Exploring Learning in Evolutionary Artificial Neural Networks

Even Bruvik Frøyen

Master of Science in Computer Science
Submission date: September 2011
Supervisor: Pauline Haddow, IDI

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

Even Bruvik Frøyen

Exploring Learning in Evolutionary Artificial Neural Networks

Master Thesis, Fall 2011 Norwegian University of Science and Technology

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science
Adviser: Pauline C. Haddow

Abstract

Evolutionary artificial neural networks can adapt to new circumstances, and handle slight changes without catastrophic failure. However, under constantly changing circumstances, resulting in unpredictable grounds for evaluating success, the lack of memory of previous adaptations are a limiting factor. While further evolution can allow adaptations to new changes, the same is required for a return to a previous environment. To reduce the need for further evolution to deal with previously seen problems, this thesis looks at an approach to encourage previous knowledge to be retained across generations. It does this using back propagation in conjunction with an implementation of the HyperNEAT neuroevolutionary algorithm.

Glossary

- Ply - An action, or actions, in a two-player game that results in the other player being allowed to perform its actions. Can be considered synonymous with “turn” in games where a turn consists of actions taken by only one player.
- Lookahead - The act of exploring the possible next moves in a game to a certain depth, in an effort to choose the move most likely to result in victory.
- Rationality - In the field of artificial intelligence, always taking the course of action that results in the most beneficial results for the individual.

Acknowledgments

Thanks to Pauline Haddow for patient help during the work on this thesis, and to Lester Solbakken for taking the time to answer numerous questions about back propagation.

Even Bruvik Frøyen
Trondheim, 23rd September 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

This chapter consists of a basic introduction to this thesis, presenting the motivation behind it, and what exactly is attempted. It also presents the research method employed, and gives an overview of the rest of the thesis.

1.1 Background and Motivation

This thesis started out through an idea of an artificial intelligence (AI) for computer games, capable of learning throughout its lifetime. The basic motivation stemmed from a fascination with computer games on the part of the author, one he had nurtured for most of his life at the time of writing this.

While always finding computer games entertaining, there was a limitation that became more and more apparent as time went by, and computer games gained complexity. The AI of the opponent. A static AI, incapable of learning or evolution, is inherently *predictable*, and as such exploitable. That is to say, once you have found a method that allows you to beat the AI, you will always be able to beat the AI. Of course, it would be a trivial matter to add some level of randomization to the AI, something frequently used in games with limited complexity. But as games become more complex, an AI that behaves without concern for the final outcome of the match, becomes increasingly self destructive.

This thesis sprung from a wish to do something about this limitation. To create an AI that can adapt to changing circumstances as well as attempting completely new behavior, i.e taking previously unseen action that cannot be predicted based on an understanding of previously seen actions. An AI that can not be predictably beaten repeatedly by the same basic strategy. At the same time, it must not forget what it has previously learned, as that would mean it could be predictably beaten by two alternating strategies.

While this is undoubtedly something beneficial to the gaming industry, this was also an interesting challenge from a general AI standpoint. AIs are useful in situations of imperfect information, where it is simply not feasible to create a system capable of performing perfectly[Russel and Norwig, 2003]. This can be due to a limited understanding of the problem area, or due to the problem arena being in (constant) change (e.g in the case of driving a car, it can not be assumed that all other motorists will behave in the same way in response to the same circumstances). The latter is the situation for any computer game AI, as the opponent, the human, will probably change his approach in response to the actions of the AI. It is as a general rule not practical to create an AI taking into account every possible action a human player could take, as this would either mean that the game is so simple in its mechanics that it can easily be played perfectly (and is therefore no challenge at all), or require an inordinate amount of computing resources (and as a result becoming an insurmountable challenge). A better approach would be to create an AI that can adapt to the human player, as the human adapts to it. This situation is not limited to computer games, but can be found in many real life situations where we would like to apply artificial intelligence: driving cars, search engines, or path finding, to name but a few.

In order to achieve such a system, there are challenges to be mastered. The AI must be capable of searching across a large search space in order to answer new challenges, while at the same time not losing previously gained information.

This thesis is an attempt in this direction, not to create an AI capable of driving a car, or playing a complex computer game, but to create an AI that uses a large search space to find new ways of doing things, while at the same time retaining previous adaptations.

In the time leading up to this thesis, the author explored various technologies that could be used to achieve such an effect [Frø yen, 2010]. It was found that the best option would be to try to combine the large search space and adaptability of evolutionary techniques, with the information retaining and learning abilities found in learning in artificial neural networks.

1.2 Goals and Research Questions

The basic idea for this thesis was to attempt to create a system that allows an AI to develop continuously, adapting to new circumstances without destroying previous adaptations. Based on previous work [Frø yen, 2010], it was decided that a combination of evolutionary neural networks and learning algorithms for neural networks held potential.

A system that can transfer the information evolved in successful individuals across generations would be able to evolve new solutions in a changing environment, and relearn previous adaptations. This would be a step towards creating a system such as the one envisioned in section 1.1. To that effect, this thesis combines an evolutionary algorithm with a learning algorithm.

THE HYPOTHESIS for this thesis, was that *utilizing a form of learning between artificial neural network individuals evolved by a neuroevolutionary algorithm can counteract forgetting old adaptations, resulting in better performance, when tested on a simple game.*

Note that the main purpose of this thesis is not to attempt to create a strong AI for the game chosen as a test bed, but to attempt to explore and test the hypothesis as stated.

1.3 Research Method

A literary search was performed to explore possible technologies that could be used. A game was chosen as a test bed for the system (Section 2.1), and a possible architecture was proposed (Section 2.3). The architecture was then implemented, and experiments were performed to evaluate said architecture in reference to the research goal (Chapters 3 and 4) .

1.4 Thesis Structure

This chapter contains an introduction to the thesis, explaining the underlying motivation, hypothesis, and general approach to explore said hypothesis.

Following this is Chapter 2, Theory and Background, dedicated to the techniques used in the experimental system proposed by this thesis, the game used as a testbed, and the structure of the proposed system.

Then follows Chapter 3, Research Results, on the experiments used to evaluate said system, as well as the results of said experiments. These results are then analyzed.

A conclusion on the viability of said system, as well as of the basic techniques used to create it, is then presented in Chapter 4, Evaluation and Conclusion, followed by suggestions for future work, given the results of this thesis.

Chapter 2

Theory and Background

In this section, the basic theories and technologies utilized in the work on this thesis will be discussed. This includes the testing arena chosen, a simple yet tactically complex board game, as well as AI techniques. The choices in techniques were guided by the results in Frø yen [2010].

2.1 Choice of Test Bed Game

To test the architecture to be proposed in this thesis, a suitable game had to be chosen as a test bed. A number of selection criteria had to be kept in mind.

- The game should have fairly simple mechanics, so as not to make it too much of a challenge to create an AI capable of performing the necessary actions, or particularly challenging to simulate, drawing attention away from the subject at hand.
- The game should not be too easy to master. If it was, perfect solutions could possibly be found quickly, resulting in it becoming difficult to measure improvement. In other words, a game with simple mechanics, but with room for complex and varied strategies.
- The game should be fully observable and deterministic, and preferably zero-sum, that is one players loss is the other players win and vice versa, to allow for repeated tests to enforce results. The results should be easy to evaluate from a fitness point of view.

2.1.1 Discarded Games

A number of games were considered with reference to the criteria as stated in Section 2.1.

Tic-tac-toe was discarded out of hand. Although it is fully observable, zero sum and based on extremely simple mechanics, it is also far to easy to master. Playing a perfect game is trivial.

A simple computer game, used in a previous AI competition from The Gathering [gat, 2010], called Submerged Temple [Bondehagen, 2010] was also considered. This consideration was based on the work in Frø yen [2010]. It was however discarded as well. While it is fully observable and deterministic, it was considered to complex to be preferable. The factors involved are not difficult to follow for the human mind, but were still deemed unnecessarily complex for the purposes of this thesis.

Go was considered as an interesting challenge. However, based on the difficulties the scientific community has had with developing AI for this game [Bouzy and Cazenave, 2001], it was decided that this would probably steal focus from the goal of this thesis.

Capture Go was considered as well, and has been used with similar purpose in mind in the past [Cai et al., 2010]. However, it has quite complex rules. A similar, yet simpler games was preferable.

2.1.2 Reversi

In the end, the choice of game fell on Reversi.

Reversi (also known as Othello) is a zero-sum, fully observable, two player, sequential board game with very simple mechanics. It is deterministic, meaning that any given action under any given board state will have a perfectly predictable result. One player is black, the other white. The board is arranged into a grid of squares, usually 8x8, but the rules can be used on any board of size nxn, where n is an even number. Though the game is simple, it has potential for complex tactics. To put it a different way, it is easy to learn how to play the game, yet it is unsolved (that is, no-one has found a sequence of moves guaranteed to result in victory for either player, or guaranteed to ensure a draw) for board sizes greater than 6x6.

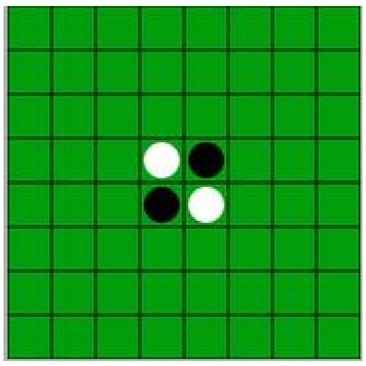


Figure 2.1: Opening Positions in Reversi

The game starts with the board in a state such as shown in Figure 2.1. The same starting formation is used on all board sizes, the four central squares are populated as the four central squares in said figure.

The player with the most pieces left at the end of the game wins, the other looses. This is achieved through capturing the opponent's pieces, which turns them into your pieces. A piece is captured by surrounding it on two sides, forming a line with the other player's pieces. That is, if the white player has a piece in square (x,y), it can be captured by black by placing pieces in squares (x-1,y) and (x+1,y), or (x,y-1) and (x,y+1), or the same formation diagonally. Having pieces in (x-1,y) and (x,y+1) will not result in a capture. It is possible to capture multiple pieces, both by lining pieces up in multiple directions, and

	x	o	o	o	

Table 2.1: Example Board State 1

	x	x	x	x	x

Table 2.2: Example Board State 2

surrounding several of the opponents pieces lined up in a straight, uninterrupted line. Only moves that result in a capture are allowed.

As an example consider a six-by-six board as shown in Table 2.1(o is used to represent white, x to represent black) .

The only valid move for black would be to place a piece to the right of the rightmost white piece, which would result in the board state shown in Table 2.2.

Given board state shown in Table 2.3, it would also be possible for black to move as shown in Table 2.4. As can be seen, the mechanics are the same both horizontally and diagonally (as well as vertically, which has not been shown here).

It is only allowed to skip a move if there are no valid moves available. Black always moves first, each player takes turns, and the game ends when both players have no valid moves left.

Reversi is interesting in and of itself as a testing ground for AIs, due to the fact that Reversi on an eight-by-eight board is still unsolved, as mentioned. This says a lot about the game's complexity. This was part of the reason for it being chosen as a testing ground in this thesis. It was believed there would be little chance of the AI quickly, or indeed at all, developing a perfect solution. This was considered a good point, as a perfect solution would be impossible to improve upon, making it impossible to document any positive

	x	o	o	o	
		o			

Table 2.3: Example Board State 3

	x	o	o	o	
		x			
			x		

Table 2.4: Example Board State 4

effect of introducing learning to evolutionary methods.

As can be seen from the above explanation of the mechanics of the game, the rules are quite simple, and the game easy to simulate on a computer. All that is needed is a representation of the board, and the ability to search in straight lines across it.

2.2 Technology

This section contains an introduction to the technologies considered and used in this thesis.

2.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) [Farley and Clark, 1954, Callan, 1999]is a bio inspired approach to AI. It takes inspiration from the basic structure of the human (or any other animal) brain. ANNs are networks of artificial neurons. A neuron takes a number of input values, runs them through a function, and passes the result on to all neurons linked to its output. See Figure 2.2. Every link will have a weight, with which values passed through the link is multiplied. Input nodes takes values from an external (to the network) source, and output nodes are used to report the response(s) the network has produced. Neurons are often organized in layers, with neurons in one layer connected only to neurons in neighboring layers. Input nodes exist in input layers, output nodes in output layers, and the rest in so called hidden layers, as they are hidden from the system defining input and receiving output.

The function applied to the values fed into a neuron (called the activation function) can be any function $f(x)$, such as a linear function, sigmoid function, threshold function, or a trigonometric function, such as sin or tanh. It is possible for a neuron to feed into itself or a neuron in a previous layer, so that the next round of activations will involve values from previous activations. This recurrent behavior results in the network being able to take previous actions into account, in other words resulting in a form of memory. Networks that feed strictly forward through the network are referred to as feed forward networks, a network as described in this paragraph as recurrent networks.

The power of ANNs stems from their ability to encode complex information in the weights of the links in the network. When presented with incomplete information, they will have a gradual drop off in performance, giving almost-right or slightly-wrong answers, rather than an outright wrong ones, as may be encountered when using techniques such as decision trees [Russel and Norwig, 2003]. In an ANN, rather than explicitly stating a solution based on a set of

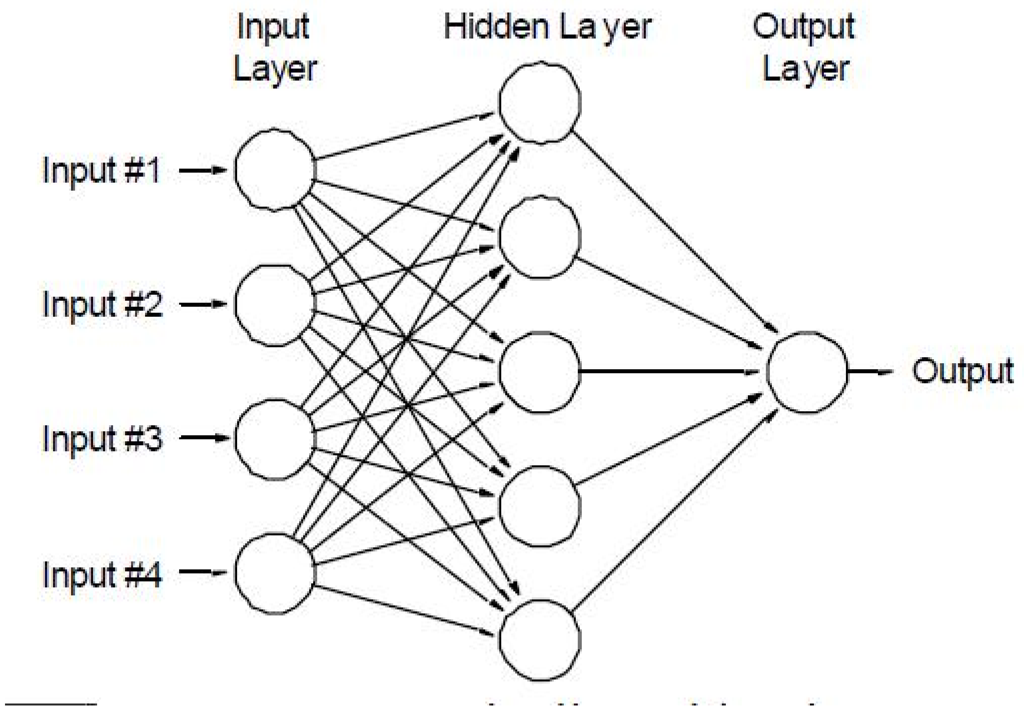


Figure 2.2: Fully Connected Feed Forward Artificial Neural Network

explicit criteria, a numerical value is returned based on the numerical input values. The ANN forms a, potentially complex, function in the form of the activation functions of the nodes and the weights of the links between the nodes, and on the input and output nodes. As such, given that the function encoded in the network is not too dissimilar from the function necessary to solve the problem, even if the return value is wrong, it may still be a close approximation to the correct answer.

