, Multiobjective NEAT evolves a network based on fitness of the network, and in this case two values - damage done and damage received. To evaluate the fitness of the neural network, we used the Starcraft II game environment. Our experimental setup was similar to our single objective

setup and had three main components, the NEAT evolution module, the evaluation adapter and the game as depicted in the Figure 4.2.

Map configurations

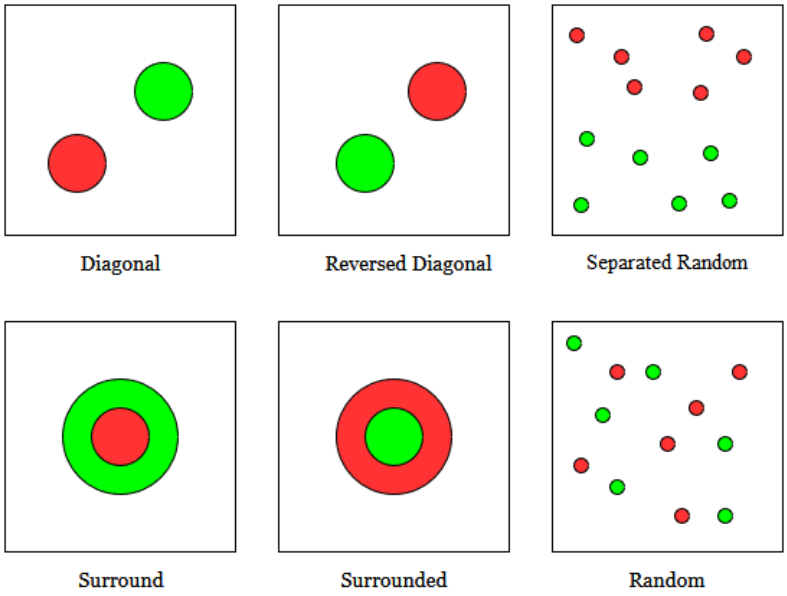

FIGURE 5.1: Map configurations for friendly (green) and enemy (red) positions used for training

We used the Starcraft II map editor to create custom maps to train our agent, and use the SCII API to generate units at a particular spawn location. We train on different unit spawn configurations with different starting positions to keep the agent from over-fitting to a specific scenario. The configurations used are diagonal, side by side, surround, surrounded and random. In diagonal, opposing sides spawn in groups diagonally on a square map. Similarly, in the side by side configuration, entities appear along the same y-coordinate separated by a fixed distance of

50% of the map size. In surrounded, friendly units appear at the center in a group surrounded by enemy units and the opposite is true for surround. Separated random had two opposing groups separated into two halves of the map and within those halves, their location was randomized. We also trained on a map with random spawning locations for all units of both players. The graphical representation of training scenarios are depicted in Figure 5.1

Fitness Function

We used a multiobjective fitness function with two objectives; one being minimizing damage received and the other maximizing damage dealt by our evolved gents. In general we want the fitness to gradually grow as evolved units get better at dealing damage and at evading attacks. At the end of each game run, we sum the remaining hitpoints for all the units in the group and subtract it from the sum of maximum hitpoints of all the individual units in the group to determine the damage received by the group, which is also the damage dealt by the opponent. We formulate both fitness such that NSGA acts to maximize both of them, and scales them such that they lie in the [0..1] range. To minimize damage received, we reformulate the fitness criterion such that we maximize 1-damage received. Equation 5.1 and 5.2 describe the two objective formulation optimized by our multiobjective evolutionary approach.

$$Max \sum_{i=0}^{e} D_{ei} = Max \sum_{i=0}^{e} \frac{MaxHealth - EH_i}{MaxHealth} \tag{5.1}$$

$$Max \sum_{f=0}^{f} (1 - D_{fj}) = Max \sum_{j=0}^{f} \frac{FH_j}{MaxHealth} \tag{5.2}$$

$$MaxHealth = Max \sum_{k=0}^{n} UnitMaxHealth_k \tag{5.3}$$

$MaxHealth$ is a constant calculated using equation 5.3, n represents total units at the beginning of each skirmish, and $unitMaxHealth_k$ represents max health of k^{th} unit as per given in Table 4.2. In equation 5.1 and 5.2, e, f, D_{ei} and D_{fj} represent the number of enemy, number of friendly units, damage done to i^{th} enemy unit and damage done to j^{th} friendly unit alive at the end of skirmish respectively. EH_i and FH_j represent the remaining health of i^{th} enemy unit and j^{th} friendly unit.

