Learning Othello using Cooperative and Competitive Neuroevolution

Bas Jacobs

INF/SCR-07-50

Master Thesis
Utrecht, Februari 29, 2008

Utrecht University, the Netherlands
Department of Information and Computing Sciences
Master Agents and Computational Intelligence

Supervisor:
Dr. M.A. Wiering (Utrecht University)

Committee:
Dr. M.A. Wiering (Utrecht University)
Dr. F.P.M. Dignum (Utrecht University)

Preface

In 2004, after having just finished my bachelor of computing sciences at the Hogeschool van Utrecht, I started my master Agents and Computational Intelligence at Utrecht University. From early age I was always fascinated by what computers were capable of doing. During the study computing sciences a course was given called Artificial Intelligence, which inspired me into continuing my studies at Utrecht University. During this course it was clear this was the right choice for me, and machine learning in combination with games was my favorite subject. After finishing my last course in November 2006, I started solicitations at various companies like Getronics, TNO and ING. Unfortunately none had the perfect assignment for me. This assignment preferably being one in machine learning. During this time I had also spoken with Dr. Marco Wiering about possibilities on doing research at the university. Exploring the possibilities of doing a research topic on Evolutionary Algorithms and Reinforcement Learning. During this time it came to my attention Michiel Vuurboom was also in search of a suitable research topic. In the past Michiel Vuurboom and I have successfully done courses and projects in both reinforcement learning and other related courses. After talking to Dr. Wiering we decided to do our master thesis together on the subject neuroevolution. Earlier on, in the course Reinforcement Learning, Michiel Vuurboom, Wouter Tinus, Tina Mioch and myself had done research on using reinforcement learning in the environment of Othello. Realizing Othello is a very good learning environment we decided to use this research as a starting point for our master thesis. After starting research at April 2007 on several learning techniques using Othello, my research for my master thesis has finished at February 2008.

During this time a lot of people have been of great help and support and I would like to take the opportunity to thank those wonderful people. First of all I would like to thank Michiel Vuurboom for being my classmate, friend and research partner during my time at the university. His inputs, knowledge and motivation have helped this research to what it is now. I would like to thank Dr. Marco Wiering for being an inspiring teacher, for his creative input and feedback during the project, and for him being ever supportive during the course of the project.

Abstract

From early days in computing, making computers play games like chess and Othello with a high level of skill has been a challenging and, lately, rewarding task. As computing power becomes increasingly more powerful, more and more complex learning techniques are employed to allow computers to learn different tasks. Games, however, remain a challenging and exciting domain for testing new techniques and comparing existing ones due to the clearly defined and easily enforced rules, complexity of games and often being fully observable and deterministic. In this thesis we will focus on the game Othello. Othello (also known as Reversi) is an old boardgame being played all over the world by new players and grandmasters alike. Othello is known for being a game which is very easy to learn but hard to master. Due to this nature of the game it is excellent for comparing existing techniques and testing new ones. Nowadays software exists which plays better Othello than the current human world champion. This software is capable of providing such a high level of play by using hard coded knowledge of the game (opening book), look ahead (mid game) and brute force calculations (end game). The goal of this research is to compare techniques in creating a player without the use of any such a priori knowledge. We intend to compare several neuroevolution techniques to random moving players and the more common reinforcement learning techniques of temporal difference learning.
Research done for this thesis can be divided into two sections: Comparison between the three neuroevolution techniques, and a comparison between cooperative and competitive learning. For part I three different neuroevolution techniques are compared: SANE, ESP and NEAT. All three use a neural network as function approximator, which is evolved using one of the three techniques. A comparison is done against random moving opponents as well as deterministic (and more skilled) opponents. NEAT emerged as best at learning how to play Othello. Part II is research done to explore the usability of different tournament types for evolving in a competitive way rather than cooperative. Using competitive learning rather than cooperative results in less games needed for evaluation of the same number of players, however information is lost as players pair against other, possibly unskilled, players. ESP is used as neuroevolution-technique. First a standard group tournament is used to test the capabilities of basic tournament training in Othello. Training is done using only learning players in the tournament as well as random moving players and deterministic players. The latter two are added to provide more knowledge into the tournaments. As basic tournaments resulted in less skillful players than was the case with cooperative learning, a more sophisticated tournament type was used: Swiss pairing. Swiss pairing does result in better learning, although still less than with cooperative. Tournament training results in less skilled players than is the case with cooperative learning.

Contents

Chapter 1

Introduction

1.1 Research motivation

Game playing has always been an interesting part in the field of Artificial Intelligence. Games like chess and Othello have been subjects of research for many years. These games are interesting because of their complexity and their many possible game states while they are also fully observable and deterministic. And because of a well defined set of rules they are easy to implement.
There are many human players who play these games at an excellent level, although since a few years the computer beats man in these games. In 1997 the chess program Deep Blue defeated the world champion for the first time [Hsu02]. In 1980 an Othello program called The Moor won a game against the reigning world champion and after 1997 world class Othello players are no match for Othello computer players [Bur97].
The current techniques to create a great computer player for chess or Othello use a priori knowledge of good strategies and use advanced search techniques to look ahead as many moves as possible. Of course a computer can look ahead much further than a human being and as computer power increases, this gap between computers and humans will also increase.
Although computers can beat man playing chess and Othello, that does not mean these computer players are intelligent. They are fast, they have a very large memory, but they use strategies that they did not invent themselves; they use human knowledge combined with computation power.
What if we can create a computer player that can learn playing games like Othello and chess without a priori knowledge about good strategies? Will they be able to learn to play a good competing level? Will they be able to learn the same good strategies as humans?
In 2005 we, Michiel Vuurboom and I, made a start with research on learning to play the game Othello without a priori knowledge using reinforcement learning and neural networks resulting in a player who plays at a decent level. Based on this result we do further research on the game Othello and this time using neuroevolution techniques.

1.2 Problem statement

Neuroevolution techniques seem promising, although at this moment there has not been that much research done on game playing such as Go and Othello using these techniques. Games like Othello with large state spaces and simple rules are interesting subjects for research on neuroevolution techniques.
With a good neuroevolution technique as a start we think it must be possible to learn to play Othello at a good level without any a priori knowledge. But to be sure of that, we first need a good neuroevolution technique.

Therefore our shared research goal for this research is:
Find the neuroevolution technique that is best at learning to play Othello.
For this we research some of the known neuroevolution techniques and find out which techniques have potential and then compare them to find out which one is best at learning to play Othello. We will compare the techniques in terms of learning potential and learning speed.

As became apparent during the running of experiments for part I, the learning time needed to create a skilled player capable of defeating the random and deterministic temporal difference player was still high and only good against the type of player it trained against. This is why for part 2 I want to do research on different learning approaches in combination with neuroevolution. More precise I would like to determine if competitive learning, rather than cooperative, increases the speed at which the same level of play can be reached.
In addition to our shared research which is found in *chapters 2*, *3* and *4* an additional part is added covering the research done into competitive neuroevolution using the Othello domain. This can be formulated as:
Can learning speed in neuroevolution be increased by using a different learning method while maintaining the same level of play?

For part I we expect all the techniques we compare to perform at least as good as the player created in 2005 at playing against a random opponent. One problem with the research in 2005 is that there is a bug in the software (explained in *chapter 6*), so the results of that research are not that good to compare to.
Expectations for part II are that competitive learning results in a shorter learning time required than cooperative learning.

1.3 Structure of the thesis

This thesis is split into two parts. The first part is a collective research about the comparison of the three neuroevolution techniques. In this part the game Othello is explained as well as the different strategies and the different opponents used in the experiments in *chapter 2*. The three neuroevolution techniques are explained in detail in *chapter 3* including their implementations used for the experiments. The experiments are described in *chapter 4* including all results and first conclusion.
Part II of this thesis covers the research done into competitive neuroevolution which is done by myself. First different competitive learning methods are explained in *chapter 5*. Following is *chapter 6* showing results of players learning against a randomly moving opponent using a limited round-robin and a Swiss pairing tournament. In the *chapter 7* experiments are shown where players learn to play Othello against a deterministic opponent using the same limited round-robin and swiss pairing tournament method.

Chapter 8 and *chapter 9* contain the conclusions and recommendations of both Part I and Part II.

1.4 Division of tasks

This thesis is divided into two separate parts. Part I is research done by both Michiel Vuurboom and me. The experiments done, implementations made, conclusions and recommendations made are done together.
Part II is research done by Bas Jacobs, and as such all implementations, experiments and written report are my own work.

Bas Jacobs

29 Februari 2008

Part I

Neuroevolution and Othello

Chapter 2

Othello

Othello is a derivative of the the Go family of games, and existed since the nineteenth century under the name Reversi. In 1974 the game Othello was formalized in Japan. Like Go, the game Othello is about capturing territory of your opponent. It is a two-player game on an 8x8 board with black and whites pieces. The initial board setup is shown in *figure 2.1*.

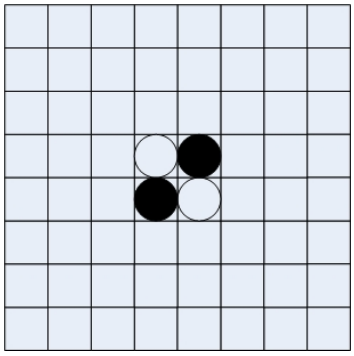

Figure 2.1: Initial board state

Each player takes turns placing pieces on the board. A player may only move to an open space that causes an opponent's piece or pieces to be flanked by the new piece and another one of the player's own pieces. The opponent's pieces are then captured. Pieces may be captured vertically, horizontally and diagonally. *Figure 2.2* shows the legal moves for black for the given board pattern.

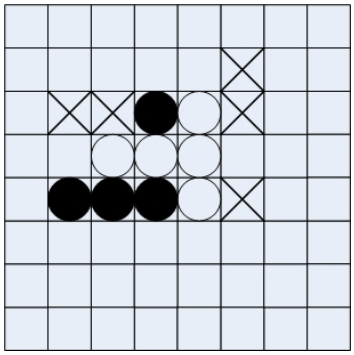

Figure 2.2: Legal moves for black

Once a move is made the captured pieces are flipped. *Figure 2.3* shows the board layout resulting from a move by black in the second row of the sixth column. The game is continued until there are no legal moves available for either player. If a player has no legal move he has to pass. The winner is the player with most pieces in the final board configuration.