There are several methods for teaching neural networks to perform specific functions. The basic approach is to give the network an example problem with a solution, and then using a function to tune the weights in the network so that the answer of the network approaches the provided correct solution. This is referred to as supervised learning. Given that the network is sufficiently large, and of a fitting shape, it should be capable of learning a function capable of at least approximating the function from which the examples used are acquired, given a sufficiently large and representative set of examples.

ANNs are believed to have great potential in the creation of computer game AIs, and have according to certain game developers seen some use [Johnson and Wiles, 2001, Charles and McGlinchey, 2004]. ANNs are a relatively well understood learning technology.

ANNs are entirely predictable in their behavior, once you know the network structure and weights. In this thesis, an increased potential for variation was desirable, to be able to adapt to new situation which may be particularly different from previous ones, as stated in 1.2. Some functionality would have to be added to allow for more dynamic behavior.

2.2.1.1 Back Propagation

A common supervised learning method for feed forward ANNs is Back Propagation [Bryson and Ho, 1969, Rumelhart et al., 1986, Callan, 1999]. It bases itself on adapting weights in the neural network to approximate a function represented by training examples, that is sets of inputs and the corresponding desired output. Given an example, when the network is given input, the corresponding output is checked against the correct output provided by the example. If the error is above an acceptable level set by the programmer, the weights are adjusted, proportionate to said error. This process is repeated, iterating across a set of examples until a criteria is met, such as a sufficiently small error or a sufficient number of iterations having been performed.

Over a number of iterations, with a sufficient number of training examples of sufficient variety, it is hoped that the network will be able to perform the tasks from which the examples are generated. This is of course contingent upon the network being sufficiently complex, that is to say it has a sufficient number of nodes in a sufficiently connected network, to perform the task in question.

The back propagation algorithm can be used in any acyclic network structure, as can be seen in Mitchell [1997].

2.2.2 Evolutionary Algorithms

Artificial Evolution is a powerful technique in computer science. It is inspired by the evolutionary processes seen in nature. As such, it is a term that covers a large number of techniques. A general feature of all Evolutionary Algorithms is a gradual approach to the desired solution, performing hill climbing behavior over a large search space. It will generally involve a certain degree of chance. The starting point for the search will generally be random (at least to a degree), and some randomization will usually be involved to avoid getting stuck in local maxima in the search space. A practical example of its use in games can be found in Lucas [2004], where the creation of a controller for the game Cellz was attempted.

2.2.2.1 Genetic Algorithms

Genetic Algorithms (GA) [Holland, 1975, Mitchell, 1996] are an often used evolutionary technique. The aspect that should be evolved (such as a series of actions, or the weights in a neural network) is encoded into a genome. There is a mapping method capable of changing the genome, or genotype, into a phenotype - what you are evolving. That is, if you were evolving a path for the traveling salesman problem, the genotype would be a basic representation that could be interpreted through some function into an ordered list of locations, which would be the phenotype. The genome is typically quite simple in form, such as a bit string, even though the phenotype may be quite complex. The mapping between the genotype and phenotype may be direct, meaning that the genotype gives the exact shape of the phenotype, or indirect, meaning that what is evolved in the genotype is a description for the construction of the phenotype.

The GA starts out with a large set of (random) genomes. It will then use a fitness function to evaluate their worth as solutions to the problem. Depending on their fitness, the individual genomes will be chosen to be passed on to the next generation. Chosen genomes may go through mutation - where random variations are introduced to the genome; or crossover - where two or more genomes are combined in simulated reproduction. See Figure 2.3.

The selection of individuals can be performed in several ways. Arguably the simplest is by roulette selection. This is where an individual is chosen at random from the population, where each individual has a chance proportional to their fitness. Tournament selection is another approach. Here, groups of individuals are selected at random to "fight" each other. The strongest in the group are chosen for reproduction. Selection pressure can be changed by changing the size of the group chosen - the larger the group, the lower the chance that weak individuals will end up chosen.

Elitism is an often used mechanism in GAs. This is when the best individual, or a number of particularly good individuals, are passed unchanged over to the next generation. This is particularly useful when the fitness function is static, but somewhat less effective when the function is changing. If the surrounding environment adversely adapts to the evolving individuals, the efficiency of elitism will be inversely proportional to the efficiency of the adaptive behavior.

Coevolution [Moriarty and Miikkulainen, 1998]is when the fitness of one individual is dependent on that of another. An example of this would be when controllers for a game are given a fitness based on how they perform against each other. Lets say that pairs are given fitness through duels, where a high fitness for one means a low fitness for the opponent. This would be an example of circumstances where elitism would have limited efficiency, unless the individual passed on through elitism has adaptation leading towards a solution of the game. Otherwise chances are opponents will adapt to it and overcome it.

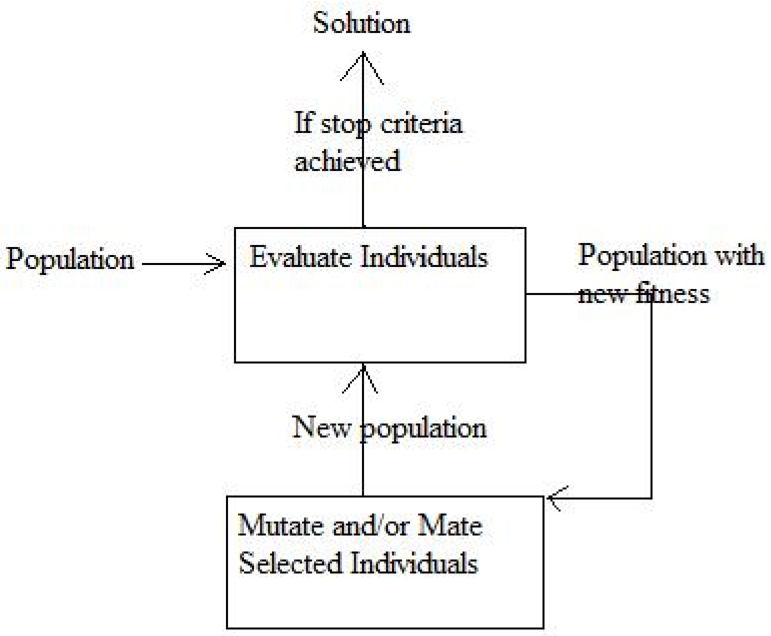

Solution
If stop criteria achieved
Population
Evaluate Individuals
Population with new fitness
New population
Mutate and/or Mate Selected Individuals

Figure 2.3: Basic Functionality of a Genetic Algorithm

Of course, coevolution does not need to be adversarial in nature. Fitness can be given for cooperative behavior.

When a sufficient number of genomes have been chosen for the next generation, the process is repeated. This goes on until a preset condition is met. The individual with the highest fitness is then presented as the solution to the problem.

2.2.3 Neuroevolution

Neuroevolution [Yao, 1993] can be considered a combination of evolutionary computation and artificial neural networks. It is used to remove the human from the design of neural networks, to varying degrees.

The most basic neuroevolutionary algorithms will use a neural network where the structure has already been designed. It will use an evolutionary algorithm to evolve the weights used in the network. Other algorithms will evolve both the network structure and the weights, allowing the network to adapt to a greater degree. Evolving both structure and weights can be both beneficial and a problem, depending on the application to which the network is to be used. On one side, a human understanding of the complexities of the task becomes less important, but on the other hand, the evolutionary process may very well miss obvious points relating to the task.

When using evolution to generate weights, or an entire network for that matter, there are two basic ways to go about transforming the genomes into their phenotype representations. It can be done directly, that is in such a manner that the genome translates directly to a network or its weights, with different parts of the genome referring to different parts of the corresponding network, or indirectly, that is in such a manner that the genome translates into a description of how one is to go about generating the network.

Neuroevolution has shown great adaptability in games, as can be seen in Westby [2011], Shi [2008], Reisinger and Miikkulainen [2007]. Given that elitism is used, when opposed by a deterministic opponent, a neuroevolutionary algorithm will be able to achieve a monotonous improvement across generations, in the best individual. This is due to the fact that given the deterministic nature of the opponent, any adaptation that performed well in the past, will perform at least as well in any later generation. Given elitism as well, the best individual of the past generation will always be present in the next. This in and of itself could be seen as a possible solution to the problems hoped to be solved in this thesis. Unfortunately, this was not so. Neuroevolution alone would be insufficient when hoping to achieve efficient adaptability to changing circumstances (in the case of games, changing opponents), as this would be non deterministic.

Burrow and Lucas [2009] showed some success using evolution to adapt the weights of a neural network. The network architecture remained unchanged. This limited the adaptability of the network, meaning that a basic approach to the game was implicitly introduced by the programmer. This can result in excellent performance when the network is of sufficient complexity (Togelius and Lucas [2006] showed results indicating the evolved neural networks could outperform human drivers when it comes to RC car racing), but it is required that the programmer has sufficient knowledge of the problem domain to create a useful structure.

When using neuroevolution only to change the weights in the network, it is important to ensure that the network is sufficiently complex to perform the task at hand. An example of the opposite isLucas [2004]. It did not have great success with using evolution to set the weights of a very simple neural network (a one layer perceptron net). However, it should be mentioned that while the evolved controller was out performed by a hand tuned controller, the author believed that better results could be achieved by a more complex network.

2.2.3.1 Neuroevolution and Games

Neuroevolution has on many occasions been used in conjunction with games of various complexity. This is due to the fact that games make for an excellent testing ground for AIs of various complexity. Examples range from more complex games such as driving games [Togelius and Lucas, 2005, 2006] and Cellz [Lucas, 2004], to simpler games such as Reversi [Westby, 2011], Nothello [Reisinger and Miikkulainen, 2007], Capture Go [Cai et al., 2010], and indeed Reversi/Othello [Chong et al., 2005].

Westby [2011], Reisinger and Miikkulainen [2007] and Chong et al. [2005] were of course of particular interest for this thesis. They showed that the game of Reversi, and games of similar mechanics, can be solved with good results by neuroevolutionary methods.

There are indications that the best way to evolve a player for Reversi is done through coevolution, a “hall of fame” of previous good players, and tournament selection[Shi, 2008, Moriarty and Miikkulainen, 1998, Westby, 2011]. However, as the purpose of this thesis was not to create an excellent Reversi player, but to explore a combination of learning and evolution. As such, this information was largely ignored in this thesis.

2.2.3.2 Neuroevolution of Augmented Topologies

Neuroevolution of Augmented Topologies (NEAT) [Stanley and Miikkulainen, 2002] is an approach to evolving ANNs, where not only weights, but also the structure is evolved. The method starts out with the smallest possible network, with no hidden nodes, and then adds more and more nodes. Neat will give all nodes the same activation function, namely sigmoid. To allow for individuals to change across generation, each is given a unique identifier, depending on at what point in evolution the individual occurred. Every mutation adding a new node or arc to the network is given an innovation number. Every innovation number present in one individual and not another, means that there is a structural difference between them. All together, the numbers become an identifier. Using this identifier, the system can then keep track of which individuals are compatible, and can perform crossover, without having to do a topological analysis of the individuals. See Figure 2.4 and Figure 2.5.The innovation numbers are inherited unchanged during crossover. The genome itself consists of node genes, defining neurons, and link genes defining links between two neurons. Mutation adds (and removes) nodes and links, as can be seen in Figure 2.4, as well as changes the weights of the links. This behavior allows the method to find a sufficiently complex network to solve the relevant problem, without creating an unnecessarily complex network [Stanley and Miikkulainen, 2002]. During crossover, matching node genes are passed on to the child by picking one at

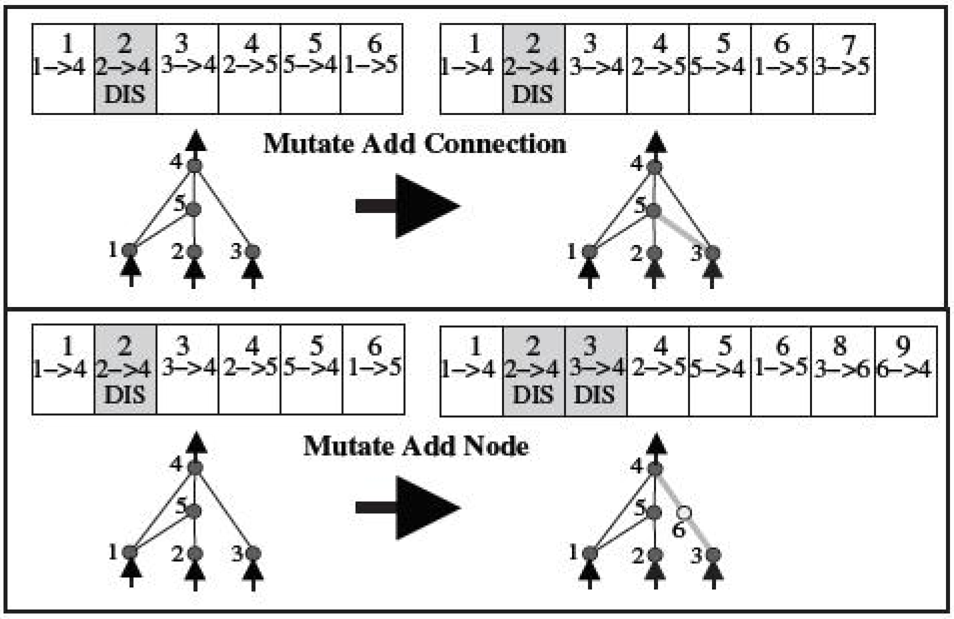


Figure 2.4: Mutation in NEAT as presented in Reisinger and Miikkulainen [2007]
Genes that have been shaded correspond to links that have been disabled through mutation. In the upper part of the figure you will note that there exists no direct link between node 2 and 4. The corresponding gene has been disabled and genes for a link between nodes 2 and 5, and a link between nodes 5 and 4 are present instead.