5.2 Results and discussion

We ran Multiobjective NEAT on a population size of 20 individuals for 20 generations. The following results are averaged over 10 different runs of a complete evolutionary epoch started with different random seeds. NEAT evolution parameters that were used for this experiment are noted in table 5.3. Each genome was evaluated based on a complete run of a test configuration, which consisted of two different random configurations and sums the corresponding objectives for each of the testing configurations to get the final fitness. We turn off the fog of war and run each scenario until one of the players loses all units or for a maximum number of frames. We describe our experiments and analyze the results below.

TABLE 5.3: Multiobjective NEAT Evolution parameters

Property	Starcraft II
Population	20
Generations	20
Initial Connection Probability	0.1
Elitism Proportion	0.2
Connection Weight Range	7
Probability Weight Mutation	0.95
Probability Add Node	0.02
Probability Add Connection	0.04
Probability Delete Connection	0.025

5.2.1 Training

Figure 5.2 and 5.3 show the Pareto fronts for two Type vs two Type scenario and three Type vs three Type scenario respectively over $1, 5, 10, 15$ and 20^{th} generations. We evolved both scenarios against the default Starcraft II AI with the same configurations of units on both sides. The Pareto fronts are generated by merging corresponding Pareto fronts for 10 evolutionary runs with different random seeds but the same initial setup.

For skirmishes of groups containing two type of units vs two type of units, we had fifteen zealots and five stalkers in each opposing group. From Figure 5.2, we see that the fitness started with over 70% in Damage done and 30% in damage received, and improved to 100% damage done but did not show much improvement in damage received. The final Pareto front after 20 generation had strategies ranging from 100% damage done and 90% damage received to 80% damage done to 65% damage received. Out network had high fitness from the start signifying the naive nature of the default Starcraft II AI.

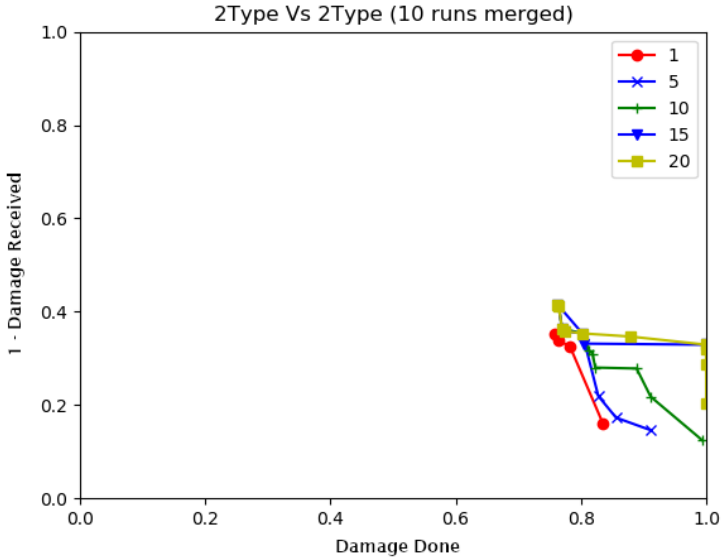

FIGURE 5.2: Pareto front progress for 2 Type vs 2 Type over $1, 5, 10, 15$ and 20^{th} generations (Average of 10 runs). Good progress is shown in the damage done axis compared to the 1^{st} generation as we see that the 20^{th} generation had individuals which can destroy all their opponents.

For groups containing three type of units vs three type of units, we had ten Marines, six Marauders, and four Medivacs in each opposing group. All of the three units were controlled by a single network, the only differentiating inputs being unit identifying inputs. We see from the Figure 5.3 that our network achieves very high fitness from the start and approaches 80% on both of the objectives by the end of 20^{th} generation. We see that there is less variation between different generations as Pareto fronts are not wide, signifying that this configuration was easy to optimize for our network. We also see that the networks reach the best results by the 5^{th} generation, further strengthening the fact that the default Starcraft II AI is naive.