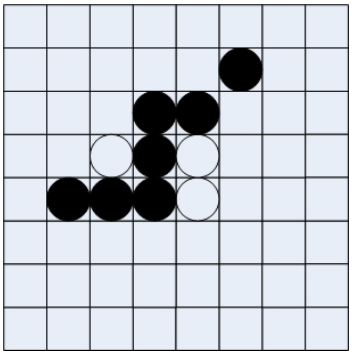

Figure 2.3: After black's turn

2.1 Playing Othello

There are different strategies for playing Othello. A player can focus on capturing the corners and edges, or he can use a strategy where capturing corners is only a sub-goal. The best human players use very sophisticated strategies and they try to look as many moves ahead as possible.

An Othello game can be divided in an opening-game, a mid-game and an end-game. Of course the end-game (approximately the last 20 moves) is the final battle in which each player tries to capture as many stones as possible. The foundation for that end-game is the mid-game. The first moves in the opening-game are important for a good mid-game. Two general classes of mid-game strategies exist in Othello: the *positional strategy* and the *mobility strategy*.

The positional strategy is simpler than the mobility strategy, but also inferior. Using a positional strategy the player has the immediate goal to capture as many stones as possible. To accomplish that, he will try to capture the edges to ring the opponent and he will try to capture the corner places at any given time because a piece in a corner can never be captured. A game with two players using a positional strategy will end up in an arms race with both players trying to get the upper hand.
The positional strategy is an easy strategy to understand and to learn and it is also easy to implement in a computer player. Most new and novice players use this strategy.

The mobility strategy on the other hand is much more complicated, but also superior. It is based on the notion of mobility: forcing the opponent to give up available moves until the player is in a position to decide exactly where the opponent will have to move. To

accomplish this the player attempts to control the center of the board, forcing the opponent to surround the player's pieces. The opponent will be forced to surrender corners and edges in the end game because of what the player does in the mid-game. A mobility strategy can be characterized by a low piece count and a large number of available moves for the player during the mid-game. Then the opponent will have many pieces and only a few available moves.
The mobility strategy is difficult to learn. Not only for a human player is this hard to master, but it is also difficult to make a computer learn this technique [Bil90].

2.2 Computer players

Human players can only look a few moves ahead. Computer programs can compute many steps ahead, only limited by their memory capacity and speed (or the amount of time one is willing to wait for the calculations to complete). It should be stated that expert human players do not scan any more moves ahead than novice players [Gro65]. Most Othello programs use a priori knowledge for playing the game. They use an opening-book for the first few moves and use complex search algorithms in combination with different strategies to decide the next move.
By now computers are fast enough to compute many steps ahead and beat the best human players in playing Othello. But they can beat human players only because of their computing power. Search-algorithms have been developed and evolved to very fast algorithms and many strategies for playing Othello have been developed in the last 40 years [Rus95].

Current expert computers players, like WZebra [And04] and Edax [Del04], can beat all human players. They use sophisticated pattern recognition and a notion of mobility to play a mobility strategy. An opening book (a large database with opening moves and their desirability) is used for the opening-game. Then an advanced search tree is used to play the mid-game using mobility strategy. The end-game is usually played by calculating the last moves (up to the last 20 moves). By calculating the end-game, the computer already knows at the end of the mid-game whether it can win the game or not. It will then try to maximize its score, or minimize its losses using look ahead to the last move of the game.
Players like WZebra, which is one of the best computer players at the moment (and free to download and use), use advanced a priori knowledge and well defined strategies combined with brute force computation to play the game. They are optimized to play Othello with the mobility strategy. All knowledge was implemented and nothing was learned.
It would be interesting to find out whether it is possible to have a computer learn the mobility strategy. Some research has been done on this subject.
In [And02] the authors claim to have developed an Othello learning player that is capable of learning a mobility strategy using the neuroevolution technique NEAT. Although the player does not play at an expert level, it was able to learn the mobility strategy.

2.3 Othello opponents

For this research several two different opponents have been created to test against. They consist of a non-deterministic and a deterministic player.

2.3.1 Random opponent

The Random opponent is a very simple player. It does not use any strategy or board evaluation at all and just picks a random move from the possible legal moves.
In *chapter 6* it is explained that the Random opponent used in earlier research ([Jac05]), where this research is based upon, contained a bug that has been fixed for this research.

2.3.2 TD-Greedy opponent

The TD-Greedy opponent is the final result of the research in [Jac05]. This opponent has learned to play Othello using random opponents combined with batch-learning using sample data from world class tournament games.
In the end it scored 83% against a random opponent and is a good opponent for novice Othello players.
This player is deterministic and it uses a neural network with 20 hidden neurons for the board evaluation. Details of this research and the player can be found in [Jac05].

Chapter 3

Neuroevolution

Creating a computer based Othello player can be done in quite a few different ways, although creating an expert one requires a priori knowledge and a lot of raw computing power. Many leaps have been made in the development of techniques capable of learning to play Othello. One of the more recent ones is the development of neuroevolution. Neuroevolution is a technique which uses genetic algorithms to train artificial neural networks.

3.1 Neural networks

An Artificial Neural Network, also known as Neural Network, is a processing unit based upon the principles of biological information processing performed by the brain [Ste96]. Key components are the neurons and the connections between them. These neurons are linked in a specific manner depending on the task a neural network is assigned to. A neural network is designed to learn by example and through examples its connections are updated in order to generate better solutions to the problem presented. A typical neural network can be seen in *figure 3.1*. The Input Layer is where information is fed to the neural network; the Output Layer gives the outcome of the neural network based upon the inputs given; The Hidden Layer allows for more complicated tasks to be learned. More hidden units and layers allow for more complicated tasks to be learned at the expense of computation time.

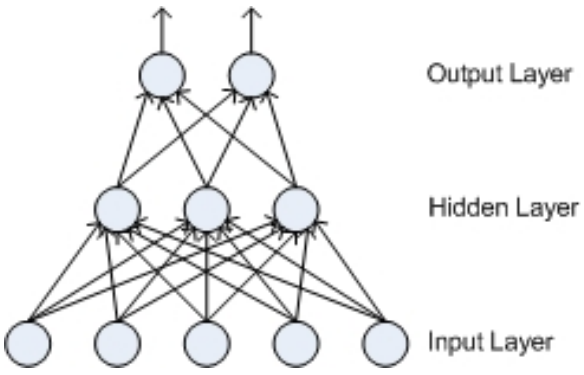

Figure 3.1: Simple feedforward neural network

Neural networks have been used in many fields such as sales forecasting, industrial process control, customer research, data validation, risk management, target marketing and more recently the gaming industry. Neural networks allow a system to map a domain state to a desired action and they are capable of generalizing over states easily which is most welcome in areas where a huge amount of states are possible.
Some of the advantages and disadvantages of neural networks as described in [Vel99] are:

Advantages:

1. Neural networks are able to learn any complex non-linear mapping / approximate any continuous function.
2. As non-parametric methods, neural networks do not make a priori assumptions about the distribution of the data / input-output mapping function.
3. Neural networks are very flexible with respect to incomplete, missing and noisy data / neural networks are fault tolerant.
4. Neural network models can be easily updated / are suitable for dynamic environments.
5. Neural networks overcome some limitations of other statistical methods, while generalizing them.
6. Hidden nodes, in feed-forward supervised neural network models can be regarded as latent / unobservable variables.
7. Neural networks can be implemented in parallel hardware, increasing their accuracy and learning speed.
8. Neural networks performance can be highly automated, minimizing human involvement.
9. Neural networks are specially suited to tackle problems in non-conservative domains.

Disadvantages:

1. Neural networks lack theoretical background concerning explanatory capabilities / neural networks as black boxes.
2. The selection of the Network topology and its parameters lacks theoretical background / It is still a trial and error matter.
3. Neural networks learning process can be very time consuming.
4. Neural networks can overfit the training data, becoming useless in terms of generalization.
5. There is no explicit set of rules to select a suitable neural network paradigm / learning algorithm.
6. Neural networks are too dependent on the quality / amount of data available.
7. Neural networks can get stuck in local minima / narrow valleys during the training process.
8. Neural network techniques are still rapidly evolving and they are not reliable / robust enough yet.
9. Neural networks lack classical statistical properties. Confidence intervals and hypothesis testing are not available.

For the disadvantages 2, 3, 7 a solution was found in the development of neuroevolution techniques.

3.2 Genetic algorithms

Genetic Algorithms (GA's) is a population-based stochastic search algorithm based on the mechanics of natural evolution. GA's are a subset of Evolutionary Computing and are used to find approximate or exact solutions to optimization and search problems. Such applications are commonly presented by research fields such as biogenetics, physics, computer science, economics, engineering, chemistry and mathematics. For this research GA's will be used to find solutions for playing Othello.
GA's are based on, but not limited to the characteristics of natural evolutionary systems. GA evolution has 5 distinct characteristics:

1. Structures, which are a genetic representation of the solution domain
2. Structures are combined to form new, better solutions
3. Structures compete for a limited resource
4. Fitness function to evaluate a solution
5. Relative production success depends on the environment

Structures are complete individuals and can act in a given environment in order to determine their ability to execute the given task. The value given for this ability is called the fitness. Fitness is a single number given to each structure as a performance measure. The fitness is determined by doing 1 trial per structure in the environment if both the structure and the environment are deterministic, or multiple trials if randomness is involved.
A structure consists of genes to describe its characteristics. These genes combined are called chromosome or genotype. The structure of these chromosomes is manually designed. *Figure 3.2* shows a representation of a simple binary chromosome. Chromosomes can be merged to create offspring which has characteristics of 2 parents (or more). This way important characteristics of successful parents can be passed on to offspring to create better solutions. In addition to merging the genes of parents, mutation is also used to maintain genetic diversity.

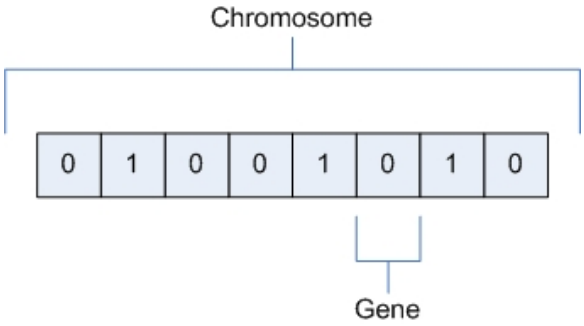

Figure 3.2: Representation of a binary chromosome

As a genotype, it is in most cases not possible to determine the fitness of the structure. Just like in nature, a genome itself is mere data, but with the data a creature can be created.