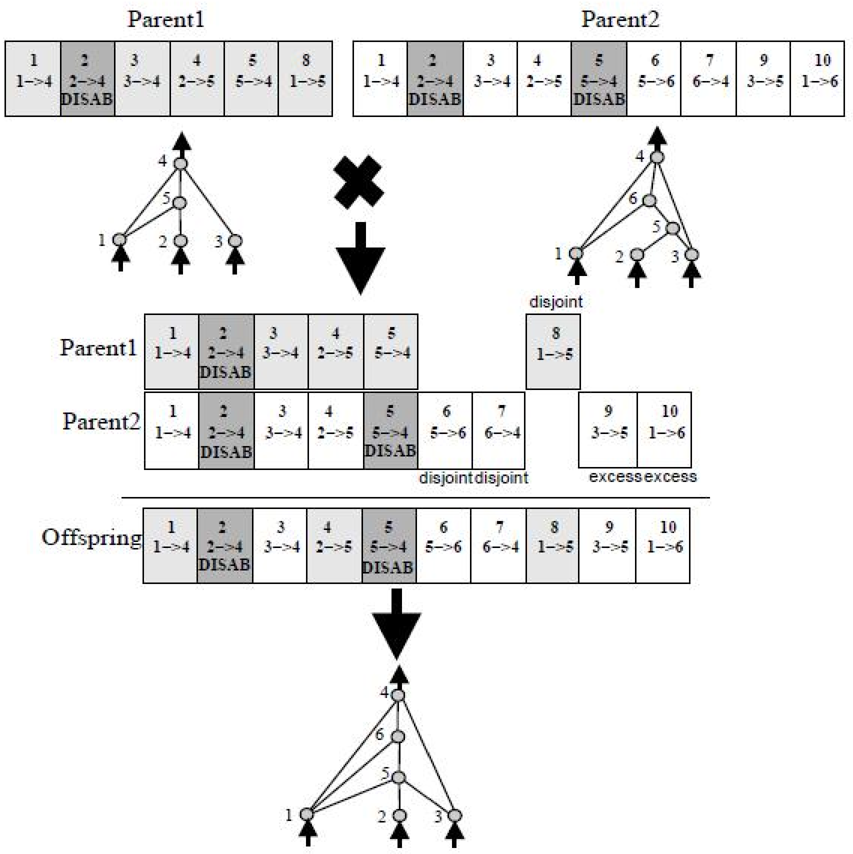


Figure 2.5: Crossover in NEAT as presented in Stanley and Miikkulainen [2002]

random between the two corresponding genes. Genes that only exist in one of the parents, are passed on given that the individual has higher fitness. In Figure 2.5, equal fitness is assumed, which results in all genes being passed on to the offspring.

Individuals in NEAT are divided into species, and compete mainly with individuals within their own species. This is to facilitate a greater search area for structures, as individual topologies have a greater chance of reaching optimization within their own species before they are set to compete with individuals of widely different structures. Individuals are divided into species based on their identifier, and new species are created through mutation. In two different individuals, the structural difference is in the form of either disjoint or excess genes. Disjoint genes are genes with a lower innovation id than the highest identifier of the genome in question, but which are still not present. Excess genes are genes with higher innovation ids than the highest in the genome in question. Speciation is given by the difference between two genomes, dependent on the number of excess and disjoint genes, and average weight differences. These factors are multiplied by coefficients, and when the result is above a threshold, the genomes are considered of different species.

The number of offspring a species is allowed is based on the average fitness within the species compared to the average of all individuals, and proportional to the total population.

To ensure a larger search space, NEAT uses fitness sharing within species. In other words, the fitness is divided by the number of individuals within the species. By sharing fitness, no one species can become too large, and differentiation is maintained. This, again, will facilitate a larger search space, and lessen the risk of the population converging to a local maximum rather than the global.

As a particular structure evolves further, there is also some hope that previous functionality is not forgotten, through crossover being dependent on species, as Reisinger and Miikkulainen [2007] states. This is an important point for the purposes of this thesis. If an ANN is created that performs well against one strategy, called strategy A, and the opponent applies a different strategy, strategy B, as counter, the ANN must be altered to handle it. However, it would be greatly preferable if it in adapting to strategy B did not forget how to fight strategy A. Otherwise, while the controller may avoid being defeated by the same trick repeatedly, it could potentially be repeatedly beaten by two alternating strategies. Considering the population as a whole, NEAT contains this functionality to a certain degree, as while NEAT works with a large search space, individual species work in a limited space. That means that while a single species contains specializations to defeat strategy A, another species may contain specializations necessary to defeat B. However, for the purposes of this thesis, it would be preferable to have all this information within a single individual.

NEAT uses a direct encoding of the network. Node genes correspond to the nodes in the network, and connection genes correspond to the connections, defining both the connections' weights and which nodes they connect. This means that any given genome has a one-to-one correspondence with a neural network.

In the experiments presented in Stanley and Miikkulainen [2002], NEAT uses a uniform activation function throughout the resulting neural network. However, there is no reason to limit oneself to this. The system utilizes node genes as well as connection genes, and presents a standard for what to do when genes subjected to crossover have different nodes. As such it is not a great challenge to introduce a few extra bits in the node genes to select an activation function from a preset set.

2.2.3.3 Hypercube-Based Neuroevolution of Augmented Topologies

Related to NEAT, is the hypercube-based neuroevolution of augmenting topologies, HyperNEAT [Stanley et al., 2009], approach. This is a method intended to exploit geometrical dependencies in the purpose to which the network is set. Contrary to NEAT, HyperNEAT uses an indirect encoding of the neural network. Whereas with a genome from NEAT you can recreate the corresponding network perfectly, a HyperNEAT genome will only tell you how to set the weight of a network with a known structure.

HyperNEAT uses an indirect encoding of the weights in the network, referred to as Compositional Pattern Producing Networks (CPPN). It evolves the weights of a neural network through the creation of a function depended on the position of the neurons. As an example, consider a Reversi board. Imagine

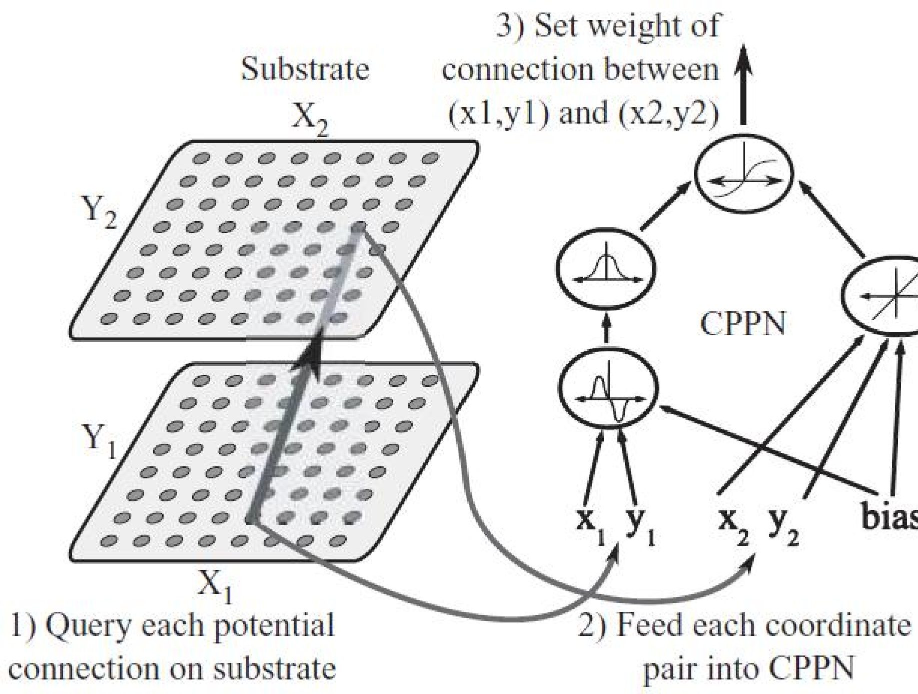
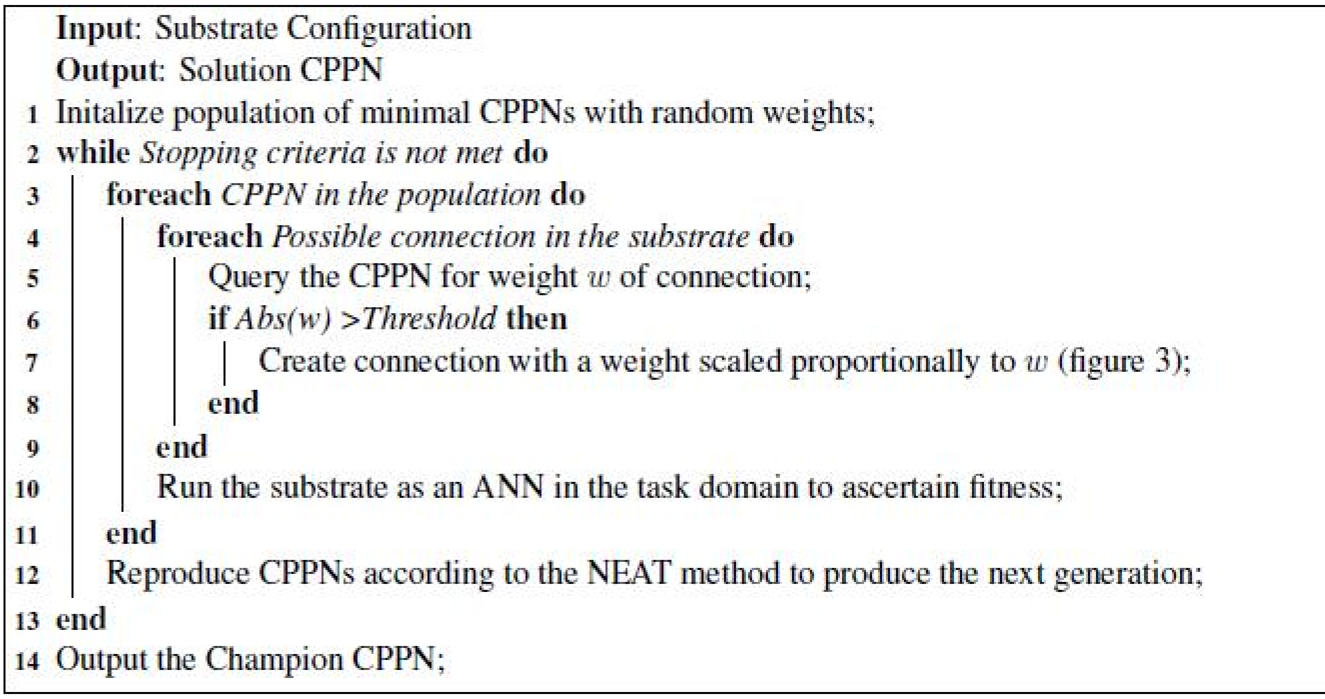

3) Set weight of connection between (x1,y1) and (x2,y2)
Substrate
X2
Y2
Y1
X1
CPPN
x1 y1
x2 y2
bias
1) Query each potential connection on substrate
2) Feed each coordinate pair into CPPN

Figure 2.6: CPPN as presented in Gauci and Stanley [2010b].

every space on the board has a corresponding neuron, with coordinates given by the two-dimensional coordinates of the corresponding board space. This set of neurons is then connected with a hidden layer, with the same number of nodes and corresponding coordinates. Given that the nodes in the first layer have coordinates (x_1, y_1), and nodes in the hidden layer (x_2, y_2), all weights are given by an evolved four-dimensional function $f(x_1, y_1, x_2, y_2)$. This is the hypercube referred to in the name of the technique. This means that geometric dependencies discovered in one region of the board can be evolved for all regions of the board. In the case of Reversi, HyperNEAT can evolve that if it has a piece in position (x, y), it is bad for the opponent to place pieces in positions $(x - 1, y)$ and $(x + 1, y)$. A conventional, directly encoded, evolutionary ANN would have to evolve this for **every** position (a, b), whereas HyperNEAT can learn it as a general rule. A further advantage of the indirect encoding is that it takes considerably less computer resources to store and evolve the weights of the network. As the weights are merely represented as a function, the CPPN, they can be calculated on the fly, rather than stored (although this does of course increase the resulting program's runtime complexity).

The connection to NEAT is in how the CPPN is evolved, and represented. In effect, rather than to evolve the network structure, as in NEAT, the structure remains constant, while a structure defining the CPPN is evolved. In section 2.2.3.2, it was mentioned that the NEAT standard as presented in Stanley and Miikkulainen [2002], while not explicitly stating, opens for node activation func-



```
Input: Substrate Configuration
Output: Solution CPPN
Initalize population of minimal CPPNs with random weights;
while Stopping criteria is not met do
    foreach CPPN in the population do
        foreach Possible connection in the substrate do
            Query the CPPN for weight $w$ of connection;
            if Abs(w) >Threshold then
                Create connection with a weight scaled proportionally to $w$ (figure 3);
            end
        end
        Run the substrate as an ANN in the task domain to ascertain fitness;
    end
    Reproduce CPPNs according to the NEAT method to produce the next generation;
end
Output the Champion CPPN;
```

Figure 2.7: Basic HyperNEAT algorithm as presented in Gauci and Stanley [2010b]

tion being defined by evolution. This is a part of the HyperNEAT standard, as presented in Gauci and Stanley [2010b]. See Figure 2.6.

In HyperNEAT, an artificial neural network with a static structure is created, formed in such a way as to reflect the geometry of the relevant task. Using Reversi as an example, a neural network with the nodes reflecting the two dimensional game board would be used, such as in figure 2.6. This network is in HyperNEAT terminology referred to as the substrate. A CPPN is then evolved, the same way a NEAT network would be evolved, but with varying activation functions in the NEAT nodes, see Figure 2.7. The CPPN then takes input reflecting the coordinates of the nodes that are to be linked, and the resulting output is then used as the weight of said link.

When examples of HyperNEAT are considered, such as Gauci and Stanley [2010b], the technique is generally put to rather simple use. The network is typically mirroring a game board, with a positive value for the player and a negative value for the opponent. Such an approach is not feasible if the game “board” (such as a map in a real time strategy game) becomes more complex than a two-dimensional grid. Adding a larger number of factors than simply the pieces of two opposing players also complicates the matter.

HyperNEAT should work well with the game chosen for this thesis - Reversi. The geometric relationship between the pieces is central to the game, and the game board is can easily be reflected in a substrate. HyperNEAT, with its sensitivity to geometry [Clune et al., 2009], can be used to good effect in board games where this is important. Gauci and Stanley [2010b] used it in conjunction with checkers.