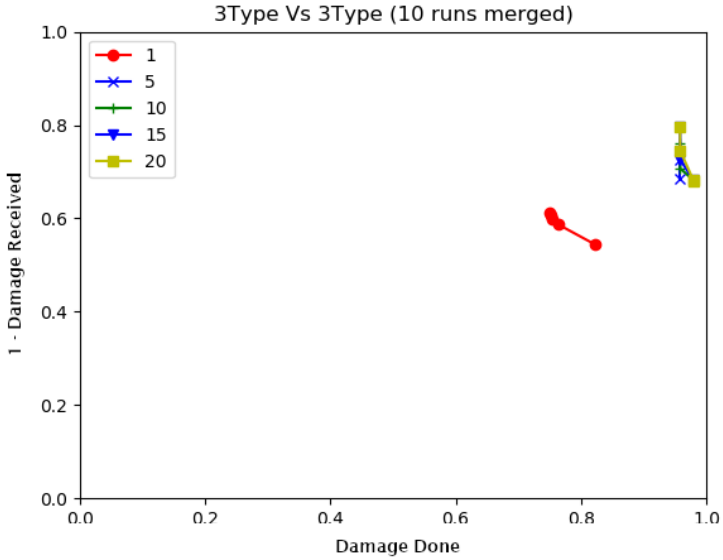

FIGURE 5.3: Pareto fronts for 3 Type vs 3 Type over $1, 5, 10, 15$ and 20^{th} generations (Average of 10 runs). We see the progress from first to the 20^{th} generation and notice that there is less variation between different generations, signifying that this configuration was easy to optimize for our network

5.2.2 Testing

We tested one of the balanced network from the best Pareto fronts against 100 random scenario to test the performance of networks in novel scenarios. We show the result of testing on two type vs two type and three Type vs three Type scenarios in Figures 5.4 and 5.5. On two type vs two type scenario, we see that majority of the fitness lie near the 100% damage done region, and majority points lie in the region greater than 50% damage done for both scenarios. This signifies that our evolved network generally kills all the opponents and even in the worst case deals 50% of the maximum damage. We can comparatively say that networks perform

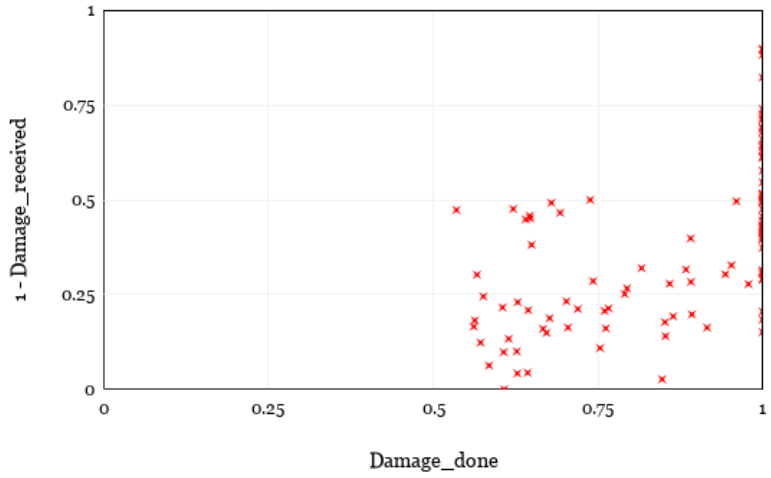

FIGURE 5.4: 2 Type vs 2 Type testing results on 100 randomly generated scenarios. Our evolved network generally kills all the opponents and in the worst case deals 50% of the maximum damage.

better in two type vs two type scenario than three Type vs three Type scenarios, as the points in two type vs two type generally lie in the far right region of the fitness graph, can also be quantitatively seen to do better on both objectives from Table 5.4. This can be attributed to the specific dynamics of different units and increase in group complexity by going from two to three types of units. In general, we can conclude that the evolved network performed well on novel random scenarios.

5.2.3 Manual Coevolution

We concluded from the above section that the default Starcraft II AI was not very strong and could be easily beaten by our evolved networks in few generations. Evolving against a naive opponent leads to sub-optimal agents that only perform well against those opponent. Thus, to generate more robust networks, we require