This is called a phenotype. The phenotype is used to determine the fitness of the genotype. According to this fitness a sorted list is created and individuals are selected for reproduction. Parents can be selected any way one chooses, although some selection techniques are more popular than others. Examples are Fitness Proportionate Selection, Tournament Selection and Ranked Based Selection.
Figure 3.3 shows a general genetic algorithm flow. First an initial population(*P(0)*) is generated; often with random values but predetermined topology. Secondly *P(0)* is evaluated to be able to select parents. Next the while loop is started to compute new generations. In this loop the parents(*P'(t)*) are selected for recombination which produce offspring(*P"(t)*). Now we have too many individuals, and so in this example, *P'(t)* ∪ *P"(t)* need to be sorted on fitness after which the best are kept as a new generation *P(t+1)*.

```
P(0)          ←  Generate initial population()
P(0)          ←  Evaluate population(P(0))
t             ←  0
While             Not-Terminated P(t)
do
  P'(t)       ←  Select mates(P(t))
  P"(t)       ←  Generate offspring(P'(t))
  P"(t)       ←  Evaluate population(P"(t))
  P(t+1)      ←  Select fittest(P"(t) ∪ P'(t))
  t           ←  t + 1
return P(t)
```

Figure 3.3: Genetic Algorithm Pseudo-Code

3.3 Neuroevolution

Neuroevolution is a technique where GA's are used to improve neural networks. There are many neuroevolution techniques, which can be classified in techniques which evolve the neural network weights versus techniques which evolve both the weights and the topology of the neural network. GA's which evolve both the neural networks weights and topology are also called TWEANNs (Topology & Weight Evolving Artificial Neural Networks).
When GA's are used to evolve neural networks, the network (which is a phenotype) has to be converted to a genotype to be able to reproduce. Weight values can be stored in a chromosome in different ways; *direct encoding* and *indirect encoding*. Direct encoding means having floating point values in the chromosome representing all weights. Indirect encoding can be determined by the developer.
Several neuroevolution techniques exist and for this research SANE, ESP and NEAT are compared.

3.3.1 SANE

Symbiotic, Adaptive Neuroevolution (SANE) [Mor96], [Mor97], is a reinforcement learning method which evolves a population of neurons through genetic algorithms to form a neural

network. Evolving a population of neurons instead of full neural networks makes it possible to develop partial solutions to the posed problem. *Figure 3.4* shows the basic steps for computing one generation in SANE.
The goal of SANE is to have each individual develop a solution which can be combined with others to form a complete and effective solution to the problem. Because individuals alone can not make an effective solution, symbiotic relations must be maintained with other individuals. When creating a full neural network, neurons are chosen from the population pool and combined to form a complete neural network. *Figure 3.5* shows the conversion between a genotype and its phenotype.

1. Clear all fitness values from each neuron.
2. Select ζ neurons randomly from the population.
3. Create a neural network from the selected neurons.
4. Evaluate the network in the given task.
5. Add the network's score to each selected neuron's fitness variable.
6. Repeat steps 2-5 a sufficient number of times.
7. Get each neuron's average fitness score by dividing its total fitness value by the number of networks in which it was implemented.
8. Perform crossover operations on the population based on the average fitness value of each neuron.

Figure 3.4: One generation in SANE

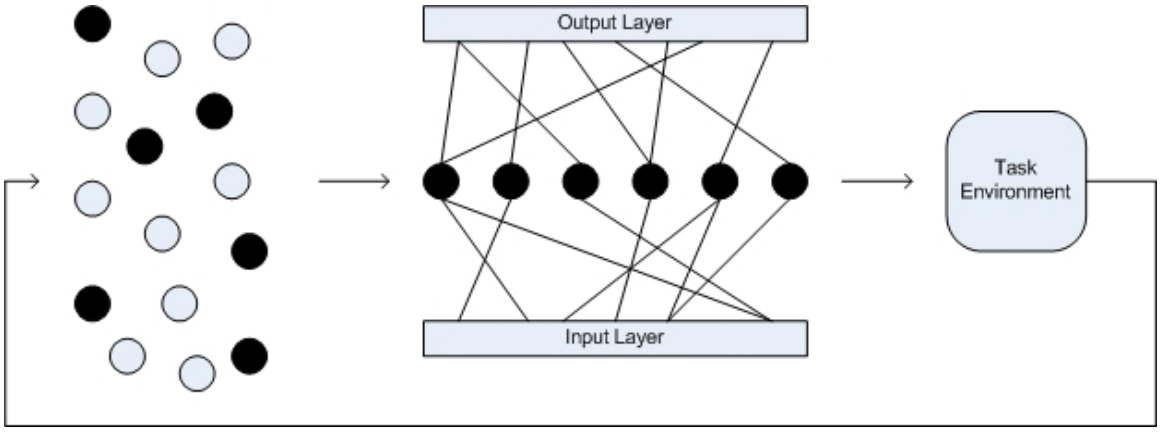

Figure 3.5: SANE, Genotype to Phenotype

Fitness can easily be determined for all individual neurons after having determined the fitness for the formed neural network. When an individual participates in a neural network, the fitness of the neural network is equally assigned to all participating neurons. This way an individual can take part in any number of neural networks. In theory all neurons should participate in neural networks with all other neurons to get an optimal weighed fitness for each neuron. Obviously this is not feasible and fitness will remain an approximation.
Specialization is an important aspect of SANE and is possible due to the individual neurons being evolved. Instead of solving the entire problem, individual neurons aim to solve a

particular aspect of the problem. Specialization is enforced knowing individual neurons cannot form a complete solution and the fitness of the neurons is based on the effectiveness of symbiotic relations it has with other neurons. Specialization prevents converging to suboptimal solutions because of the diversity in the population.

3.3.2 ESP

Enforced Sub-Populations (ESP) [Gom99], is a reinforcement learning method very similar to SANE. ESP evolves a population of neurons through genetic algorithms to form a neural network. Like SANE, ESP evolves a population of neurons, but unlike SANE, the specializations are not kept in a single parent pool. A drawback with SANE is the interbreeding of different specializations which result in a lot of individuals with similar characteristics as well as very few to no protection of new (still weak) species. ESP enforces a subpopulation for each hidden neuron of the neural network as can be seen in *figure 3.6*. Neurons in a subpopulation can only recombine with neurons from its own subpopulation. These enforced subpopulations allow a much faster specialization than is the case with SANE (where all specializations have to emerge from one large pool). Having subpopulations protects weaker species from dominant ones taking over the population. Also having neurons being placed at the same location in the neural network, and being linked to the same neurons increases learning speed and allows better learning for recurrent networks.

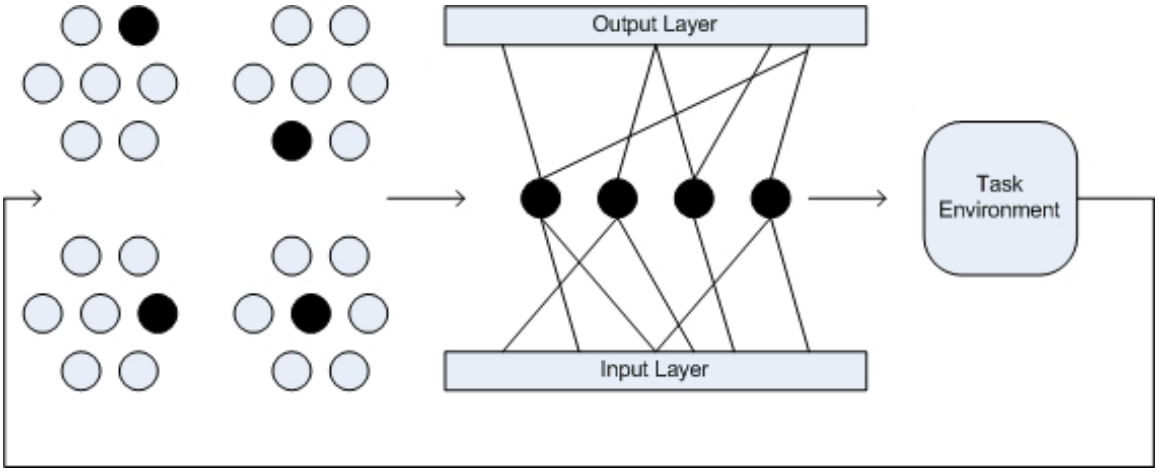

Figure 3.6: ESP, Genotype to Phenotype

3.3.3 NEAT

Neuro-Evolution of Augmenting Topologies (NEAT) [Sta02], is a reinforcement learning method, although not like SANE or ESP. NEAT is a TWEANN, a GA which evolves both weights and topology of neural networks. Like SANE and ESP weights are evolved through generations allowing a better solution to be reached. In addition, changes can be made to the topology in terms of links and nodes. This allows NEAT not only to search the search-space but also to minimize it during evolution.

The initial topology in NEAT can be setup by humans to fit the problem to be solved. Initial topologies more closely matching the optimal topology are recommended as it reduces the time required to evolve to the optimal topology. NEAT's topologies often become increasingly more complex as they become more optimal, strengthening the analogy between GA's and natural evolution.
NEAT uses direct encoding to describe network structures as indirect encoding would limit the topological search to the class of structures which would be designed for the indirect encoding. As NEAT evolves topologies, this is clearly an unwanted characteristic. *Figure 3.7* shows the genotype and phenotype of NEAT. The genotype consists of two types; node genes and connect genes. Node genes come in three types; sensor, hidden and output. Hidden nodes are removed or added through evolution. Connect genes represent the links/weights between nodes. A connect gene defines one link/weight between two specified nodes and can be enabled or disabled through mutation and crossover operators.
Specialization is also allowed by NEAT due to the historical markings assigned to each individual in the population.