As mentioned in 2.2.3, evolving both structure and weights of a neural network removes the human factor further from the game. Using the proper evolutionary algorithm, some assurance can be given that the resulting network will be of sufficient complexity to fulfill its purpose. By using HyperNEAT, the system proposed in this thesis would lose this. However, it was decided that while not having the network structure designed by humans would be beneficial, HyperNEAT would still be a good choice, due to the aforementioned sensitivity to the geometrical distribution of the nodes, through its use of a CPPN. A HyperNEAT based player would probably be able to learn Reversi faster than a system based on NEAT or another algorithm that would evolve the network structure as well. As stated, where HyperNEAT can understand the importance of the relative positions of the pieces as a general rule for the whole board, NEAT or an algorithm based on the same principles would have to evolve the rules for every position on the board separately.

2.2.4 Simple Recurrent Networks

When it comes to retaining information between generations in evolutionary artificial neural networks (EANNs), Ans et al. [2002] presents an interesting approach. This is an addition to learning in ANNs to allow for the learning of knowledge presented in sequence. The basic idea is using supervised learning to teach future generations the knowledge encoded in previous generations. The method presented is supported in Ans et al. [2004].

When a neural network learns something, through e.g back propagation, the weights are tuned in an attempt to make the network achieve the solutions given in the training set. If a new sequence of learning is introduced, chances are that the same weights that were previously tuned will be retuned to this new information, meaning that previous information is forgotten. Ans et al. [2002] presents a method to avoid this through the use of pseudopatterns. These pseudopatterns consists of random input patterns coupled with the associated output patterns as generated by the original network. When a new training set is presented, these pseudopatterns are interleaved with the training examples, meaning that the network attempts not only to learn a new set of weights, but also the weights representing the functionality of the previous training set. Typically, this is done with a pair of networks. One is an unmodified previous network, used to generate the pseudopatterns, while the other is the one being trained. The two articles mentioned also presents what is called Reverberating Simple Recurrent Networks, wherein the input values used to create pseudopatterns are generated by the networks. According to the articles, this modification increases the efficiency of the pseudopatterns.

To put this into context with this thesis, consider again the situation with game strategies A and B. Once the network adapted to strategy A is beaten, and a new generation is ready to be tested against B, random input is fed to the A network. The output from A is saved along with the random patterns fed to it, and is together referred to as a pseudopattern. The new network, which will attempt to beat strategy B, is then fed the same input as network A was. A learning method is then used to tune the weights in the new network towards the output given by network A. The articles give no indication that this technique was attempted used in conjunction with evolution, however, but is rather meant to use with back propagation networks (see Section 2.2.1.1).

The basic idea seemed of great interest. Creating reverberating networks, with an extra set of outputs to create inputs, for the next set of pseudopatterns was considered to be a bit invasive, in that it required more of a human touch in the structure and weights of the network. However, pseudopatterns were considered a sound means of transferring information between networks, and should work well with networks across generations. Particularly so, if the networks between which the information was to be transferred could be trusted to have the same structure, as would be the case with the substrates of HyperNEAT (see Section 2.2.3.3).

2.2.5 Minimax Search

While not a technique associated with machine learning or artificial evolution, minimax search[Russel and Norwig, 2003] is a method often utilized in artificial intelligences playing discrete time board games. It is used in association with lookahead.

When a game playing AI is using state evaluation, that is choosing a move based on a future state resulting from that move, it is not unusual to look several moves ahead, and associating a value to the imminent move based on the result off the moves further down the line. This is done as follows: Let us say a search is being done n layers deep. For every layer m, there are k groups, or branches, of moves, where k is the number of possible moves in layer m-1 that does not result in the game ending. Each branch consists of all possible moves following the possible moves of layer m-1.

Let us say that the nth move is done by the opposing player (hereafter referred to as player 2). Every possible move is given a value based on its worth to the player making the search (hereafter referred to as player 1).

In layer n-1, which corresponds to a move by player 1, every possible move is given a value equal to the lowest value of its corresponding branch on layer n (minimized). This is due to the fact that move n is performed by player 2, and it is assumed that player 2 will make the move that is most destructive to player 1.

In layer n-2, which corresponds to a move by player 2, every possible move is given a value equal to the highest value in its corresponding branch in layer n-1 (maximized). This is due to move n-1 being player 1's move, and the assumption that player 1 will make the move that results in the best possible outcome for him, given that player 2 does the same.

This keeps up all the way up to layer 0, where the player will choose the move with the highest value, see Figure 2.8.

Alpha-Beta Pruning To limit the tree structures generated by minimax searches, alpha-beta pruning[Russel and Norwig, 2003] is typically utilized. This is a method used to stop following branches that will probably never be followed by the players in the actual game (given that they act rationally). That is, in a layer where the algorithm is minimizing, if the value of a group is known to be higher than another, that group can be discarded for consideration.

Consider a search such as the one explained in section 2.2.5. Let us say that in layer k, which corresponds to a move made by player 2, it is found that move t in branch p will result in player 2 winning the game. It is then assumed that

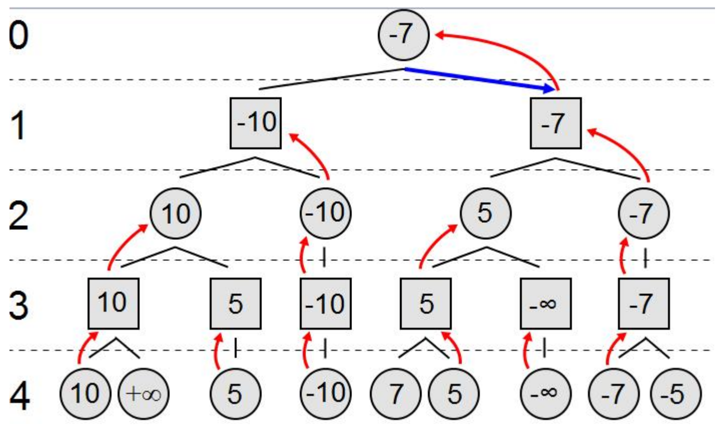


Figure 2.8: Minimax Search
Image by Nuno Nogueira, licensed under the Creative Commons Attribution-Share Alike 2.5 Generic license.

player 2 will make the rational choice, and choose this move should the situation where it is possible ever occur. Hence, there is no point in exploring layer k+1 from this group, as it will probably never be reached. Instead, the depth search is stopped, and group p will be given the value of a loss for player 1.

Correspondingly, on a layer corresponding to a move made by player 1, the search is similarly stopped, should a branch that results in player 1's victory be found.

Considering Figure 2.8, branch 7 (counting from the left) in layer 4 contains -infinity. Knowing this layer will be minimized, any other branches in layer 4 in that group would have been disregarded as soon as this value was found, as there could be no smaller value found and any higher value would be irrelevant.

Minimax search can greatly increase the efficiency of state evaluating AIs, but at the cost of increased runtime complexity. It takes time to extrapolate all the possible successive states, down through the plys. Alpha-beta pruning will decrease the load, but that is dependent on the situation. Given lookahead over a number of plys, and the state not being one close an end to the game, utilizing minimax searching can make a significant difference. However, in a game such as Reversi where the number of possible states are limited - particularly on smaller board sizes - and with a limited number of plys' worth of lookahead, it will generally increase the effectiveness of a naive player quite a bit. Given for example a greedy player, in the case of Reversi typically one that ignores board positions and merely considers the number of pieces left to the player after the move, a minimax search will make a significant difference.

2.3 The System

In light of the above findings in section 2.2, the following system was chosen for experimentation. It consisted of two main parts: the adaptive, HyperNEAT based system for generating players, and the simulator used to evaluate them.

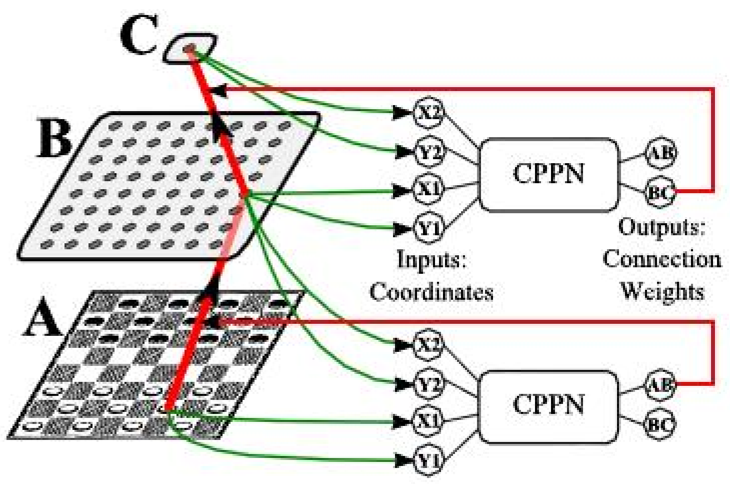


Figure 2.9: Generation of Substrate from CPPN as presented in Gauci and Stanley [2010a]
The CPPN is generated by the NEAT algorithm, and used to find the weights of the substrate, that is the actual ANN to be used for the target task. The two CPPNs seen it the figure is the same one in both cases. Note that the two layers of links take weights from different output nodes in the CPPN.

2.3.1 The Adaptive System

HyperNEAT, as seen in section 2.2.3.3 is a technique that lends itself well to boardgames such as Reversi, and was therefore chosen for this system. Using HyperNEAT, the resulting ANN can gain an understanding of the basic geometry of the game, which is central to the chosen game.

HyperNEAT results in a feed forward network, which furthermore lends itself to learning via back propagation. It was decided to implement this as the learning portion of the system.

A CPPN with four inputs and two outputs would be evolved. The resulting networks where then used to set weight values to the links in a static structure fully connected artificial neural network, with one input layer and one hidden layer, both with a number of nodes equal to the number of fields on the Reversi game board to be used, and one output layer with a single node.

Weights were generated from the CPPN in the following manner: the four inputs in the CPPN networks were used to input the coordinates of the nodes being linked, the first output being used to set the weights of links between input and hidden nodes, and the second output being used to set weights between hidden and output, as per HyperNEAT as presented in 2.2.3.3. Note that the same output could not be used for links in both cases, as that would result in duplicate links between the two link layers. See figure 2.9.

These resultant static structure networks, or substrates, would then be used to generate the fitness of the genome used to generate the NEAT network used to generate the weights.

Evolution of the network was performed as per HyperNEAT specifications [Stanley et al., 2009, Gauci and Stanley, 2010a], using a custom fitness function.

The substrates should have to be capable of generating pseudopatterns, and learning from them. It was decided to use back propagation to achieve this.

When using pseudopatterns in conjunction with evolution, a concern was that the pseudopatterns could overwhelm the mechanisms evolved by the algorithm. To avoid this, it was planned to create pseudopatterns from the individual to be trained as well. The learning network would then be trained on both sets of patterns, interleaved. This method was inspired from Ans et al. [2002], see Section 2.2.4.

With a neuroevolutionary technique capable of evolving structure as well as weights, a potential problem with this approach could be that capability of the network was dependent on the structure. To use an example, if information was to be transferred from network A to network B, and A and B had a fundamentally different structure, network B may not be able to mimic the functionality of network A, no matter how the weights are tuned. It could be that B would simply not be sufficiently complex. A way to rectify this problem could have been to evolve a third network to a point where it could learn the functionality of both A and B. However a different approach was chosen for this thesis. By using HyperNEAT, which will always result in essentially the same network structure (with some variation where weight falls below the threshold, see Figure 2.7), there will be little difference in complexity between network A and B.

Of course, a possible risk is that networks A and B use a fundamentally different approach to measuring the value of a board. If one individual is minimizing the opponent, while the other is maximizing itself on the board, it is difficult to say what would happen when the two networks try to learn from each other, if the network is sufficiently complex to internalize both approaches, or a synthesis there of. This is difficult to avoid when using an evolutionary method to generate weights in a network, but not statistically likely to happen often.

2.3.1.1 Implementation

For the implementation, a Java NEAT framework called ANJI (Another Java NEAT Implementation) [James and Tucker, 2005] (version used ANJI v2.0), available under the GNU Public License, was used. To be able to use ANJI as a basis for a HyperNEAT implementation, in the form of the CPPN, certain modifications had to made, and they are touched upon in this section. However, the encoding of the genome, as well as the implementation of the various mutation and crossover operations as mentioned in section 2.2.3.2, genotype to phenotype mapping, etc, were left as is. As ANJI is a NEAT implementation, a high level explanation can be found in Section 2.2.3.2.

ANJI could be configured to use two different methods of mutation for adding nodes. It could follow the classic NEAT method of having a certain probability of adding a hidden node at some point in the network, or it could be set to add a node at every point were a hidden node could be added with a certain probability. That is, given three points were a node could be added, and setting a probability of p to add a node through mutation, the second method would add approximately 3p nodes. As the system endeavored to follow implement HyperNEAT as presented in Section 2.2.3.3, only the classic approach to mutation was used.

Modifications Firstly, a basic framework for a static structure artificial neural network to be used for the substrate had to be made. It was a simple java construct, strictly feed forward with no recursion, and all neurons organized in layers. Seeing as it was designed for just one purpose, that is Reversi, it was designed only to create three layer networks (one input, one hidden, one output). Furthermore, the network was implemented with the ability to save to and load from simple text files, as well as the creation of pseudopatterns, and back propagation learning.

Second, according to Gauci and Stanley [2010b], HyperNEAT should be capable of using a number of activation functions in its CPPN. ANJI, as a NEAT implementation, could only define one activation function that will be used for each node type throughout any given run. That is, it was possible to define that input nodes use activation function A, hidden nodes activation B, and output activation C, but not that any given node type choose functions from the set <A,B,C,D,..,E>. A modification was made to allow for any given node to be given a random function, chosen from a preset set, upon creation, as according to HyperNEAT, see Section 2.2.3.3. This was achieved by modifying the method responsible for adding new nodes upon mutation, as well as the method adding new nodes upon start-up. A random number was generated, and used to pick a function from a preset list every time a new node was generated. The genome already stored the activation function of each individual gene, allowing them to persist between generations.

Finally, classes for fitness functions that utilized the substrate and the simulator (section2.3.2) had to be created. The exact behavior defined in these classes had to vary to a certain degree from experiment to experiment, but they all used the CPPN developed by the modified ANJI implementation to set the weights of a substrate, which was then used as to evaluate states in simulated Reversi games. This was achieved simply by sub-classing the relevant fitness class in the framework.

2.3.2 The Simulator

To generate the fitness values of the individuals, a simulator was needed to simulate the games of Reversi played by the individuals. It would have to be capable of keeping track of the players turns and the board states, as well as reporting the results of games.

To test the ANNs generated by the adaptive system (Section 2.3.1), some sort of opponent to play against was needed. To have an objective reference between runs, coevolution would not be useful. Fitness generated through evolved ANNs playing against each other, would only indicate the various ANNs' level of play in comparison with other individuals within the population. That would mean that an individual with an excellent performance withing one generation, could potentially be far inferior to a poorly performing individual from another generation - or run - given that the rest of the individuals were even more superior. Hence, a player with a constant behavior would become needed for experiments. This part of the system therefor also included static behavior players used for testing of the neural net based players.