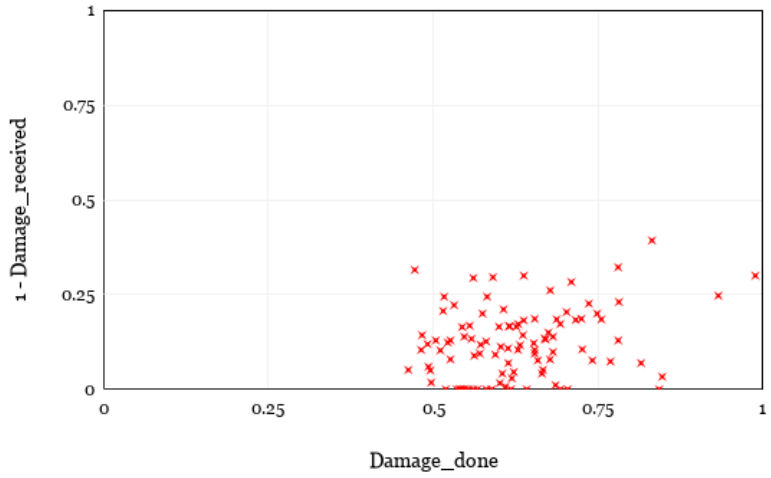

FIGURE 5.5: 3 Type vs 3 Type testing results on 100 randomly generated scenarios. Here we see that the majority of points lie in the region greater than 50% but the remaining health lie below the 50% mark, which can be attributed to increase in group complexity.

more capable networks to evolve against; and we already have a more robust opponent at our disposal. we could use the agent evolved against the default AI as the new opponent as it is proven to be better than the default AI. Thus, we utilized the concept of coevolution where players in both sides of a game are evolved, and applied this in a manual iterative fashion. We selected the best individual from the evolved Pareto front, made it our opponent, and evolved against it for the next round. We perform this manual coevolution for two more runs to obtain more robust individuals than what we would have gotten from just the first iteration of evolution against the base AI. We used same training and testing setup as normal evolution for manual coevolution to keep results comparable.

We show the results of three runs of manual coevolution in Figure 5.6, 5.7 and 5.8 for two type vs two type; and in Figure 5.9, 5.10 and 5.11 for three type vs

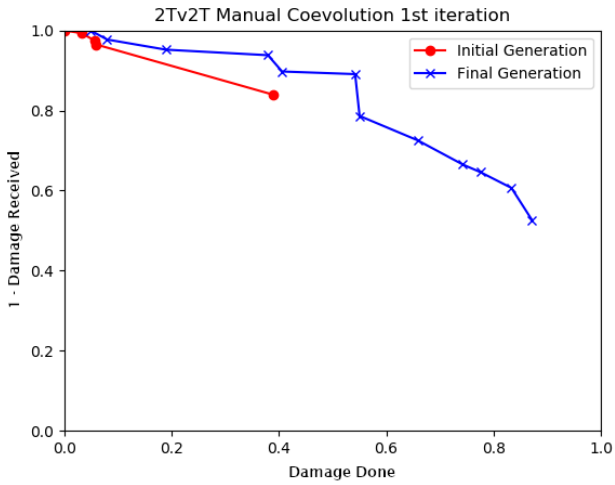

FIGURE 5.6: 2Types vs 2Types manual coevolution Pareto fronts for 1^{st} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best from normal evolution.

TABLE 5.4: Comparison of robustness between best individuals from normal evolution and Manual Coevolution (MCOE) runs.

Method	Damage Done		1 - Damage Received	
	2v2	3v3	2v2	3v3
Normal	0.857966	0.629189	0.376345	0.122628
MCOE run-1	0.859106	0.640296	0.381967	0.124016
MCOE run-2	0.867881	0.691048	0.382371	0.250223
MCOE run-3	0.871826	0.710296	0.386840	0.280662

three type correspondingly. In each iteration shown in figures, we see that the first and last Pareto fronts show considerable improvement. The red line shows the first generation Pareto front and blue line shows the Pareto front in the last generation. As each iteration is evolved against the best individual from the last iteration, the progress of Pareto front shows that we have more robust individuals than the previous iteration.

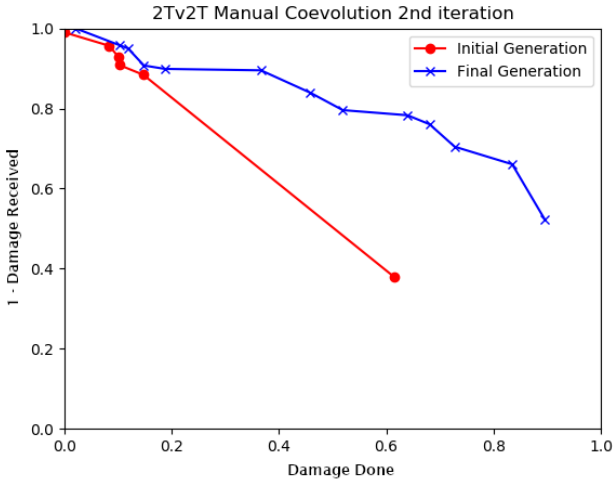

FIGURE 5.7: 2Types vs 2Types manual coevolution Pareto fronts for 2^{nd} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best form 1^{st} MCOE iteration.