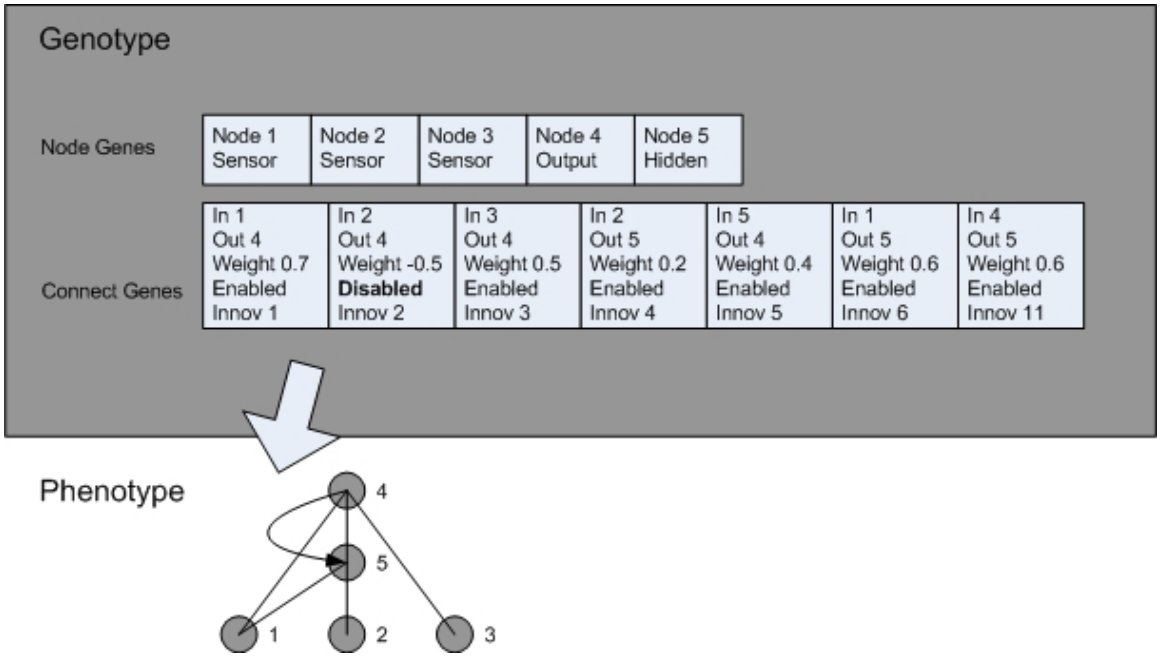

Figure 3.7: NEAT, Genotype to Phenotype

3.3.4 Other Neuroevolution Techniques

Besides SANE [Mor96], [Mor97], ESP [Gom99] and NEAT [Sta02] other neuroevolution techniques have been developed like TEAM [Ald02] and CoSyNe [Gom06]. The Eugenic Algorithm with Modeling (TEAM) is an extension of the evolution technique The Eugenic Algorithm (EuA [Pri98]). TEAM is a technique for evolving a population not only by standard crossover and mutation, but also by directing evolution. This is done by maintaining historical information on correlations between allele and fitness. The available software

contains code for evolving binary genes only. TEAM was not part of the experiments due to time constraints and remains an interesting technique to experiment with for creating Othello players.
Cooperative Synapse Neuro Evolution (CoSyNe) [Gom06] is another neuroevolution technique. CoSyNe searches at the level of individual network weights instead of neurons. Like ESP and NEAT, CoSyNe has n subpopulations, equal to the amount of network weights to be evolved. All subpopulations have an equal predetermined size. Phenotypes are created by selecting one individual from each subpopulation and inserting it at its position. Each subpopulation has individuals specifically for one position in the neural network. As CoSyNe was released after the choice of techniques, it was left out.

3.3.5 Neuroevolution and Games

There has not been much research in learning games like Othello with neuroevolution techniques. The three techniques discussed here have been used in control tasks like pole-balancing problems and similar tasks, but not in playing games.

As stated before NEAT has been used to learn a mobility strategy in Othello ([And02]). Although not like an expert player, it was able to learn mobility. The game Go has been studied using SANE and ESP in [Lub01] and in [Per01], but only for small Go boards (up to 7x7 positions). These examples show that there is still a lot of interesting research that can be done on neuroevolution and board games.

Chapter 4

Experiments

4.1 Implementation

To test the three neuroevolution techniques a Java implementation of Othello was used, along with Java implementations of the three techniques.
Java was used because of previous work on Othello [Jac05] was also done in Java, so an Othello framework ready for experiments already existed.
The three techniques were originally written in C or C++ but they all have a Java implementation as well.

4.1.1 Othello

The implementation has been divided into two main groups, Environment classes and Player classes. Environment classes contain code for the game itself, while Player classes contain (any form of) intelligence for playing a game of Othello using the Environment classes.
An abstract view of the environment implemented in Java is given in *figure 4.1*.

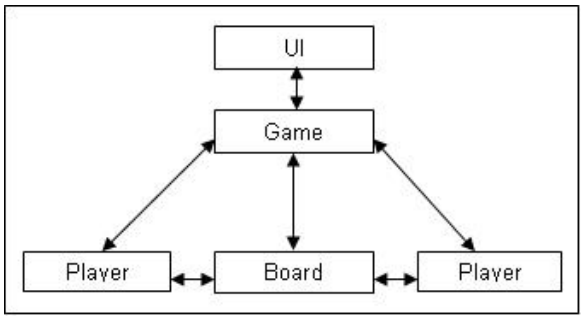

Figure 4.1: Othello Java classes

The environment has several features so it can be used for experiments, like keeping track of the scores, playing multiple games and several others.
The player classes are a collection of all implemented players and the Player interface. Each game consists of two players chosen from the available implemented players.
A collection of players used in previous work ([Jac05]) can be used as opponents for the neuroevolution techniques. There is a human player, which is in fact a user interface so a real human player can play Othello. There is also a random player who plays random moves. The last player used in this research is the player using the neural network that was learned in this previous research, the Temporal Difference player (TD player). It plays a good game against novice players and can beat random players 83% of the time without looking ahead.

During this research look-ahead was implemented for this TD-player, using first a min-max algorithm, and later on an alpha-beta search algorithm. When the TD-player looks ahead 3 moves or more it beats a random player 99% of the time.
Because the Othello environment is separated from the players, it is easy to add all kinds of different players. With this approach it was not difficult to add neuroevolution players using the code from other authors.
The neural networks used in the different techniques always consist of 64 input neurons and 1 output neuron. The number of hidden neurons and the way the neurons are connected varies.
The 64 input neurons represent the 64 board positions. The input is 1 if a board position contains a piece of the player, -1 if the position contains a piece of the opponent and 0 if the position is empty.
The output neuron is a sigmoid activation function and represents the evaluation value of the given board state.
The fitness function used in the genetic algorithms is a win percentage after playing 50 games. At the end of each generation, the best neural network is allowed to play 1000 games to set the fitness off the champion of that generation.

4.1.2 SANE

The JavaSANE package contains the source code for the Hierarchical SANE system, based on SANE-C by Moriarty, [Mor96], but rewritten in Java. This package is designed to be an easy starting point for applying JavaSANE to a new domain.
For this research it was just a matter of rewriting the fitness function so that it plays Othello to evaluate the neural network. Of course some parameters had to be tuned so the correct topology for the neural network was used.
Several features have been added to this package to make it easier to analyze the results and to monitor evolutionary progress.

4.1.3 ESP

The Java ESP package contains the source code for the Enforced Sup-Populations system which is nearly a direct port of the ESP C++ package that was used for research in [Gom99]. It supports different kinds of neural networks like Simple Recurrent Networks, Second Order Recurrent Networks and Fully Recurrent Networks, but for this research only simple feed forward networks are used.
With Java ESP it was also not difficult to add an Othello configuration and fitness function and integrate it in the Othello environment.

4.1.4 NEAT (JNeat and Anji)

In this research two Java implementation of NEAT have been tried, JNeat and Anji.
First JNeat was used for this research. JNEAT was written by Ugo Vierucci based on the original C++ package by Kenneth Stanley which was used in the NEAT research [Sta02]. JNeat is an extensive package with a user interface to monitor progress and it uses a complex object model. It was not that hard to integrate it with the Othello environment and create

the right fitness function, but it was difficult to add and change the code to make it easier for this research to monitor progress and analyze results. In fact this package was not really well-written and the Italian comments did not make things easier.
On top of all this it did not turn out to work that good. It never really learned anything despite all different approaches and configurations.
So JNeat was put aside and Anji was used instead. Anji did learn and was easy to implement. Anji was written by Derek James and Philip Tucker based on descriptions of NEAT in papers published by Kenneth Stanley and Risto Miikkulainen ([Sta02]). It was not directly based on the NEAT C++ package, so it probably differs in some aspects.
Anji is very well written and is easy to configure and adjust to make it ready for the Othello experiments.

4.2 Initial experiments

Before running the final large experiments several initial experiments have been performed to fine tune the different techniques. All three techniques have several parameters to tune like selection mechanisms for the genetic algorithm, the different mutation rates and several others. Also the knowledge representation of the Othello player, the neural network, had to be tuned to see what (initial) topology would work best.
To compare three different techniques, several parameters were set the same for all techniques, so it was possible to compare the outcome. These parameters include population size and number of games each individual can play. This way, the three techniques have the same number of evaluations.
Each experiment was allowed 500.000 evaluations. An evaluation is one game of Othello against a random playing opponent. A random opponent was selected as most skilled players require a lot of time per game. Also when playing against a skilled player, which is often deterministic, the evolutionary technique only learns a limited amount of states. Every generation consists of 100 individual neural networks and they all play 50 games of Othello each generation. This means each experiment took 100 generations. Each experiment was repeated 3 times. This should be enough to tune the parameters.

4.2.1 Tuning SANE

To tune SANE several experiments have been performed. The main focus for tuning SANE was finding out what neural network works best for SANE. To test this a fully connected network was tested with different numbers of hidden neurons to find out what network size would work best. Also different mutation rates were tested.

Testing network sizes: 20, 40 and 60 hidden neurons.
These networks have been tested with fully connected neurons with the default SANE parameter for the mutation rate which is 20%. Although the final results did not differ that much, except for the obvious difference in learning speed, it seemed that 40 hidden neurons had a better result in the long run. The number of hidden neurons, 40, may look a bit arbitrary, but it was not only chosen because of the outcome of this experiment, but also because of past experiences with Othello and neural networks and the expectations that 20

would be too few to learn and 60 too many to learn fast enough. The fact that the results showed not much difference between the network size, justifies an arbitrary choice.