2.3.2.1 Implementation

This was a fairly simple Java construct. A Board object was used to set up and keep track of the board itself, as well as to administrate the turns of the players, and to ascertain when the game was over and its result. The Board contained two Player objects, each of which directly manipulated the board state. The Player objects contained an evaluator of some sort, embodying the heuristic used by the player to consider moves. Each Player would take the board state, generate all possible successor states, and choose the most beneficial next move (as given by their player specific board evaluation method). When no available moves were found, the Player in question would report this to the Board. When both Players consecutively reported passes, the board would end the game, count the pieces for each player, and return the result.

In the simulator, the players' pieces on the board were represented as integers in a two-dimensional array, 0 being an empty square, 1 being black and -1 being white. However, internally, both players saw their own pieces as 1 and the opponent as -1. That is to say, before the board was considered by a white player, the board values would be inverted. This was to ensure that a player designed to play as one colour would still be technically able to play as the other colour (though possibly not as well).

The Players Two static behavior players were implemented. A Greedy Player, and a Random Player.

The Greedy Player was implemented to look through the possible successor states, and choose the move resulting in the most pieces belonging to the player on the board. In other words, the player evaluated states to have a value directly proportional to the number of pieces belonging to it being on the board.

The Random Player was implemented to pick a successor state at random, using Java's built in pseudorandom number generator.

For the Greedy Player, code was implemented to facilitate n-ply lookahead, using minimax and alpha-beta pruning (see Section 2.2.5). It was not implemented for use with the Random Player. Lookahead would serve no purpose for it. The random player would choose a move at random, regardless of what moves may follow, and the resulting state of the board.

In the case of the artificial neural net based player, it was decided not to utilize any lookahead, in large part to reduce complexity. As the ANN based player would evolve the evaluating function, it should be capable, at least in part, to achieve an simplistic lookahead of sorts, in its evaluation of board positions. Also, the reduction in complexity was deemed to make up for a reduction in the ANNs' ability as a Reversi player, as the point of interest in the experiments would be the results of introducing learning in various ways, rather than the evolution of a Reversi player.

Chapter 3

Research Results

3.1 Experiments

In this section follows an explanation of the experiments used to judge the performance of the system produced for this thesis.

3.1.1 Experimental Setup

The basic system was one as described in Section 2.3. Every experiment was to be repeated a number of times (specified in each experiment), and the averaged results was used, to achieve statistical significance of the results.

Fitness inn all experiments was based on the results of the evolved individauls when playing against one of the static behavior players, mentioned under Section 2.3.2, in Reversi. That is to say, during the evolutionary process (as described in 2.3), the individual would play a number of rounds against the specified opponent, the individual's fitness given by the average over all rounds. This was not the fact where the system is set to play against the Greedy Player. Given that the player was deterministic, and change in the neural network only occurred between sets of matches, every match within one fitness evaluation would have the exact same result. Hence, the evolved individuals only played once against Greedy Players to generate fitness.

Fitness was generated as follows. Each evolved individual was used to generate a fully connected feed forward network, with one input layer, one hidden layer and one output layer. The input and hidden layers has the same number of nodes as there are squares on the game board. This resulting neural network was then encapsulated in a Player object, used in the game simulator. The simulator then simulated a number of games, reporting the results.

In all experiments, the fitness was given by the number of pieces of the individual's colour at end of play, divided by the number of pieces total at the end of game. In other words, if K is the number of pieces belonging to player k at the end of the game, and $K^{-1}$is the number of pieces belonging to the opposing player at the end of the game, fitness f_k of player k generated across n games is given by

$$f_k = \frac{\sum_{t=1}^{t=n} \frac{K_t}{K_t + K_t^{-1}}}{n}$$

where n>=1.

While it may seem tempting to use the number of pieces controlled divided by the maximum possible number of pieces (in other words the number of squares on the board) at the end of game, this would create a bias for longer games. With fitness defined like that, a player that took all the opponent's pieces in three plys would have gained a lower fitness than one who did it in ten plys. This was not desirable.

Note that because fitness was given by the ratio of pieces on the board belonging to the player to the total number of pieces on the board at end of game, averaged over a number of games, there was no way to know if the resulting player had solved the game. Even if a player were to achieve an average of 0.51, this could easily mean that it won massively in a few rounds, and bare lost in the rest, or similar (unless given a deterministic opponent, of course). This was not deemed a problem, as the goal of this thesis was not to create an excellent Reversi player, but to ascertain the viability of introducing learning in the form of pseudopatterns to neuroevolution.

3.1.2 Experiment 1: Colour Specific Player Knowledge Transfer

The intention of the first experiment was merely to ascertain if any effect could be achieved by utilizing pseudopatterns to transfer information between ANNs generated by the HyperNEAT algorithm.

The first experiment was to train an individual on a six by six Reversi board, playing only one colour. The opponent was a four ply lookahead Greedy Player, that means a Greedy Player that chose a move based in state evaluation of board states after four moves total (two by each player). As the opponent was deterministic, fitness was generated from a single game per genome, per generation.

After training an individual to a perfect performance (that is, fitness of 1.0), pseudopatterns were generated from the individual. A new individual was trained playing as the other colour. Both players were then tested against the same Greedy Player, but playing as the other colour. These results were used as a baseline, to be compared to results in the latter part of the experiment.

Pseudopatterns were then generated from each individual, and used to train the other. Both player were then tested again, playing both as their original colour, and the opposing colour. The results were compared to those previously gathered. See Figure 3.1.

In implementations of HyperNEAT, such as presented in Gauci and Stanley [2010b], weights with an absolute value below a certain threshold would result in the removal of the relevant link in the substrate. A consideration that had to be made in regards to this convention, was that by actively removing links, the different substrates would acquire different structures, and therefore different knowledge potential. This could potentially result in different networks becoming unable to learn the information encoded in the different networks. With this in mind, the experiment was performed twice, once where links with weights below the threshold had their weights set to 0.0, and once with said links being removed entirely from the substrate, by being set to a null value.

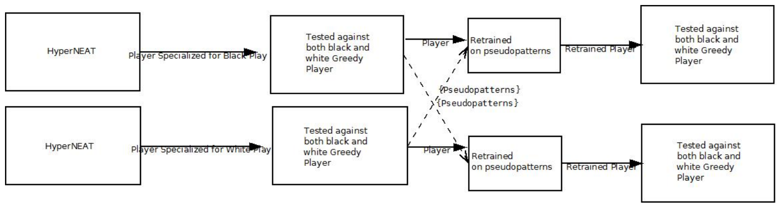

HyperNEAT
Player Specialized for Black Play
Tested against both black and white Greedy Player
Player
Retrained on pseudopatterns
Retrained Player
Tested against both black and white Greedy Player
{Pseudopatterns}
{Pseudopatterns}
HyperNEAT
Player Specialized for White Play
Tested against both black and white Greedy Player
Player
Retrained on pseudopatterns
Retrained Player
Tested against both black and white Greedy Player

Figure 3.1: Diagram of Experiment 1

Settings Both experimental setups were run 40 times, with the same settings. All games were played on 6x6 boards. 4 levels of lookahead was used for the Greedy Player, using minimaxing and alpha-beta pruning, to ensure a certain level of play. The level of lookahead was chosen through experimentation.

Weights with an absolute value below 0.2 were set to 0.0 in the runs were setting the arcs to null were avoided. In the other runs, the corresponding arcs were completely removed from the substrate. This threshold was chosen based on Gauci and Stanley [2010a].

For the HyperNEAT settings used on this experiment, consult Table 3.1.

All activation functions in the substrate were set to sigmoid. Linear activation functions were tested as well, but sigmoid seemed to work better in experiments performed to test the system. In the CPPN, the four input and the two output nodes were set to use linear functions in all runs, and all hidden nodes set to use sigmoid. Previous to this experiment, the system had been explored with simple experiments, and it was found that against the Greedy Player, these settings would find solutions faster than when letting the algorithm select activation functions itself. If solutions could be found, it was found preferable that they be found fast.

Fitness was as stated in Section 3.1.1, generated from a single game, as the opponent used was deterministic.

As for the coefficients used for spesciation, the settings were chosen based on experimentation and the results presented in Stanley and Miikkulainen [2002].

The information transfer via pseudopatterns was implemented according to the basic presentation in Ans et al. [2004]. When transferring information from net A to net B, a new net C would first be created. This would then be trained with pseudopatterns from both A and B. B would then be trained with pseudopatterns created from C.

After completing the runs, backpropagation was used with the following settings: a new net with randomized weights was trained on 120 randomly generated pseudopatterns. The learning net was then trained on 100 randomly generated pseudopatterns from the new net. All patterns were run a maximum of 5 times, with a minimum of 1, until average error dropped under 0.09. Learning rate was set to 0.01. These settings were chosen through experimentation.

3.1.3 Experiment 2: Pseudopattern-Supported HyperNEAT

In the first experiment, pseudopatterns were only used to train the final individual, and pseudopatterns were generated and applied in a fairly naive manner - there was no evaluation of the patterns, or their application.

Board Size	6x6
Opponent	Greedy Player
Number of Rounds Played	1
Weight Threshold	0.2
Population Size	100
Max Number of Generations	100
Add Connection Mutation Rate	0.05
Remove Connection Mutation Rate	0.02
Add Neuron Mutation Rate	0.01
Stranded Neurons	pruned
Weight Mutation Rate	0.8
Weight Change std.dev.	2.0
Maximum Weight	500
Minimum Weight	-500
Elitism	true
Crossover Rate	0.85
Minimum Species Size for Elitism	2
Selection Method	roulette
Recurrence in CPPN	disallowed
Species Size Fitness Adjustment	no
Substrate Activation Function	Sigmoid
Input Node Activation Function	Linear
Hidden Node Activation Function	Sigmoid
Output Node Activation Function	Linear
Excess Gene Coefficient	1.0
Disjoint Gene Coefficient	1.0
Weight Difference Coefficient	0.4
Speciation Threshold	0.5

Table 3.1: HyperNEAT Settings for Experiment 1
The settings for the HyperNEAT implementation used in experiment 1: Colour Specific Player Knowledge Transfer. The maximum and minimum weight bounds refer to the CPPN, not the substrate. The minimum size for elitism refers to the minimum size a species must have to have its members considered for elitism. The activation functions of input, output and hidden nodes refer to the CPPN.

In other words, there was very little control exerted on the process. While this could be considered beneficial in the way that it avoided human bias sneaking in to the process, allowing evolution to run its own course, it also opened for far more destructive results from the learning process. Broadly speaking, there were two refinements that could be introduced to the process.

One was exerting more control on the generation of pseudopatterns, ensuring that they were representative of the network generating them. That is, ensuring that the pseudopatterns embodied the desired adaptation present in the network.

The other was exerting more control on how they were applied to the learning network, ensuring that the right patterns were applied to the right networks. That is, ensuring that the patterns were not applied to a network that already had the adaptation that the patterns embodied, or ensuring that patterns that only created destructive results were not applied.

In this experiment, it was attempted to apply the second refinement. Not because the first refinement was deemed less useful, but rather that the first refinement would require far more complex evaluation. There are some attempts to achieve the first refinement in Ans et al. [2002] and Ans et al. [2004], but they appear to require far more control of the network (weights and structure). Such an approach is poorly compatible with neuroevolution, where the factor of human design is attempted removed to varying degrees.

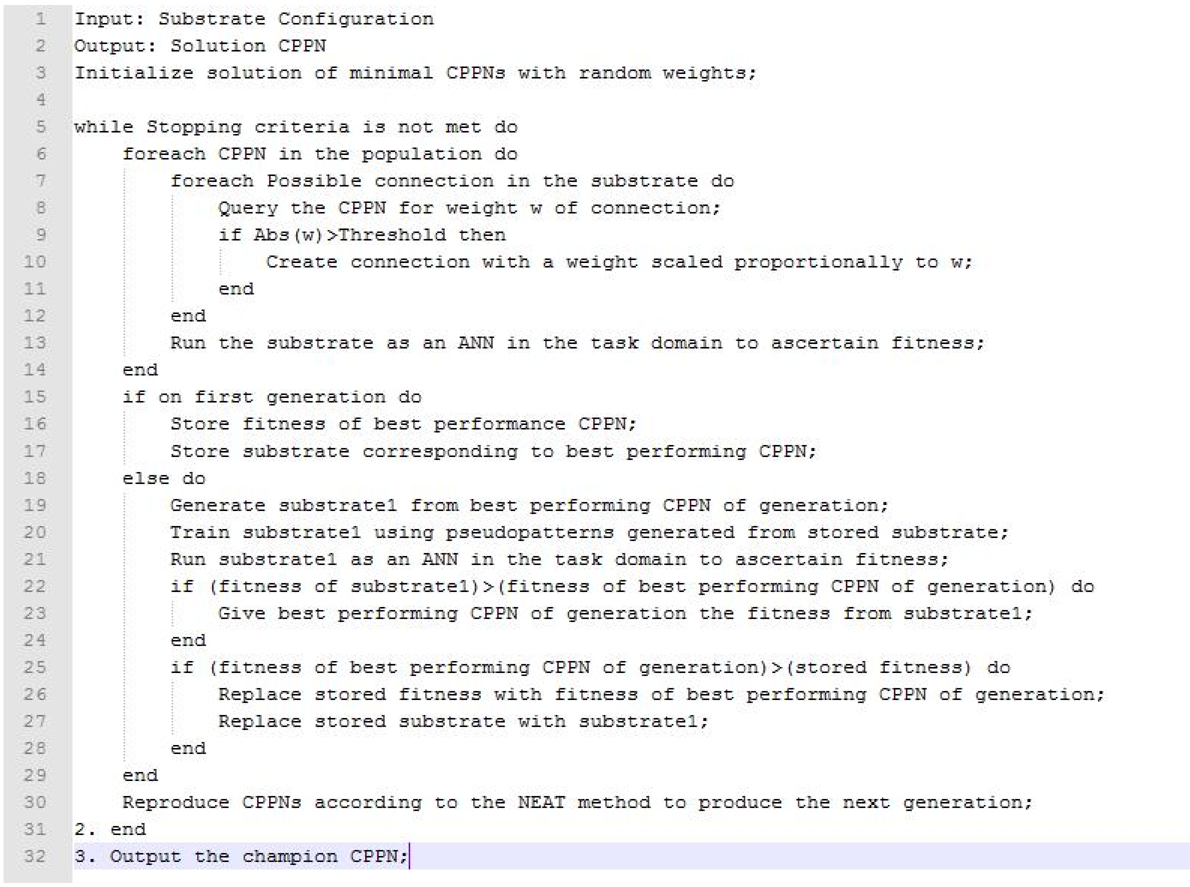
In this experiment, the performance of a non-learning population, that is a population evolved through the standard HyperNEAT algorithm using no learning, was used as a baseline. Compared to this, was the performance of a population, where the best individual of every generation was trained on pseudopatterns generated from the previous best individual across all generations. If the training resulted in an improvement, the individual used to generate the substrate was given a new fitness taking this improvement into account. If the trained individual (or untrained if training was found to have destructive or no results) was found to have a higher fitness value than the previous best individual, this new individual would take over this role, and be used to generate new pseudopatterns to be used for training.