We show the testing results for normal and various runs of manual coevolution in Table 5.4 for comparison. We show values for two objectives for both two vs two and three vs three scenarios on random scenarios. We see that the fitness values for all objectives increase monotonically from normal evolution to 3^{rd} run of manual coevolution, which reinforces the result from the Pareto graphs. We see greater improvement in three vs three scenario for both damage done (12%) and 1-damage received (where it was more than doubled) compared to two vs two as it was already doing very well. We can conclude that the best performing network did get more robust against random testing scenario with each iteration of manual coevolution. Thus, we believe, multi-objective competitive coevolution will prove to be a viable approach to high performance cooperative control for heterogeneous groups of game entities or agents.

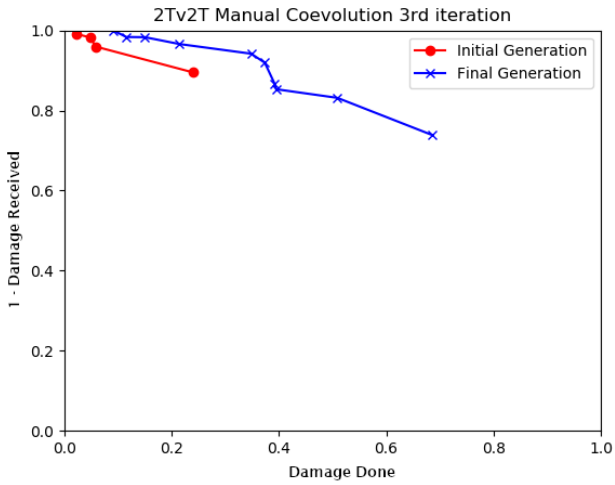

FIGURE 5.8: 2Types vs 2Types manual coevolution Pareto fronts for 3^{rd} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best form 2^{nd} MCOE iteration.

In the next chapter, we conclude the thesis and remark on some ways it can be extended in the future to evolve networks that generalize to more complex micro configurations.

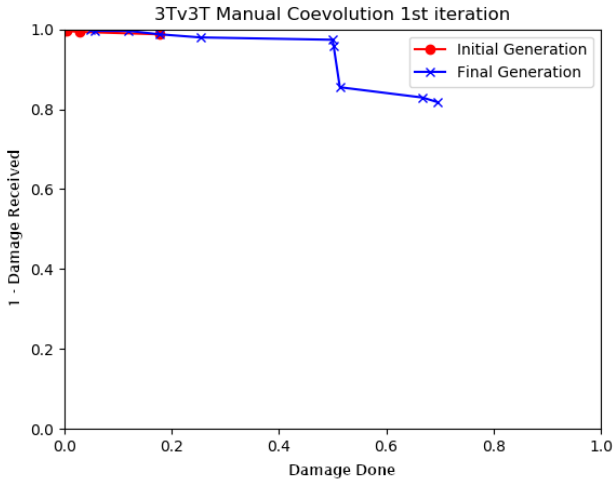

FIGURE 5.9: 3Types vs 3Types manual coevolution Pareto fronts for 1^{st} iteration showing significant improvement of the Pareto front in the damage done objective when manually coevolving against the best from normal evolution.

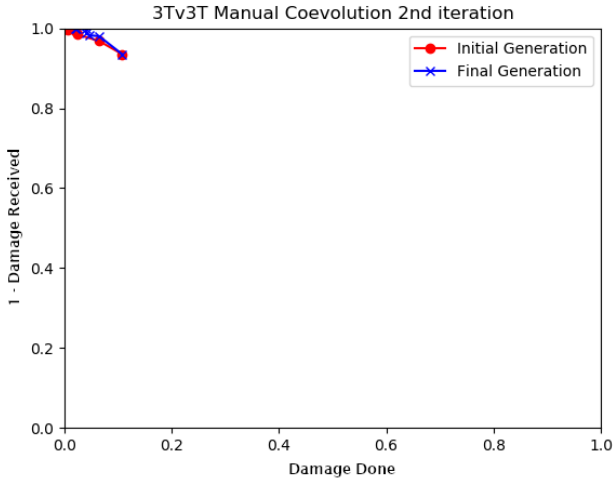

FIGURE 5.10: 3Types vs 3Types manual coevolution Pareto fronts for 2^{nd} iteration. Here we see that this iteration of manual coevolution shows minuscule improvement over the best individual form 1^{st} MCOE iteration.