Testing mutation rate: 20%, 40% and 60%
The mutation rate is the chance that the weight of a neuron is mutated every generation. The default SANE value is 20%. This value was tested against 40% and 60% with a fully connected network. The results of these experiments did not differ that much. It looks like 40% performs slightly better than the other two in the long run. At least it has a better average fitness in the top 10 in every generation in the end. The overall average fitness of the population is about the same in every generation for the different mutation rates.
Several other parameters were not tuned because they did not seem to matter that much or appeared fine with pre-initial experiments.

4.2.2 Tuning ESP

For ESP both the network size and the mutation rate have been tested. The first goal was to find out what size neural network works best with Othello. The second goal was to find out which mutation rate offers a good learning rate.

Testing network sizes: 20, 40 and 60 hidden neurons.
Fully connected networks were tested with default ESP parameter for mutation rate which is 40%. After 100 generations it was clear that a network with 20 hidden neurons does not perform as good as the ones with 40 and 60 hidden neurons. The ones with 40 and 60 hidden neurons do not differ that much. Both show a learning curve and have the same fitness in the end.
Because of SANE and other past experiments with Othello the 40 hidden neurons seems a good network size.

Testing mutation rate: 20%, 40% and 60%
With a fully connected neural network with 40 hidden neurons the different mutation rates were tested. After 100 generations it was clear that a mutation rate of 40% is the best. It has a fast learning rate and is still learning at the end of the experiment. Both 20% and 60% have a lower fitness in the end.
Several other parameter were not tuned because they did not seem to matter that much or appeared fine with pre-initial experiments.

4.2.3 Tuning NEAT

Tuning NEAT (with the Anji implementation in this research) took some more time. Pre-initial experiments made it clear that several parameters had to be tuned. For NEAT the parameters *weight mutation rate*, *connection mutation rate*, *neuron mutation rate* and the *speciation threshold* were tested.
The initial topology at the start of each experiment is a fully connected neural network with 10 hidden neurons.

Testing weight mutation rate: 0.005, 0.01, 0.1, 0.25, 0.5

The weight mutation rate is the probability of existing connection weights being mutated. Very low values were tested here. Because pre-initial tests showed that a low value for this was good, several tests had to be performed to see how low this value should be. In the end 0.01 turned out to be a good value.

Testing add connection mutation rate: 0.03, 0.06, 0.2, 0.3, 0.5
The *add connection mutation rate* is the probability of new connections being added with an initial weight with a random value from a normal distribution.
First the values 0.03, 0.06 and 0.2 were tested, but it turned out that 0.2 performed much better than 0.03 and 0.06. So apparently a higher value was needed. Therefore 0.2 was tested against even higher values, 0.3 and 0.5. The results of these tests did not differ that much. It seemed that 0.2 performed just a little better than the other two. At least it is still learning at a higher rate than the other two after 100 generations.

Testing add neuron mutation rate: 0.1, 0.02, 0.005, 0.001, 0.0005
The *add neuron mutation rate* is the probability of new nodes being added to an existing node in the neural network.
After the first tests with 0.1, 0.02 and 0.005 it turned out that 0.005 performed better than the other two, so 0.005 was tested against 0.001 and 0.0005. The mutation rate of 0.005 turned out to be the best because the other two values did not seem to learn anymore after 100 generations.

Testing speciation threshold: 0.1, 0.2, 0.4, 0.5, 0.6
The speciation threshold is the compatibility threshold used to determine whether two individuals belong to the same species.
After the first tests with values 0.1, 0.2 and 0.4 showed that 0.4 was the best value another test was done with 0.4 against 0.5 and 0.6. The results are close, so it is probably not necessary to do more test. The value 0.6 turned out to be the best value for the speciation threshold.

4.3 Final experiments

Final experiments were conducted once the tuning of the different neuroevolution algorithms was completed. Tuning was needed to assure maximum performance of all three techniques when comparing them in the Othello environment. The final experiments were done allowing a lot more evaluations per experiment than was the case with the initial experiments. Obviously this was done due to time constraints and shorter tuning experiments did set a trend to allow proper variable tuning. Each experiment was allowed 2.000.000 evaluations, and each experiment was repeated 10 times. One evaluation means one game of Othello. Each fitness measure counts as one evaluation and for each generation every individual was measured, even if the individual was already measured. Because individuals were trained against an opponent which makes random moves, the extra fitness measures means the assigned fitness will be more accurate.
It should be noted that the size of neural networks and thus the computation time was not taken into account.

4.3.1 Time

Running experiments takes time. All experiments are repeated 10 times and each individual experiment of each technique against the random opponent takes about 8 hours to complete. This means that repeating this 10 times will take 80 hours for each experiment against the random players.
The experiments against the deterministic opponent, as described in the last paragraphs of this chapter, take even more time, up to 10 hours for each individual experiment.

Because of the amount of time it takes to perform an experiment choices had to be made for which experiments are done and which are left out. Probably more and better results were possible if there was more time available. More on this in the last chapters.

4.4 SANE - Final experiments

4.4.1 Properties

As said before SANE was trained up to 2.000.000 evaluations per experiment. From the initial experiments the following parameters were found to be most optimal for learning Othello against a random moving opponent:

- Mutation rate = 0.4
- Neural network = Fully connected feed forward
- Networks created per generation = 100
- Number of hidden neurons = 40

4.4.2 Running Experiments

After running each SANE experiment ten times, it was clear SANE was performing on par with standard reinforcement learning techniques such as Temporal Difference learning; scoring 83% [Jac05].

As a performance measure for every generation the champion of the generation was allowed to play 1000 games in order to get a more accurate fitness. This can be seen in *figure 4.2*.

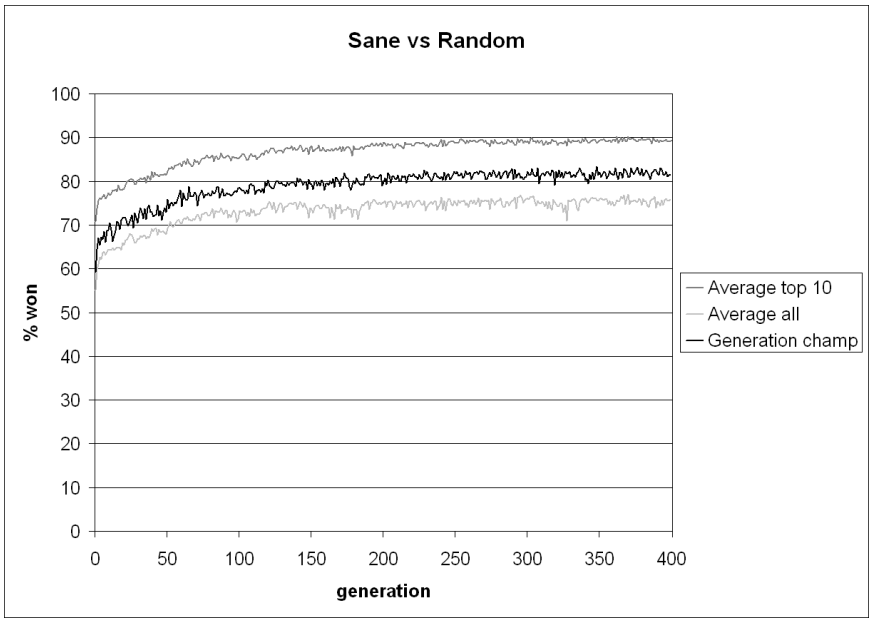

Figure 4.2: SANE, experimental results - 2000 neurons

SANE certainly learns properly, but not more than any default reinforcement learning technique (83%). Sane also ends up with a 82% - 83% score.
When comparing the *champion* to the win percentage of all the evaluated networks (*average all* in *figure 4.2*), it is clear the champion performs better than all the evaluated networks. Of course this was to be expected but seeing a gap of 6% wins suggests a large spread in fitness. Unfortunately this is a characteristic of SANE having all neurons in one large pool and thus allowing very bad networks to be formed as well. This might also be the reason for all of the ten experiments to have a rather large spread in win percentage of the champion at the end of each experiment (win percentages between 77% and 86%). The initial population size (2000) might also have been too large, resulting in new neurons being selected every trial and thus preventing proper learning.
There also is a difference of 8% - 9% between the *champion* percentage and the *average top 10* percentage. This is because the top 10 consists of the individuals with the 10 highest fitness scores and this fitness is determined after playing only 50 games. The best individual plays 1000 games and its fitness will be the champion fitness. This champion fitness is bound to be lower than the average top 10 fitness because of the inaccuracy of playing only 50 games compared to playing 1000 games.

Two more settings have been tested; 400 neurons (*Figure 4.3*) and 800 neurons (*Figure 4.4*) as population size to have each neuron participate in a network 10 times and 5 times on average respectively. As can be seen both population sizes of 400 and 800 performed slightly better than the initial population size of 2000. Unfortunately this is no real improvement to make SANE perform significantly better than it did with a population of 2000. Only the

experiment with 400 neurons results in a score higher (about 85%) than the original which suggests that maybe the neuron pool can be made even smaller for better results, although a score of 85% might be the top for SANE. No more experiments using SANE have been performed due to time constraints.
Although mutation rate might be high (40%), tuning tests showed this to not influence the evolution significantly.
Alternative selection and replacement methods could improve SANE's performance.