In other words, at any one point in time, only one individual was kept across generations, to generate pseudopatterns. When a new individual was found to be the best across all generation to have occurred so far, it would be kept, taking the place of the last one. Meaning that when this occurred, old patterns would only have an influence over future individuals, given that the individual used to generate new patterns was improved by the old patterns, and therefore internalized them before new patterns were generated.

In this experiment, the opponent was the random player. This was to make it more difficult for the evolved individual to find a perfect solution. It was expected - and corroborated by the results seen in Section 3.2.1 - that a deterministic opponent does not require a great deal of adaptation to beat.

If this approach should work, it would be possible that it would not only result in the improved performance of the best individuals of the run, but in some limited manner also encourage the evolution of networks that could more easily assimilate the information from the best individuals of the previous generations.

The specifics of the extended HyperNEAT algorithm can be seen in Figure 3.2.



```
 1  Input: Substrate Configuration
 2  Output: Solution CPPN
 3  Initialize solution of minimal CPPNs with random weights;
 4
 5  while Stopping criteria is not met do
 6      foreach CPPN in the population do
 7          foreach Possible connection in the substrate do
 8              Query the CPPN for weight w of connection;
 9              if Abs(w)>Threshold then
10                  Create connection with a weight scaled proportionally to w;
11              end
12          end
13          Run the substrate as an ANN in the task domain to ascertain fitness;
14      end
15      if on first generation do
16          Store fitness of best performance CPPN;
17          Store substrate corresponding to best performing CPPN;
18      else do
19          Generate substrate1 from best performing CPPN of generation;
20          Train substrate1 using pseudopatterns generated from stored substrate;
21          Run substrate1 as an ANN in the task domain to ascertain fitness;
22          if (fitness of substrate1)>(fitness of best performing CPPN of generation) do
23              Give best performing CPPN of generation the fitness from substrate1;
24          end
25          if (fitness of best performing CPPN of generation)>(stored fitness) do
26              Replace stored fitness with fitness of best performing CPPN of generation;
27              Replace stored substrate with substrate1;
28          end
29      end
30      Reproduce CPPNs according to the NEAT method to produce the next generation;
31  2. end
32  3. Output the champion CPPN;
```

Figure 3.2: HyperNEAT Algorithm Extended With Learning

The reader may notice the algorithm is much the same as the HyperNEAT algorithm presented in Figure 2.7, in Section 2.2.3.3. The additions are in lines 15 through 28.

Board Size	6x6
Opponent	Random Player
Number of Rounds Played	100
Weight Threshold	0.2
Population Size	100
Max Number of Generations	100
Add Connection Mutation Rate	0.06
Remove Connection Mutation Rate	0.03
Add Neuron Mutation Rate	0.02
Stranded Neurons	pruned
Weight Mutation Rate	0.8
Weight Change std.dev.	2.0
Maximum Weight	500
Minimum Weight	-500
Elitism	true
Crossover Rate	0.8
Minimum Species Size for Elitism	3
Selection Method	roulette
Recurrence in CPPN	disallowed
Species Size Fitness Adjustment	no
Substrate Activation Function	Sigmoid
CPPN Node Activation Functions	step, tanh, linear, sigmoid, tanh cubic, signed step, evsail-sigmoid
Excess Gene Coefficient	1.0
Disjoint Gene Coefficient	1.0
Weight Difference Coefficient	0.4
Speciation Threshold	0.3

Table 3.2: HyperNEAT Settings for Experiment 2

Settings Given that the opponent in this experiment was a Random Player (that is, not a player chosen at random, but a player that chooses its moves at random), fitness had to be given by the average performance over a number of games, rather than just from one game, to give some idea of the evolved players performance. Fitness was as stated in Section 3.1.1, generated across a hundred games.

The runs, both for the HyperNEAT and the extended algorithm, were performed 20 times.

The games were played on a six-by-six board.

A perfect solution could not be expected to be found (though theoretically possible, given that the game is solved on 6x6 board). As such, the experiment was only allowed to run for a hundred generations, rather than until a solution was found. The HyperNEAT algorithm and the extended HyperNEAT algorithm were both run 20 times.

The HyperNEAT and extended HyperNEAT runs both used the same settings for the evolution. See Table 3.2. Note that the speciation threshold was set slightly lower than in the previous experiment. This was due to a desire to increase the size of the search space. By utilizing a lower threshold, individuals were given longer time to evolve within their own species, optimizing in direction of local maxima, and as such avoiding convergence of the whole population. During tests performed on the system, this seemed to find better peak individuals with greater ease.

The arcs in the substrate were set to be non-nullable, that is arcs with weights below a threshold of 0.2 were given the weight 0.0, rather than being removed from the substrate.

For this experiment, a slightly different back propagation scheme was used, based on positive experimental results. In the first experiment, a third network was used to gather the data from the other two, by being trained on pseudopatterns from the learning and teaching networks. The learning network was then trained on patterns from the third network. In this experiment, the learning network was first cloned. It was then trained on a set of pseudopatterns created from a previously evolved network. A new set of pseudopatterns were then created from the clone network, and used to train the learning network. The process was then repeated, for a set number of cycles, or until the error dropped below the threshold. For every iteration, new pseudopatterns were created from both the teaching and cloned networks. By generating new patterns, a wider representation of both the teaching network and the learning network would be used to teach the learning network.

The learning rate, number of pseudopatterns and error threshold were also altered with respect to the previous experiment. Learning rate was set to 0.04, 100 pseudopatterns from the teaching network were mixed with 200 pseudopatterns from the cloned network, back propagation ran on these for a maximum of 300 cycles, or until average error was below 0.001. According to results gathered from basic experiments, this resulted in a good ratio between learning new adaptations and destruction of old adaptation when using pseudopatterns as described for this experiment.

3.2 Experimental Results

This section contains the results gathered from the experiments in Section 3.1, along with some analysis of said results.

All graphs in this sections were rendered using JFreeChart 1.0.13 [JFr, 2009], released under GNU Lesser General Public License.

3.2.1 Experiment 1: Colour Specific Player Knowledge Transfer

As mentioned in 3.1.2, the experiment was carried out twice, once with links with weights below a certain threshold being removed from the substrate, and once with them set to 0.0. As such, each of the following subsections have been split in two, presenting the results separately.

3.2.1.1 Results

Note that all runs terminated upon reaching a perfect solution. This occurred after a varying number of generations. As the data presented here is the average over all the runs, the graphs run until the generation wherein the longest

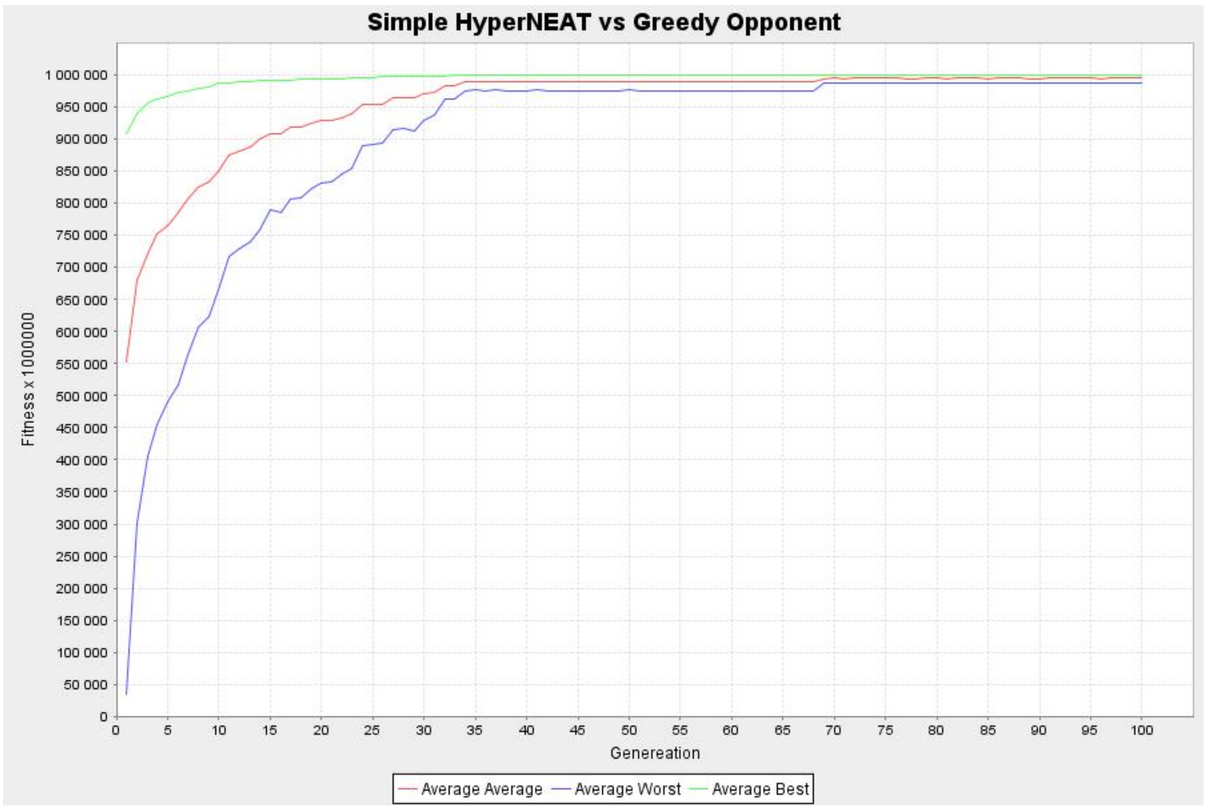


Figure 3.3: HyperNEAT vs Greedy Opponent

lasting run found a solution. When a run reached a solution before this, every subsequent generation was considered to have a perfect score on the entire population.

To give an example, consider two runs, each with three individuals. The first run reached a perfect solution (fitness 1.0) on the first generation, with an average fitness of 0.5. The second run reached a perfect solution on the second generation, with an average of 0.3 on both generation, and the best individual being 0.4 on the first generation. Tracing the resulting average best results would then be $\frac{0.4+1.0}{2} = 0.7$ on the first generation, and $\frac{1.0+1.0}{2} = 1.0$ on the second. The average average would be $\frac{0.5+0.3}{2} = 0.4$ on the first generation, and $\frac{1.0+0.3}{2} = 0.65$ on the second generation. The average of the worst individual was treated the same way.

Non-Nullable Arcs Average generation of solution: 10.5125

Minimum number of generations before solution: 1

Maximum number of generation before solution: 100

See Figure 3.3.

The average performance of the final evolved individual before and after being trained using back propagation, was as follows:

Averages over 40 games:

Black playing as black: 1.0

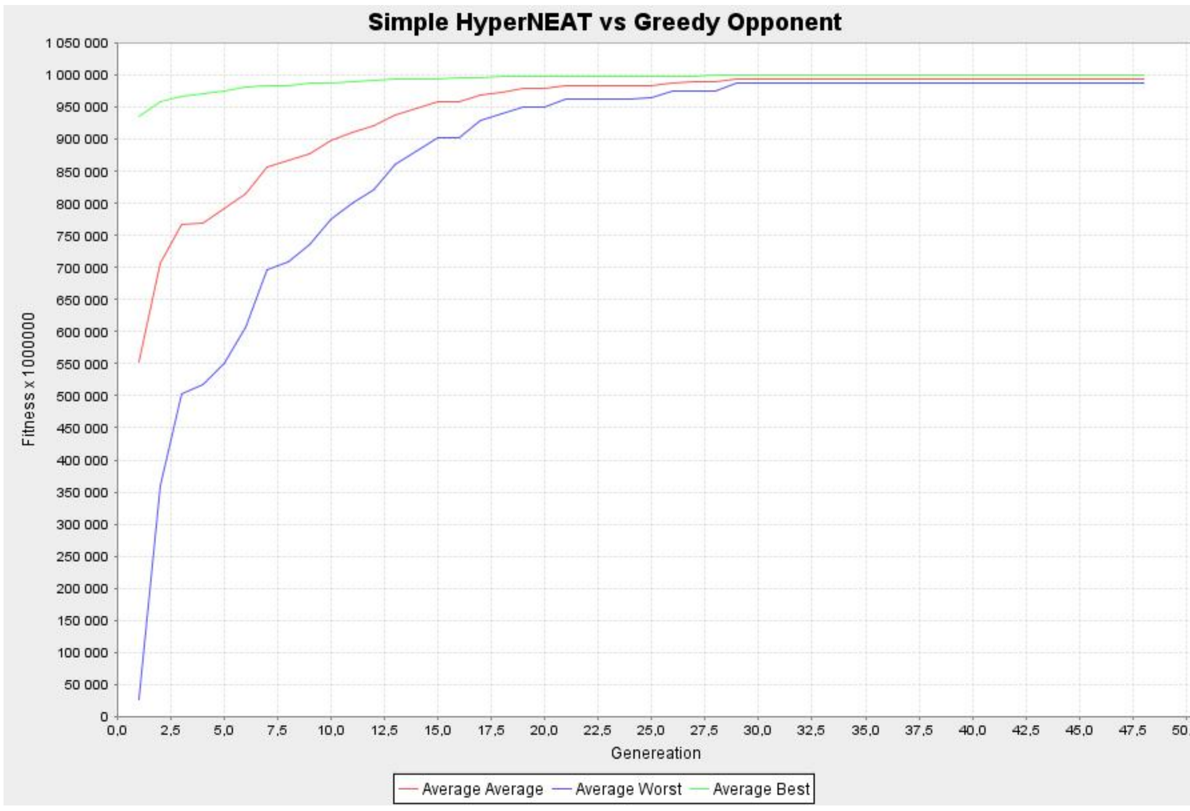


Figure 3.4: HyperNEAT vs Greedy Opponent, with nullable arcs in the substrate

Black playing as white: 0.2875
White playing as black: 0.6488
White playing as white: 1.0
Retrained black playing as black: 0.7786
Retrained black playing as white: 0.3479
Retrained white playing as black: 0.6424
Retrained white playing as white: 0.8882

Nullable Arcs Average generation of solution: 6.4
Minimum number of generations before solution: 1
Maximum number of generation before solution: 48
See Figure 3.4.