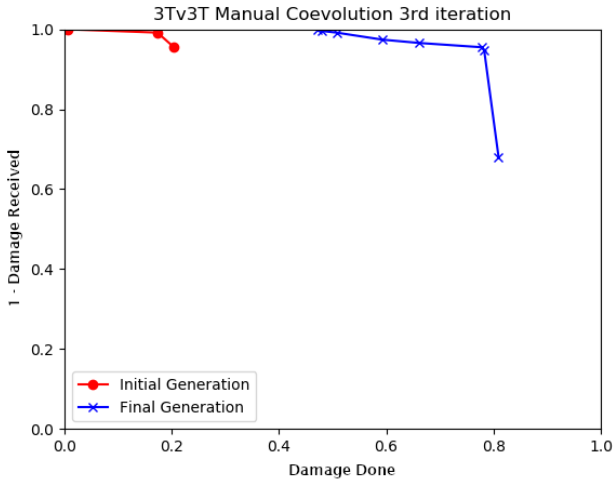

FIGURE 5.11: 3Types vs 3Types manual coevolution Pareto fronts for 3^{rd} iteration showing significant improvement of the Pareto front from the first to the last generation when manually coevolving against the best form 2^{nd} MCOE iteration.

Chapter 6

Conclusion and Future Work

Our research focused on using NEAT to evolve neural networks that could provide robust control of a squad in an RTS game skirmish. We showed that our representation of game state provided to NEAT sufficed to evolve high performing micro, and show that training on a variety of scenarios leads to more robust and high performing micro. The evolved networks generalized well to different numbers of opposing units and different starting configurations.

We used our own simulation environment (NSC2) for initial experimentation and exploration. Because our simulation runs much faster than Starcraft II, we were able to explore multiple input and output representations and different evolutionary hyperparameters to hone in good representations and parameters. We then used this experience to reproduce our results in the popular RTS game Starcraft II. Here, ranged hellions evolved kiting behaviour against melee zealots (like in our simulation environment) which is an effective strategy for ranged units against melee units.

We also applied Multiobjective Neuroevolution and manual coevolution to generate RTS micro tactics for heterogeneous groups. The evolved agents were able to control groups containing multiple type of units in a group and could easily beat the default Starcraft II AI. We further used manual coevolution using the evolved AI as an opponent to generate more robust agents. We show that the evolved networks generalized well to different number of opposing units and different starting configurations. We believe these results show that NEAT holds promise as a potential approach to evolving RTS game micro.

With a general neural network representation and with NEAT, we think that our approach can be effectively extended to approach more complex scenarios and group configurations. This work can be further expanded by considering the output of the neural network as the probability of it being active rather than simple thresholding. Using recurrent neural networks or Long Term Short Term (LSTM) networks would enable incorporating sequential strategies spanning multiple time frames. In addition, we can add more state information about the units and the environment, or we could learn the state representation directly from the raw game input.

Bibliography

[1] Santiago Ontanón et al. "A survey of real-time strategy game ai research and competition in starcraft". In: *IEEE Transactions on Computational Intelligence and AI in games* 5.4 (2013), pp. 293–311.

[2] Michael Buro. "Call for AI research in RTS games". In: *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*. 2004, pp. 139–142.

[3] Glen Robertson and Ian Watson. "A review of real-time strategy game AI". In: *AI Magazine* 35.4 (2014), pp. 75–104.

[4] Siming Liu, Sushil J Louis, and Christopher A Ballinger. "Evolving Effective Microbehaviors in Real-Time Strategy Games". In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.4 (2016), pp. 351–362.

[5] Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[6] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[7] Blizzard. *StarCraft2 Client library designed for building scripted bots and research*. 2018. URL: https://github.com/Blizzard/s2client-api.

[8] Aavaas Gajurel et al. “Neuroevolution for RTS Micro”. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018. URL: https://ieeexplore.ieee.org/document/8490457.

[9] Aavaas Gajurel and Sushil J Louis. “Multiobjective Neuroevolution for Real Time Strategy Game Micromanagement”. In: *2019 ACM Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2019.