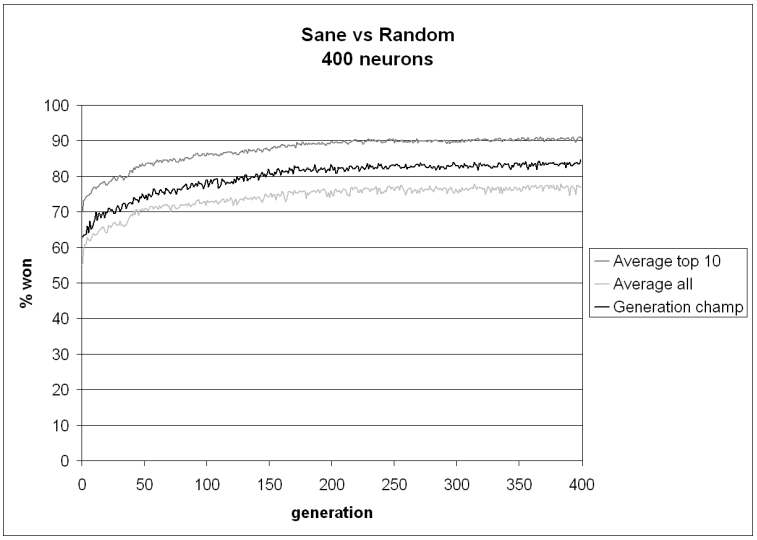

Figure 4.3: SANE, experimental results - 400 neurons

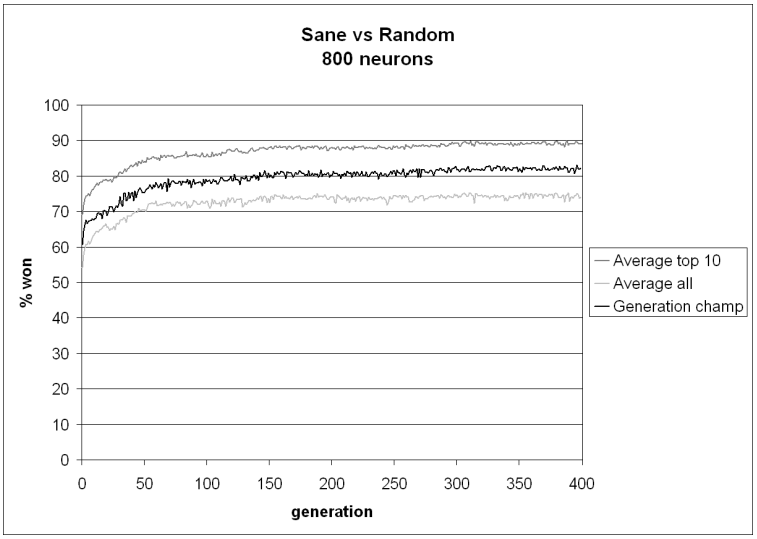

Figure 4.4: SANE, experimental results - 800 neurons

4.5 ESP - Final experiments

4.5.1 Properties

ESP was also trained up to 2.000.000 evaluations per experiment. The following parameters were considered most effective for ESP:

- Mutation rate = 0.4
- Delta coding = true & false
- Networks created per generation = 100
- Neural network = Fully connected feed forward
- Number of games per network = 50
- Number of hidden neurons = 40

4.5.2 Running Experiments

For ESP two different settings were used; delta coding and no delta coding. At first ten experiments were performed with delta coding enabled. Delta coding allows ESP to create more diversity when no new champion had been discovered for a while.

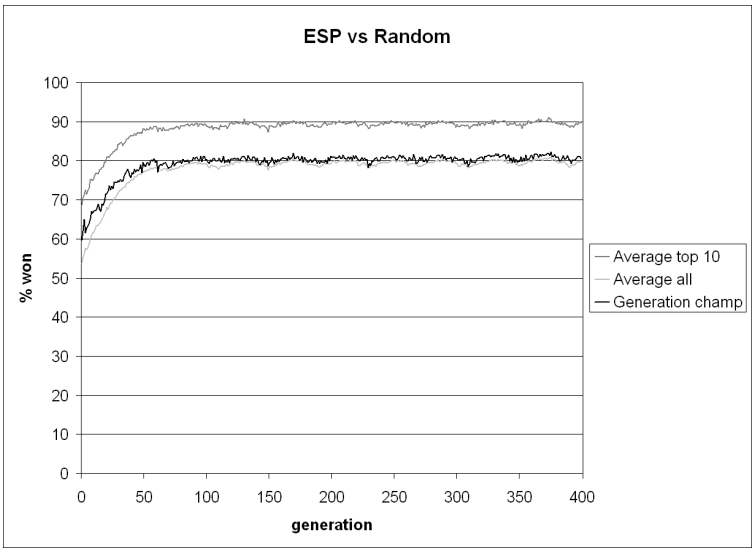

Figure 4.5: ESP with delta coding, experimental results

ESP with delta coding performs almost as good as the earlier experiments done with SANE. The first experiment was done using a subpopulation size of 100. *Figure 4.5* shows the results for ESP with delta coding. The champ in ESP reached 80%. The humps in the graph are times when delta coding is done. Unfortunately this prevented ESP from getting a score above 82%. More tests with setting the delta coding parameters might result in better performance. ESP with delta coding reached its maximum at generation 100.
Before a set of delta coding parameters was tried, delta coding was disabled. This resulted in much better performance which can be seen in *figure 4.6*. With delta coding disabled

ESP was allowed to continue evolving without the continuous setbacks resulting from delta coding. Without delta coding ESP learns much more smoothly and reaches the maximum obtained by ESP with delta coding at generation 70. Now this is nothing new, but without delta coding ESP continues evolving and has converged at generation 200 having a winning percentage of 87%.

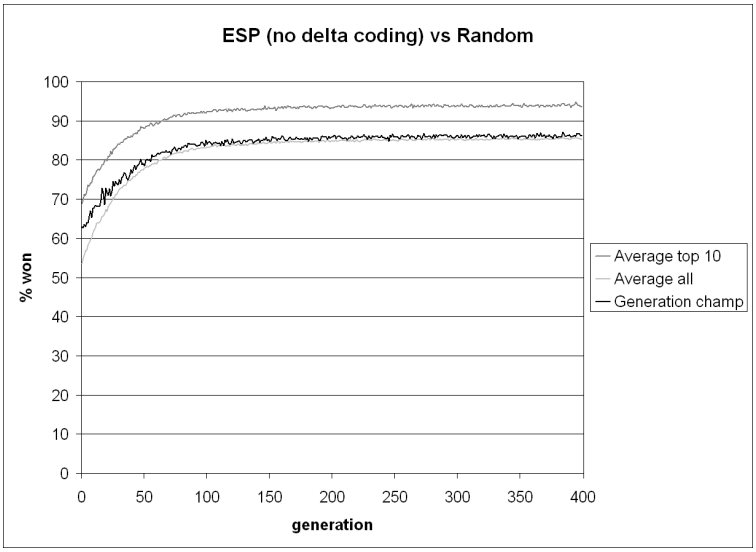

Figure 4.6: ESP without delta coding, experimental results

These first two experiments are performed with a subpopulation size of 100 and with 40 subpopulations (one for each hidden neuron). Which means that when creating a network in the population each neuron will be selected only once each generation.
In [Gom99] the neurons in the subpopulations are tested in different neural networks for a good evaluation of the neuron. That is why two extra experiments have been done with smaller subpopulation sizes. A subpopulation size of 10 and 20 were tested, so each neuron is evaluated 10 and 5 times respectively. No delta coding is used in these experiments.
The results are shown in *figure 4.7* and *figure 4.8*.

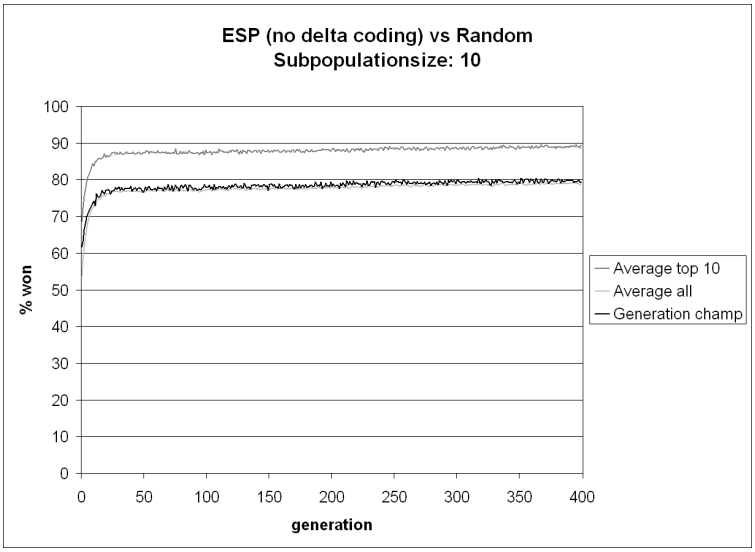

Figure 4.7: ESP subpopulation size of 10 without delta coding

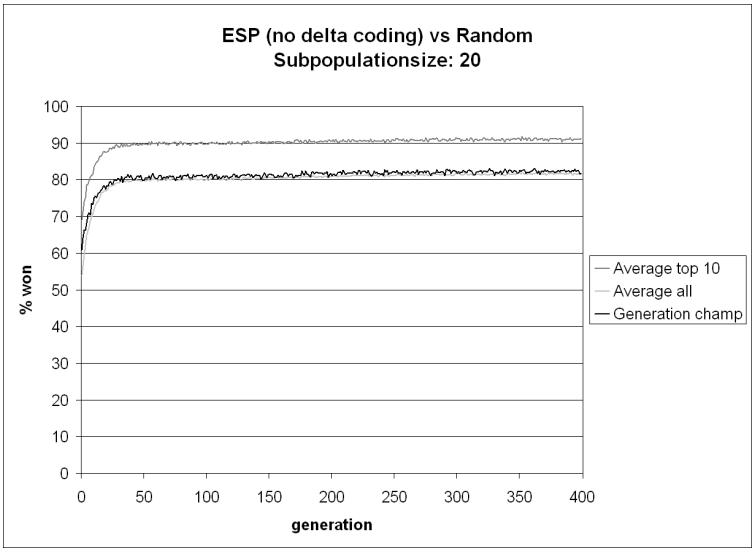

Figure 4.8: ESP subpopulation size of 20 without delta coding

The results show that when using smaller subpopulation sizes and thus evaluating a neuron more than once each generation, does not give better results. The endresults of the experiment with the subpopulation size of 100 are better.
However the learning speed is significantly higher in these last experiments. A maximum is reached around generation 50 instead of generation 150 with the larger subpopulation size. Perhaps there is some potential using smaller subpopulations which can be exploited by using delta coding to create more diversity after a stagnation of the learning speeds.
So several new experiments have been performed using the two different subpopulation size 10 and 20 and using two different stagnation values 40 and 100. This stagnation value is the number of generations in the experiments without improvements. So a stagnation value of 40 means that when there has not been a significant improvement over the last 40 generation, delta coding is used on the current population.
The results of these four new experiments are shown in *figure 4.9*, *figure 4.10*, *figure 4.11* and *figure 4.12*.