The average performance of the final evolved individual before and after being trained using back propagation, was as follows:
Averages over 40 games:
Black playing as black: 1.0
Black playing as white: 0.2639
White playing as black: 0.6573
White playing as white: 1.0
Retrained black playing as black: 0.9847

Retrained black playing as white: 0.2639
Retrained white playing as black: 0.6358
Retrained white playing as white: 0.9924

3.2.1.2 Evaluation

In the case of the runs with non-nullable arcs, we can see that all but one run had achieved a perfect solution by generation 70, and most of them by generation 35. It is hard to tell why a single individual did not achieve this before the run ended at generation 100. However, seeing as it was only one individual, it was assumed that this was merely a freak of statistics, and probably not a statistically significant aberration. It was as such largely discounted as unimportant.

A slight problem with the system as it was presented was found during these experiments. The random nature of the pseudopatterns mean that it can not be known how representative of the network it is generated from it will be. This means that the number of pseudopatterns needed is not known. The papers Ans et al. [2002, 2004] present some refinements to avoid this problem. However, for this thesis it was decided to rather attempt to increase the number of patterns, and/or otherwise evaluate them. Of course, a problem with this was that if the input of the generated pseudopatterns were not spread out sufficiently, they could end up causing the trained network merely to cover a small part of what the original network was capable of. The trained individual could end up completely different from the training individual.

An interesting result that could be seen in both runs, was that the individual trained to play as black did considerably poorer as white, than the individual trained as white did playing as black. This is probably more due to the nature of the game, Reversi, than the algorithm utilized in the experiment. A rule of the game is that the black player always moves first. This results in an advantage, as it limits the white players opening choices. It results in a further advantage on a smaller board, as the available moves are further limited. This again could result in the evolved black player needing a lesser understanding of the game to win, whereas the white player must be able to perform more complex strategies to win.

In both sets of runs, it was apparent that the inclusion of back propagation had an effect on the result. However, there were certain differences in the exact effect.

Non-Nullable Arcs In the first set of runs, retraining appeared to have a negative effect on the black player playing as black. Though it should be noted that the actual fitness value was not the win rate, but the percentage of pieces belonging to the player at games end. This meaning that a fitness in excess of 0.7 could mean a 100% win rate, as it means that, on average, 70% of the pieces at games end belonged to the player, meaning a 100% win streak may have been present, though it is impossible to know after the fact. While the opponents were deterministic, the generations of pseudopatterns was not, and as such neither the results of the retraining. However, the retraining appeared to result, on average, in an improvement for the black player playing as white, at the cost of its performance as white. The white player, on the other hand, seemed to simply be damaged by the learning process.

Considering what was previously noted in this section, the advantage the black player has in Reversi on a six by six board, the damage done to the white player was not altogether surprising. It was conceivable that the strategy of the black player was more, for lack of a better word, naive. As the black player performed poorly playing as white, there is no surprise that the white player learning from the black player would have a destructive effect on the white player's performance as white. The negative effect on the players game as black could have been caused by the same. If the white player had a fundamentally different approach, the influence of the black players simplistic approach could be exclusively damaging. It was also possible that the damage was caused by a flaw in the learning process. It was hoped that further experimentation would shed more light on the matter.

Considering the possibility that the strategies employed by the black players and the white players were fundamentally different to the point that learning from their counterparts would have exclusively destructive results, the decrease in the black players' performance as black is clearly understandable, while the slight increase in the black players' performance as white after retraining, may simply have been the result of noise in the experiments. However, it may also signify an actual improvement. It is possible that the black players approach to the game was so simplistic that being trained even in a fairly incompatible approach may be preferable to no training at all. That the black players' approach to the game when playing as white was lacking, could be clearly seen by the poor performance pre re-training. Further experimentation was required.

Nullable Arcs Interestingly, the statistics of the two sets of runs, pre-learning, were fundamentally similar. Not an unexpected result considering the same deterministic opponent was used in both sets. However, once learning was applied, there were some differences.

The destructive effect of learning on the black players' game as black, as well as on the white players' game appeared generally more limited, and no positive effect on black players' game as white was seen. In the case of the lack of positive effect, this seemed to indicate further that the positive effect seen in the previous set of runs was a result of noise. It could also be a result of incompatibility between the learning and teaching networks.

No analysis was made of the differences in substrate structures resulting from the HyperNEAT algorithm, so there was no way of measuring the differences between the learning and teaching networks. However, looking at the gathered statistics, the effect of the back propagation seems to be weaker than in the previous set of runs. It was not a significant difference, so it was difficult to say with any real certainty, but a probable cause was found to be the fact that unlike in the previous set of runs, the substrates could have quite different structures. Theoretically, the structures could vary from a fully connected, with $(36 \times 36) + 36$ arcs total, through a minimally connected network with 2 arcs (input-to-hidden and hidden-to-output), to a nonconnected structure (theoretically possible, but has not been verifiability observed in experiments, such a substrate would result in the first possible move found always being chosen by the player). Given a different structure, it was not a surprise that learning would be more difficult. Given that a substrate is of sufficient complexity, it should be able to learn the same functionality as any other substrate. However, a substantially

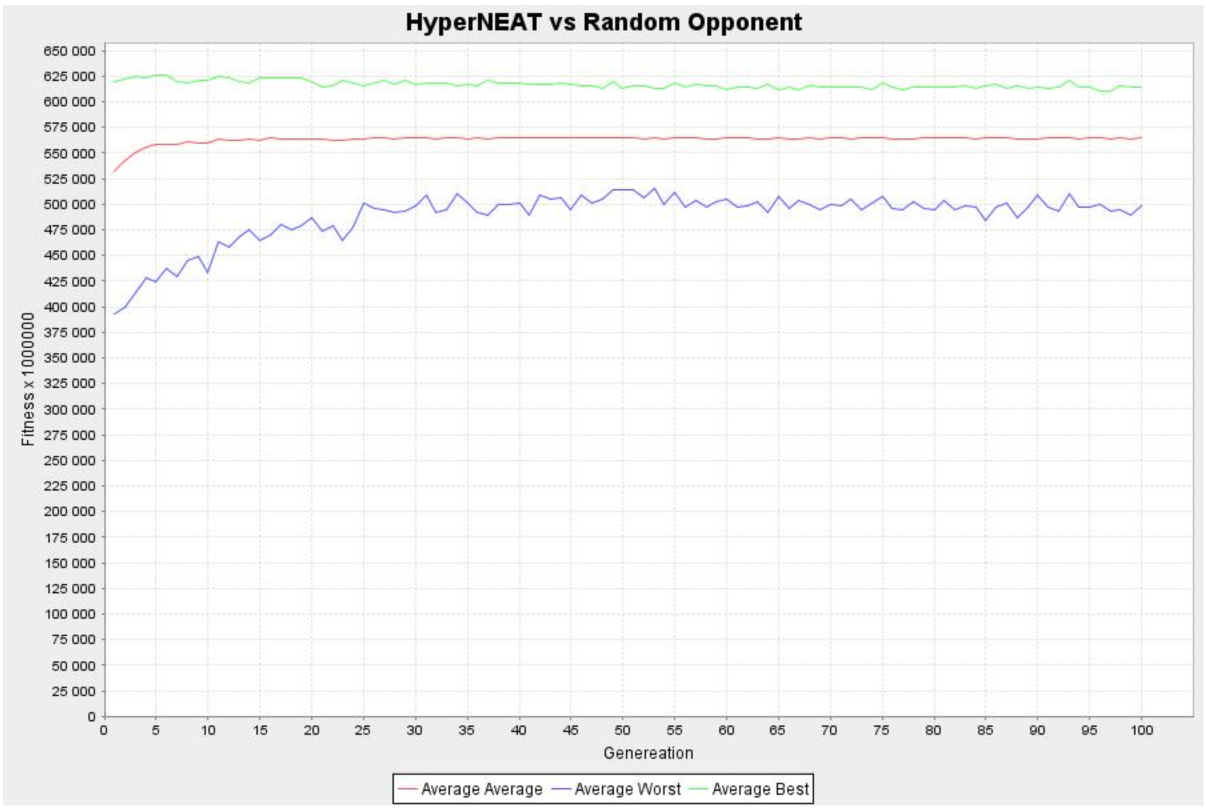


Figure 3.5: HyperNEAT vs Random Player
Averages generated through playing 100 games against the Random Player, gathered across 20 runs.

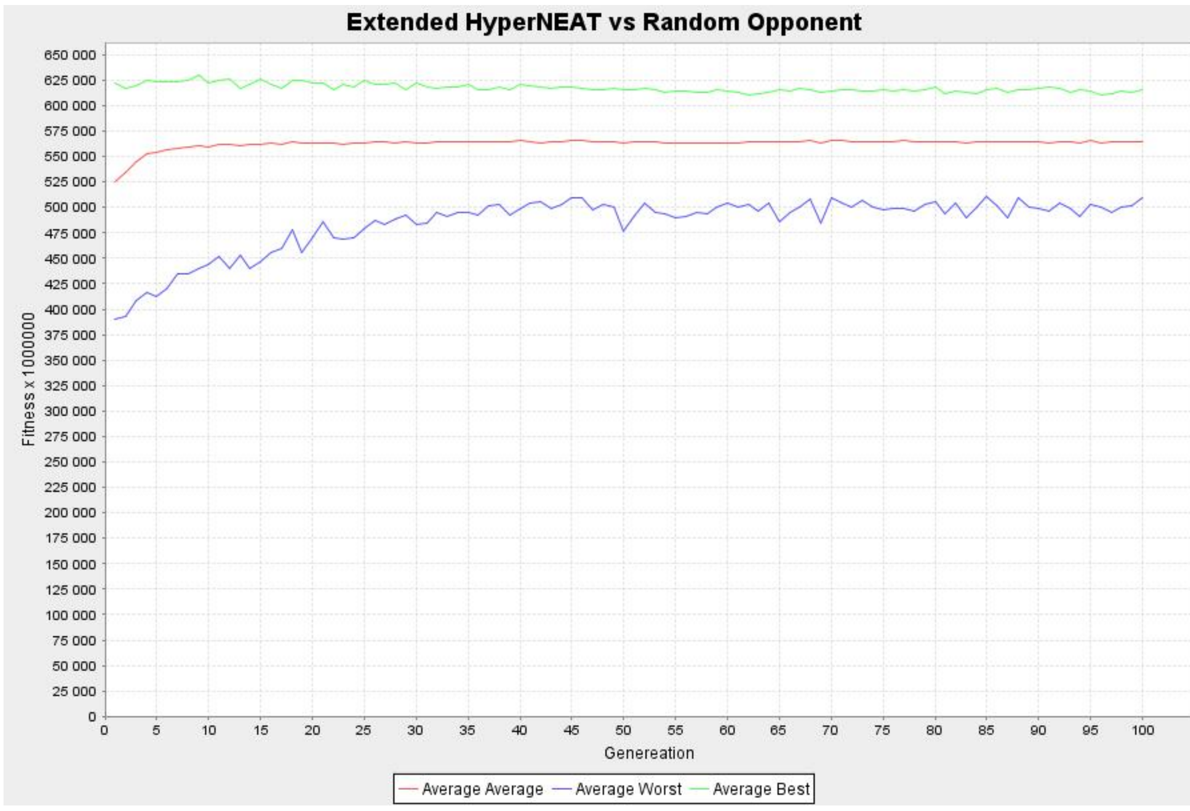
different structure would probably need more time/learning examples to achieve the same functionality. Granted, the limited effect could easily be attributed to the low learning rate, the limited number of pseudopatterns used, and/or the limited number of learning cycles. However, it was clear that the effect were more limited than in the case of the substrate with the non-nullable arcs. All in all, these results were considered indicative of it being wise to ensure the different substrates having the same structure.

3.2.2 Experiment 2: Pseudopattern-Supported HyperNEAT

Unlike in the previous experiment, the experiment was only performed with the arcs in the substrate being given a weight of 0.0 when below threshold. This was due to the results seen in experiment 1, which indicated that this enhanced the learning potential in comparison to when the arcs where set to null.

3.2.2.1 Results

The results are the average results across 20 runs.



Average positively influenced substrates pr. run: 3.3556
Peak positively influenced substrates in a single run: 7
Minimum positively influenced substrates in a single run: 0

Figure 3.6: Extended HyperNEAT vs Random Player

Pure HyperNEAT implementation vs Random Player Opponent See Figure 3.5 for the performance of the HyperNEAT generated individuals vs the Random Player, averages created over the total number of runs.

Extended HyperNEAT vs Random Player See Figure 3.6for the performance of the extended HyperNEAT individuals vs the Random Player, averages generated across all runs.

3.2.2.2 Evaluation

As noted in Section 3.1.3, both the runs with a pure HyperNEAT and the ones with the extended algorithm, used networks where weights below the threshold were set to 0.0, rather than being removed. Note that this should have no effect on the runs with the pure HyperNEAT implementation. Removing the links would have the same effect as setting their weights to 0.0. The only possible difference could have been a slight reduction in runtime complexity of the board evaluations, but that would have been a negligible improvement.

Pure HyperNEAT We can see that the pure HyperNEAT implementation did not perform spectacularly, as can be seen in Figure 3.5. While we can see that the average player did on average win against the random opponent, the populations appeared to converge to an average performance of roughly 0.562. Granted, given the fitness function used (see Section 3.1.1), it is entirely possible that this means that the average player won 100% of the time. However given the stochastic nature of the opponent, it is impossible to know after the fact, and given the low average, it is improbable. However, the goal of this experiment was not to create a particularly good Reversi player. The results gathered from these 20 runs were only meant for comparison.

Extended HyperNEAT The extended HyperNEAT algorithm did not perform significantly better than the HyperNEAT algorithm. Although the results seen in Figure 3.6, show that individuals did receive a positive effect from the learning process, it is not readily apparent from the graph that any increase has occurred. However, the fact that the application of pseudopatterns did in cases have a positive effect was a positive result, and the most interesting point to take away from this experiment. It was considered an indication of the thesis hypothesis being correct.

Furthermore, the fact that a subset of the individuals received a positive effect from the training process was taken to mean that the addition of an evaluation of the application process was an important addition to the approach presented in this thesis, and as such a successful refinement. Further experimentation with this and other refinements would be preferable.

Chapter 4

Evaluation and Conclusion

This chapter contains a summary, and a discussion of the results presented in Chapter 3. It also contains suggestions for future work, based on what has been presented in this thesis.

4.1 Summary

A literary search was performed. Based on this, a system utilizing back propagation with pseudopatterns to enable information transfer between separate individuals generated by an implementation of HyperNEAT, was designed and implemented. Two experiments were performed on the system, using the board game Reversi to test the system.

The first experiment was intended to lay a foundation for further experimentation. The purpose was to transfer information between individuals adapted to slightly different circumstances, through blindly applying pseudopatterns through back propagation. Information was transferred, though most often with destructive results. It was also, unsurprisingly, shown that networks of different structures received a lesser effect from the learning process. Based on the results, it was deemed that the approach presented in this thesis benefits from the different network having essentially the same structure.