[10] Rahul Dubey et al. “Evolutionary Multi-objective Optimization of Real-Time Strategy Micro”. In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019.

[11] Mat Buckland and Mark Collins. *AI techniques for game programming*. Premier press, 2002.

[12] Steve Rabin. *AI Game programming wisdom 4*. Vol. 4. Nelson Education, 2014.

[13] Ben G Weber and Michael Mateas. “A data mining approach to strategy prediction”. In: *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE. 2009, pp. 140–147.

[14] David Churchill, Abdallah Saffidine, and Michael Buro. “Fast Heuristic Search for RTS Game Combat Scenarios.” In: *AIIDE*. 2012, pp. 112–117.

[15] Stefan Wender and Ian Watson. “Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar”. In: *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE. 2012, pp. 402–408.

[16] Amirhosein Shantia, Eric Begue, and Marco Wiering. "Connectionist reinforcement learning for intelligent unit micro management in starcraft". In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE. 2011, pp. 1794–1801.

[17] Deepmind. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. 2019. URL: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

[18] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[19] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[20] Johan Hagelback and Stefan J Johansson. "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*. International Foundation for Autonomous Agents and Multiagent Systems. 2008, pp. 631–638.

[21] Michael Buro. "ORTS - A Free Software RTS Game Engine". In: *Accessed March* 20 (2007), p. 2007.

[22] Alberto Uriarte and Santiago Ontanón. "Kiting in RTS games using influence maps". In: *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2012.

[23] Rahul Dubey et al. “Evolutionary Multi-objective Optimization of Real-Time Strategy Micro”. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018.

[24] Brian Allen and Petros Faloutsos. “Complex networks of simple neurons for bipedal locomotion”. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE. 2009, pp. 4457–4462.

[25] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. “Evolving competitive car controllers for racing games with neuroevolution”. In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM. 2009, pp. 1179–1186.

[26] Francisco Gallego Durán et al. “Driving bots with a neuroevolved brain: Screaming Racers”. In: *INTELIGENCIA ARTIFICIAL* (2005).

[27] Jacob Schrum and Risto Miikkulainen. “Evolving multimodal behavior with modular neural networks in Ms. Pac-Man”. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 325–332.

[28] Lauren E Gillespie, Gabriela R Gonzalez, and Jacob Schrum. “Comparing Direct and Indirect Encodings Using Both Raw and Hand-Designed Features in Tetris”. In: (2017).

[29] Matthew Hausknecht et al. “A neuroevolution approach to general atari game playing”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (2014), pp. 355–366.

[30] Tuponja Boris and Šuković Goran. “Evolving neural network to play game 2048”. In: *Telecommunications Forum (TELFOR), 2016 24th*. IEEE. 2016, pp. 1–3.

[31] Kenneth O Stanley and Risto Miikkulainen. “Evolving a roving eye for go”. In: *Genetic and Evolutionary Computation Conference*. Springer. 2004, pp. 1226–1238.

[32] Jacob Kaae Olesen, Georgios N Yannakakis, and John Hallam. “Real-time challenge balance in an RTS game using rtNEAT”. In: *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE. 2008, pp. 87–94.

[33] Iuhasz Gabriel, Viorel Negru, and Daniela Zaharie. “Neuroevolution based multi-agent system with ontology based template creation for micromanagement in real-time strategy games”. In: *Information Technology and Control* 43.1 (2014), pp. 98–109.

[34] Jacky Shunjie Zhen and Ian D Watson. “Neuroevolution for Micromanagement in the Real-Time Strategy Game Starcraft: Brood War.” In: *Australasian Conference on Artificial Intelligence*. Springer. 2013, pp. 259–270.

[35] Phillipa Avery and Sushil Louis. “Coevolving influence maps for spatial team tactics in a RTS game”. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM. 2010, pp. 783–790.

[36] Chris Miles and Sushil J Louis. "Co-evolving real-time strategy game playing influence map trees with genetic algorithms". In: *Proceedings of the International Congress on Evolutionary Computation, Portland, Oregon*. IEEE Press. 2006.

[37] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[38] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[39] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[40] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[41] *Starcraft Vulture*. 2018. URL: http://liquipedia.net/starcraft/Vulture.

[42] Colin Green. *SharpNEAT*. 2018. URL: http://sharpneat.sourceforge.net/.