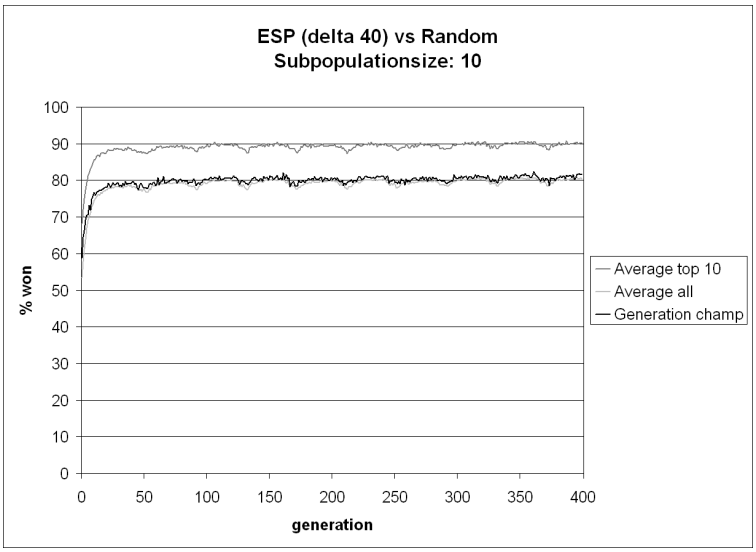

Figure 4.9: ESP, subpopulation size of 10, delta coding stagnation value 40

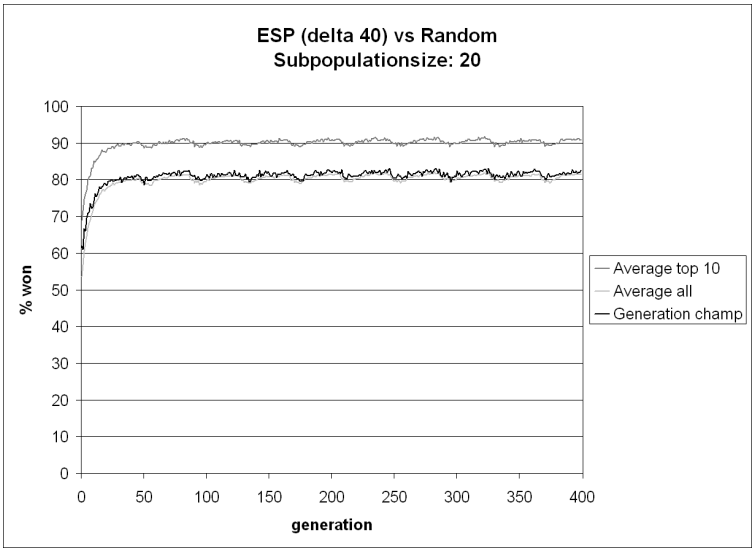

Figure 4.10: ESP, subpopulation size of 20, delta coding stagnation value 40

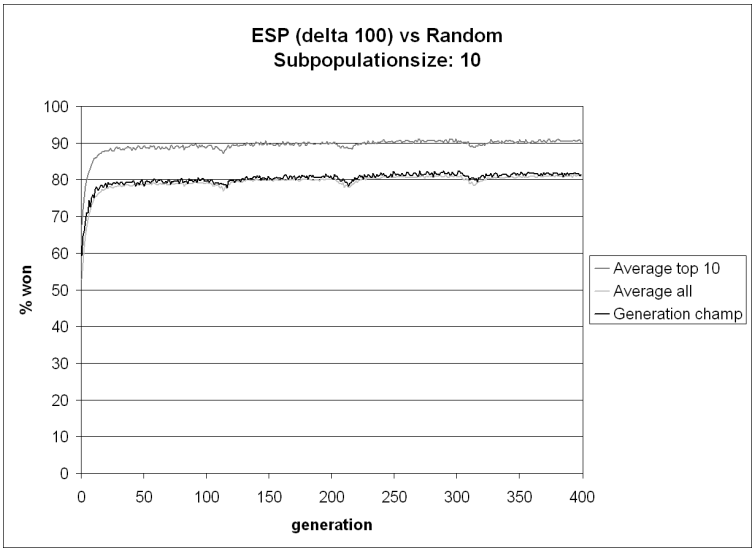

Figure 4.11: ESP, subpopulation size of 10, delta coding stagnation value 100

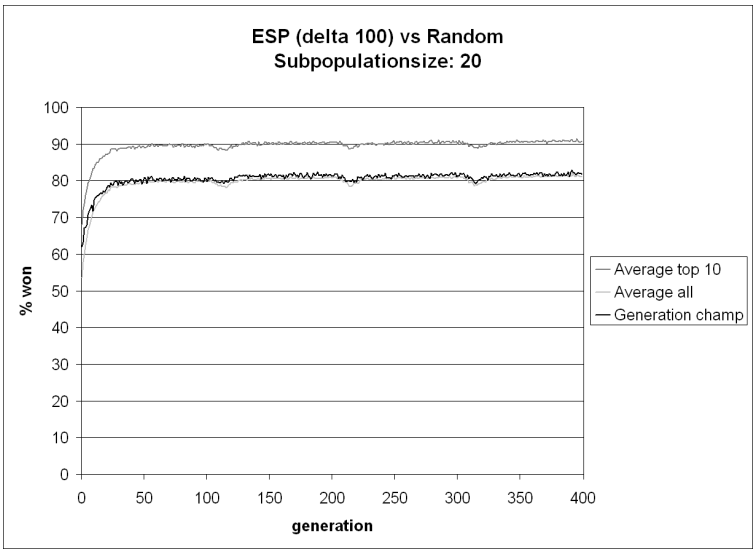

Figure 4.12: ESP, subpopulation size of 20, delta coding stagnation value 100

Comparing these results with the results without the delta coding show no real difference. In the end it does not seem to make a difference when delta coding is used or not for the smaller subpopulations. So ESP shows the best results when no delta coding is used and large subpopulation pools are used.

When comparing the *champion* percentage with the *average top 10* in the different graphs, the average top 10 score is higher than the score of the *generation champ*. This difference of 9 - 10% can be explained by the fact that the individuals are given fitness values based on 50 matches. After assigning the fitness there is a certain inaccuracy in the fitnesses assigned. After sorting the individuals, the ones with the highest (and also the most inaccurate) fitness make up the top 10. When having the best individual play 1000 matches as a more accurate fitness measure of that generation the *champion's* fitness is bound to be lower than the fitness of the *average top 10*.

The *champion* and *average all* show both graphs are closely together. The champion scoring better than the average of all evaluated individuals was something to be expected. The fact that both are close in win percentage shows that all evaluated individuals are not much spread out in win percentage. At the end of the ten experiments done with ESP without delta coding shows champions win percentage being close to one another. This, in contrary to SANE, shows ESP is much more reliable and stable in evolving.

ESP clearly performs better at Othello against a random opponent than SANE does. When using a subpopulatation size of 100 and no delta coding there is a difference between SANE and ESP of about 5%.

4.6 NEAT - Final experiments

4.6.1 Properties

As both other techniques, NEAT was also allowed to perform up to 2.000.000 evaluations per experiment. For NEAT the following parameters were used for the final experiments:

- Population size = 100 networks
- Add connection mutation rate = 0.2
- Add neuron mutation rate = 0.005
- Weight mutation rate = 0.01
- Speciation threshold = 0.6

4.6.2 Running Experiments

For NEAT ten experiments were done as well. *Figure 4.13* shows the results of these experiments.

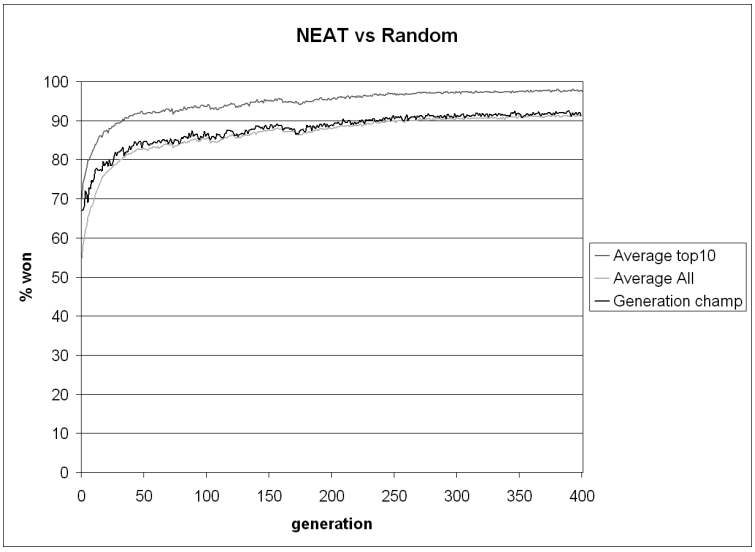

Figure 4.13: NEAT vs random, experimental results

NEAT reaches a win percentage of 90% at generation 200. NEAT does continue to improve to 92% in the end, and longer test runs might be needed to see how well NEAT can perform in the long run. Analysing the three graphs shows the *average top 10* to be at 98% wins, and the *champion's* win percentage to be slightly above the *average all* graph as is the case with ESP. Also at around generation 150 there is a dip in win percentage. This is the result of one of the ten experiments performing badly at that time.
Clearly NEAT performs better than both ESP and SANE.
Interesting to see were the amount of hidden units being evolved for the NEAT networks. The top 10 of individuals were started at 10 hidden units by default. The amount of hidden units was reduced to 2 and the amount of connections was reduced to 40-50 by the end of

the experiment while the win percentage kept rising. Apparently playing against a random playing opponent requires very few knowledge to be able to successfully beat it. This was confirmed when looking into the neural networks of the champions formed at the end of the experiments. Most champion networks had only two corner input nodes connected which suggests that a simple strategy that focuses on capturing some corners is enough to defeat a random opponent.

4.7 Deterministic opponents

Because of the few amount of hidden nodes and connections remaining in NEAT, two more additional experiments were done to test if this was indeed the result of training against a random moving opponent. Both NEAT and ESP were trained against the Temporal Difference (TD) Othello player which was created in [Jac05]. Because the goal of this experiment was to test NEAT, one other technique was needed (ESP) to verify the results from NEAT. ESP was chosen over SANE as ESP performed much better. The TD player was trained using a database of world class games and scored 83% against a random moving opponent.
The parameters for NEAT and ESP were the same as were used when training against a random moving opponent. For ESP delta coding was disabled for these tests. When having two deterministic players play multiple games of Othello, they will play the same games over and over. To avoid this, the first four moves of each player were made randomly resulting in 244 possible initial states.
Figure 4.14 shows the ESP results against the non-learning temporal difference opponent. The thing to notice is the more gentle learning curve than was the case when learning against a random moving opponent (*figure 4.6*).