The second experiment used pseudopatterns to transfer information across generations, applying pseudopatterns from the historically best individual to the best individual of each generation through back propagation. The change was discarded if it was not an improvement. A positive result from the learning process could be traced in a subset of the networks where it was attempted, and was taken to be an indication of the thesis hypothesis being correct. The subset was limited, which was taken to indicate that the refinement introduced in this experiment - a control of the results of the training process - was of great importance. The subset was of such a size that simply blindly applying pseudopatterns to a large population would probably, on average, have quite destructive results.

All in all, the experiments showed results that were considered indicative that the approach was promising. Information was transferred as expected, given a sufficient amount of pseudopatterns. However, some evaluation was necessary after application to deem whether it would be worth keeping the change.

It was found that a difference in network structure would counteract the effect of the pseudopatterns, as would be expected, however it did not appear that a limited difference in structure had catastrophic results.

4.2 Discussion and Conclusion

The first experiment was judged to show that the method used was viable to certain degree, but required further refinements. The application of pseudopatterns through back propagation would of course have an effect. However more control of the pseudopatterns and their application was required. No practical way to evaluate the value of pseudopatterns a priori was found. As such, it was not apparent whether a pattern embodied the successful adaptations of the network from which it was generated, nor what about a particular network signified a successful adaptation. In other words, during the work for this thesis, the most efficient way to evaluate the usefulness of a pseudopattern was merely to use them for training, and compare the network pre and post training. If exactly what in the network was responsible for the networks success was known, then the evolutionary process would have been redundant after all, one could simply have created a network embodying the best of all. Without such knowledge, the only approach to the generation of pseudopatterns found was merely to generate them with random input. It was hoped that in sufficient numbers that would result in representative pseudopatterns. Finding other ways of evaluating pseudopatterns, possibly a priori, appears to be an interesting direction for future work, see Section 4.3.

The application of pseudopatterns was refined as stated in the previous paragraph, in the second experiment. It was shown that only a limited number of the networks trained gained an improvement in performance. However, only a small number of networks were given training. It seemed probable that if every individual was trained on previous well-performing individuals, the results could be improved. The question then would be if the improvement in the population would be sufficient to justify the increase in runtime complexity. An interesting direction for future work.

An interesting possible effect of training all individuals in a population would be encouraging networks more receptive to learning from other networks to be evolved. That would again result in a greater number of individuals receiving beneficial results from training. Attempting to evolve such receptability as a feature of neural networks would be an interesting direction for future work.

Given that a limited number of the networks given training showed improvement, it can be inferred that while the basic approach is viable, without the post-training evaluation of the individual the population would likely have suffered. In other words, it was debatable if it could be counted on a positive result on average, given blind application, even if applied to a large number of individuals. This means that some refinement such as post training evaluation of the networks is of great importance to the approach presented in this experiment. Of course, it is not given that the refinement present in the second experiment was the only viable one, but it seemed clear that some refinement is required. Blind application is not a reasonable option.

A weak point of the second experiment, was that the effect of teaching was exclusive to the substrate. In other words, the effect could not be seen in the chromosome. A consequence of the indirect encoding utilized in HyperNEAT. A consequence of this was that even if an individual received a positive effect from the applied pseudopatterns, this effect, and with it the synthesis adaptations, would not carry over to the next generation, unless the same chromosome should be shown to be the best individual and receive training once again. And then, given the random nature of the pseudopattern generation, there are no guarantees the results would be the same. This is something of an undesirable effect, as the combined adaptation may be of interest. In effect, the only way such an adaptation could be kept in its original form, was if the network became the new all time best, and as such stored in its entirety. However, it would not be a part of the evolutionary process. For further experimentation, it would be desirable to test methods other than this.

In conclusion, the results were deemed positive, indicating the thesis hypothesis being correct, but inconclusive. Further experimentation would be desirable.

4.3 Future Work

While there were some promising results seen in the experiments, they were limited in both number and scope. There are many more experiments that could be done. The results in this thesis can be used to create several refinements to experiments following along the line seen in Chapter 3.

Following from the results seen in Chapter 3, other experiments jump to mind.

In the experiments in this thesis, only one kind of opponent was used within each separate trial. Interesting results could be gathered by trying to transfer information between networks trained on opponents with wildly different strategies. Individuals could be trained on a deterministic opponent, and then be tested against a random player, or vice versa. The performance could then be compared to the performance of the same individuals after being trained with pseudopatterns gathered from individuals trained on the relevant opponent. Such an application would align with the basic motivation for this thesis (see Section 1.1), the desire to create a system capable of keeping knowledge of old tactics while adapting to new.

In the same vein, a player could be evolved to a certain point playing against a player exhibiting a particular behavior, and then cloned. The individual could then be evolved further, playing against an opponent with a radically different approach to the game. After achieving a certain fitness, it would then be interesting to see how it would perform against the opponent it was evolved against first, and if any improvement would result from applying pseudopatterns from the clone.

The results would as such likely say much about the viability of the hypothesis of this thesis.

The first experiment showed that information transfer between networks with different structures was less effective than between networks of equal structure. However, an effect was still seen, though weaker. It would be interesting to know more about the limiting effect, to see just how different two networks could be before the weakening effect become catastrophic.

In the second experiment, only the best individual of each generation was exposed to pseudopatterns from previous generations. An improvement would be to attempt to train all individuals of each generation. Of course, it would be important to evaluate whether the results would be worth the increase in runtime complexity.

Another interesting possible consequences of teaching every individual, would be the possibility that individuals that easily learn from previous individuals could be encouraged, again increasing the effect of learning on the process as a whole. Whether or not such teachability is something that can be evolved, was considered to be of great interest to explore further.

It was mentioned in Section 4.2, that during the work on this thesis, no way to evaluate the value of a pseudopattern to a network a priori had been found. If such a method could be found, the efficiency and effectiveness of the approach attempted here could be greatly increased. Such a method would mean being able to extract strictly beneficial features from previous networks, to be applied to new networks. This would likely result in neuroevolutionary algorithms utilizing learning exhibiting much faster hill climbing behavior.

Also seen in the second experiment, was that due to the indirect encoding utilized in HyperNEAT, changes in the substrate, through learning, were not reflected in the chromosome. The only way the changes could be carried over generations, as mentioned in 4.2, was when the individual proved to be the best adaptation across all generations. Methods should be explored to retain the changes in the chromosome. An obvious approach would be to link the applied pseudopatterns, or even the training substrate, to the chromosome. However, this seems wasteful in terms of computing resources, particularly as the storage requirements would increase explosively with every generations as more teaching was performed (trained on substrates trained on substrates etc). Another approach could be to store the difference in weights resulting of the training process along with the chromosome. This would also keep the storage requirements constant.

Changing the fundamental approach to the training could also be a possibility. In the approach taken in this thesis, all learning was performed on the level of the substrate. However, it is also possible to apply learning to the CPPN. However, the results may be more drastic. As the result would affect the function defining weights of the substrate, rather than the weights themselves, the effect on the substrate could end up being quite different in different regions, rather than the more uniform effect expected with the use of back propagation in this thesis. Furthermore, it was noted in Section 3.1.2 that differences in structure between networks can have a negative effect on the learning process. Using learning with the CPPN, this would be a factor. The CPPNs' structure is subject to mutation. As is of course the activation functions of the individual nodes when following the HyperNEAT standard. Utilizing learning with CPPN could have very unpredictable results, and would likely require extensive experimentation.

In this thesis, Reversi was chosen as a test bed for the system. As we could see in the second experiment, neither HyperNEAT or the extended algorithm performed especially well, developing AIs for Reversi. It would be interesting to take the principles of the system in this thesis, and apply them to other games, or problem fields, such as driving, or pole balancing problems. It would be interesting to see if the results achieved with Reversi is transferable to other problems. Any problem where the same basic principles are retained, while the challenges faced can still change, would benefit from any successful application of what was attempted here.

Bibliography

JFreeChart, 2009. URL http://www.jfree.org/jfreechart/. 3.2

The Gathering, 2010. URL www.gathering.org. 2.1.1

Bernard Ans, Stéphane Rousset, R.M. French, and Serban Musca. Preventing catastrophic interference in multiple-sequence learning using coupled reverberating Elman networks. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society*. Citeseer, 2002. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2607&amp;rep=rep1&amp;type=pdf. 2.2.4, 2.3.1, 3.1.3, 3.2.1.2

Bernard Ans, Stephane Rousset, Robert French, and Serban Musca. Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting. *Connection Science*, 16(2):71–99, June 2004. ISSN 0954-0091. doi: 10.1080/09540090412331271199. URL http://www.informaworld.com/openurl?genre=article&doi=10.1080/09540090412331271199&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3. 2.2.4, 3.1.2, 3.1.3, 3.2.1.2

Anders Bondehagen. Submerged Temple, 2010. URL http://www.gathering.org/tg10/no/creative/compos/hardcore-programming-compo/compo-case/. 2.1.1

Bruno Bouzy and Tristan Cazenave. Computer Go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001. 2.1.1

Arthur E. Bryson and Yu-Chi Ho. *Applied optimal control : optimization, estimation, and control.* 1969. 2.2.1.1

Peter Burrow and Simon M. Lucas. Evolution versus Temporal Difference Learning for learning to play Ms. Pac-Man. *2009 IEEE Symposium on Computational Intelligence and Games*, pages 53–60, September 2009. doi: 10.1109/CIG.2009.5286495. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5286495. 2.2.3

Xindi Cai, Ganesh K. Venayagamoorthy, and Donald C. Wunsch. Evolutionary swarm neural network game engine for Capture Go. *Neural Networks : the official journal of the International Neural Network Society*, 23(2):295–305, March 2010. ISSN 1879-2782. doi: 10.1016/j.neunet.2009.11.001. URL http://dx.doi.org/10.1016/j.neunet.2009.11.001. 2.1.1, 2.2.3.1

Robert Callan. *The Essence of Neural Networks.* Pearson Education Limited, 1999. 2.2.1, 2.2.1.1

Darryl Charles and S. McGlinchey. The past, present and future of artificial neural networks in digital games. In *Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton*, pages 163–169. Citeseer, 2004. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.9367&amp;rep=rep1&amp;type=pdf`. 2.2.1

Siang Y. Chong, Mei K. Tan, and Jonathon D. White. Observing the Evolution fo Neural Networks Learning to Play the Game of Othello. *IEEE Transactions on Evolutionary Computation*, 9(3), 2005. 2.2.3.1

Jeff Clune, Charles Ofria, and Robert T. Pennock. The Sensitivity of HyperNEAT to Different Geometric Representations of a Problem. 2009. 2.2.3.3

B. Farley and W. Clark. Simulation of Self-Organizing Systems by Digital Computer. *Information Theory, IRE Professional Group on*, 4(4), 1954. 2.2.1

Even Bruvik Frø yen. *Continually Developing AI for Computer Games Using Evolutionary Techniques (unpublished).* Specialisation project, Norwegian University of Technology and Science, 2010. 1.1, 1.2, 2, 2.1.1

Jason Gauci and Kenneth O Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation*, 22(7):1860–98, July 2010a. ISSN 1530-888X. doi: 10.1162/neco.2010.06-09-1042. URL `http://www.ncbi.nlm.nih.gov/pubmed/20235822`. (document), 2.9, 2.3.1, 3.1.2

Jason Gauci and Kenneth O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation*, 22(7):1860–98, July 2010b. ISSN 1530-888X. doi: 10.1162/neco.2010.06-09-1042. URL `http://www.ncbi.nlm.nih.gov/pubmed/20235822`. (document), 2.6, 2.7, 2.2.3.3, 2.2.3.3, 2.3.1.1, 3.1.2

John Henry Holland. Adaptation in natural and artificial system. 1975. 2.2.2.1

Derek James and Philip Tucker. ANJI, Another NEAT Java Implementation, 2005. URL `http://anji.sourceforge.net/`. 2.3.1.1

D. Johnson and J. Wiles. Computer games with intelligence. *10th IEEE International Conference on Fuzzy Systems. (Cat. No.01CH37297)*, pages 1355–1358, 2001. doi: 10.1109/FUZZ.2001.1008909. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1008909`. 2.2.1

S.M. Lucas. Cellz: a simple dynamic game for testing evolutionary algorithms. *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, pages 1007–1014, 2004. doi: 10.1109/CEC.2004.1330972. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1330972`. 2.2.2, 2.2.3, 2.2.3.1

Melanie Mitchell. *An Introduction to Genetic Algorithms.* Massachusetts Institute of Tecnology, 1996. ISBN 0-262-13316-4. 2.2.2.1

Tom Michael Mitchell. *Machine Learning.* McGraw-Hill Book Co, 1997. ISBN 0-07-115467-1. 2.2.1.1

David E. Moriarty and Risto Miikkulainen. Forming Neural Networks Through Efficient and Adaptive Coevolution. *Evolutionary Computation*, 5(4), 1998. 2.2.2.1, 2.2.3.1

Joseph Reisinger and Risto Miikkulainen. Acquiring evolvability through adaptive representations. *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO*, page 1045, 2007. doi: 10.1145/1276958.1277164. URL `http://portal.acm.org/citation.cfm?doid=1276958.1277164`. (document), 2.2.3, 2.2.3.1, 2.4, 2.2.3.2

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. 1986. 2.2.1.1

Stuart Russel and Peter Norwig. *Artificial Intelligence A Modern Approach.* Pearson education International, second edi edition, 2003. ISBN 0-13-080302-2. 1.1, 2.2.1, 2.2.5, 2.2.5

Min Shi. An Empirical Comparison of Evolution and Coevolution for Designing Artificial Neural Network Game Players. 2008. 2.2.3, 2.2.3.1

Kenneth O Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmented Topologies. *Evolutionary Computation*, 10(2), 2002. (document), 2.2.3.2, 2.5, 2.2.3.2, 2.2.3.3, 3.1.2

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, January 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202. URL `http://www.ncbi.nlm.nih.gov/pubmed/19199382`. 2.2.3.3, 2.3.1

J. Togelius and S.M. Lucas. Evolving Controllers for Simulated Car Racing. *2005 IEEE Congress on Evolutionary Computation*, pages 1906–1913, 2005. doi: 10.1109/CEC.2005.1554920. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1554920`. 2.2.3.1

J. Togelius and S.M. Lucas. Evolving robust and specialized car racing skills. *2006 IEEE International Conference on Evolutionary Computation*, pages 1187–1194, 2006. doi: 10.1109/CEC.2006.1688444. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1688444`. 2.2.3, 2.2.3.1

Petter Westby. *Evolution of Artificial Neural Networks for Othello Board State Evaluation.* Master thesis, Norwegian University of Science and Technology, 2011. 2.2.3, 2.2.3.1

X. Yao. A review of evolutionary artificial nerual networks. *International Journal of Intelligent Systems*, 1993. 2.2.3

Chapter 5

Appendices

5.1 Legal Notes

ANJI v2.0 used under the GNU Public License.

JFreeChart 1.0.13 used under GNU Lesser General Public License.