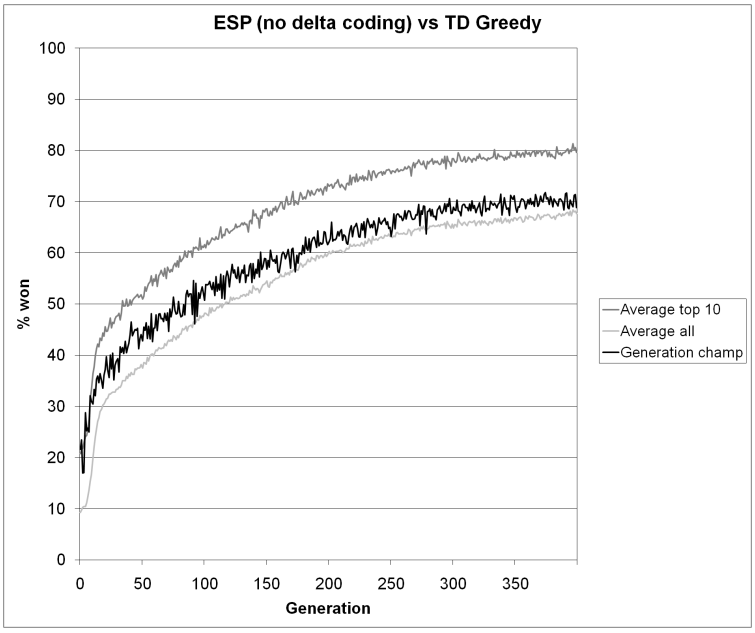

Figure 4.14: ESP vs TD, experimental results

Figure 4.15 shows NEAT's performance when training against the non-learning temporal difference opponent. When training against a more skilled opponent such as a TD opponent, 6 hidden neurons are needed along with 200 connections.
Also NEAT is more capable of learning against a TD opponent as can be seen in both figures. ESP has its *champion* and *average all* close together suggesting many similar individuals. NEAT has its *champion* and *average all* not close together suggesting a much higher degree of diversity present in the population.

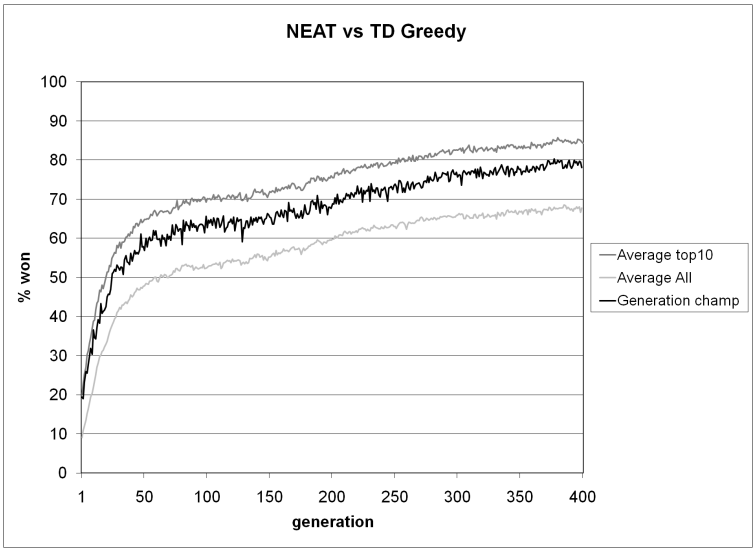

Figure 4.15: NEAT vs TD, experimental results

Part II

Neuroevolution: Cooperative and Competitive

Chapter 5

Tournaments

A tournament is an organized competition held between a large number of competitors. Many tournament types aim at providing a very reliable ranking of participants, while others aim at determining the best participant out of all competitors. This competition is often a sport or game where clearly defined rules apply.
Tournaments can be distinguished into two types: tournaments with a short time interval, and tournaments with a long time interval. Tournaments with a short time interval are generally held at a single venue. Many matches are played in this short time span. These types of matches are often used by chess, card games and ice skating.
Tournaments with a long time span are often held at different locations and more often last for months. Typical applications of this tournament type are soccer leagues and basketball leagues.
With both time spans, a large variety of ways to determine opponents (also called pairing) can be used. In the past many types of pairing have been used. All having their advantages and disadvantages. Here some of the more known tournament types are listed:

- Knockout
- Group
- Multi-Stage
- Promotion and Relegation

These types of tournaments all have their advantages as well as disadvantages and so they are explained in the following chapters.

5.1 Knockout tournaments

A well known tournament system such as the knockout tournament is often used in tournaments. A typical application of a knockout tournament is Wimbledon(www.wimbledon.org). Figure 5.1 is a typical knockout tournament roster.
In a knockout tournament players play one opponent in a fixture they've been seeded in. Winning players proceed to the next round, while losing players are removed from the tournament. This results in less players and fixtures remaining until only one fixture is left. The winner of this fixture is considered to be the tournament winner.
Knockout tournaments are commonly used in two versions: single-elimination and double-elimination tournaments.
In single-elimination, players who lose a match, even after an excellent streak thus far, are removed from the tournament. Only Single-elimination is used when there are more than

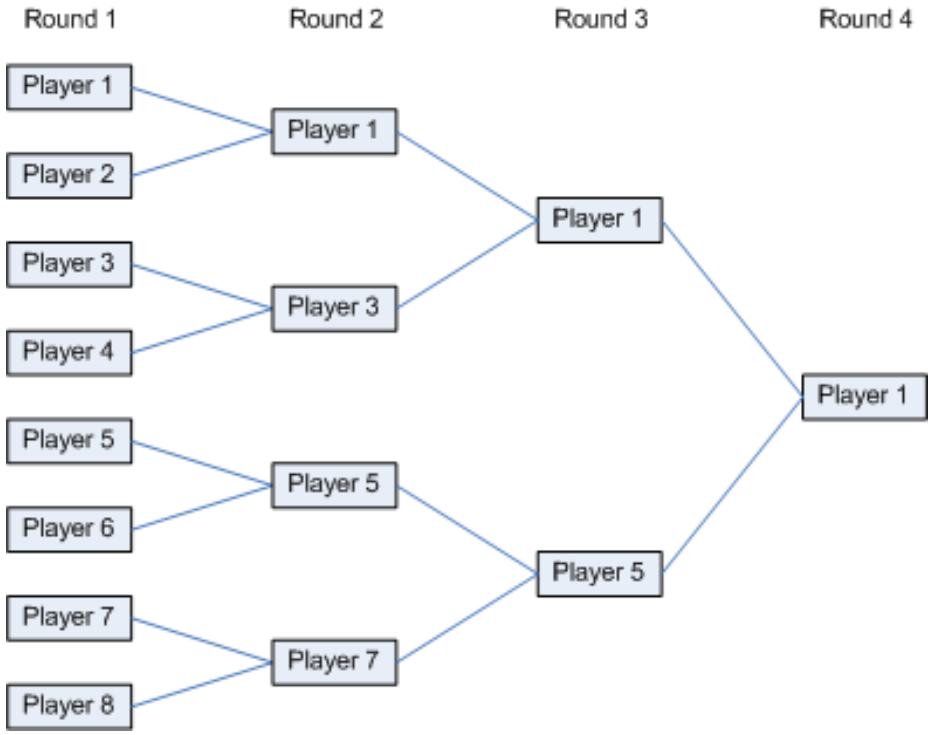

Figure 5.1: Typical knockout tournament

two competitors per fixture.
Double-elimination, can be used when fixtures consist of two competitors. Double-elimination allows a player a loss without being removed from the tournament.
A drawback from knockout is the fact players who meet a player who is exceptionally strong against their style of play can be removed from the tournament even though the player performs very well against all other players in the tournament. A typical tournament size of 16 players would require 4 rounds to be played, and 5 rounds would be needed to handle 32 players. In addition, players who did not pass the current round, are out of the tournament. The knockout tournament aims at determining the best player of the tournament, but fails to give a proper rank to the other players.

5.2 Group tournaments

A group tournament involves all participants to play a fixed number of matches, defined prior to the tournament. The tournament itself is divided into rounds, in which every participant plays its assigned match. During each round all matches are played and often the next round is only started once all matches have been finished. After each match, points are awarded to the participants of the match. These points are in this research 1 point for winning, 0.5 for drawing and 0 points for losing the match. A ranking is created after each round based upon the total number of points a participant has or the average points a participant has received over the course of the tournament. A ranking is created

not only for the purpose of knowing leading participants, but it is also often used to create new pairings for the next round. With group tournaments some common types can be identified:

- Round-robin
- Swiss

5.2.1 Round-robin

A round-robin tournament is a tournament requiring each competitor playing all participants (excluding itself) at least once. Round-robin tournaments have been used since old times to create a very reliable ranking or participants. Unfortunately with a large number of participants the required number of rounds and matches becomes huge and thus unfeasible.

For example, if 50 players would participate in the tournament, it would require 1225 games to be played in 49 rounds.
A variation that can be made to the standard round-robin tournament is having all participants play each other two or four times instead of the standard one. This is particularly useful when used in matches where participants can pick sides, e.g. black or white in a chess game. These types of round-robin are commonly used in soccer leagues such as the FIFA World Cup (www.fifa.com/worldcup) or chess tournaments such as the World Chess Championship 2007 (www.chessmexico.com).

5.2.2 Swiss

A Swiss tournament uses a special pairing method to pair players or teams that need to play each other. It was first used in 1985 at a chess tournament in Zurich - Switzerland, giving the algorithm its name. Since then it has been widely used in a large number of different tournaments such as chess, Othello, scrabble and bridge. Swiss pairing, used in a Swiss tournament, is based on the principle of pairing players with opponents of roughly the same strength. Like the round-robin and knockout tournaments, a Swiss tournament is divided into rounds in which players play their assigned opponent. Players who win their match are given 1 point, draws result in 0.5 point and a loss means 0 points. All players proceed to the next round of the tournament no matter if they lost or made a draw. The first round players are paired based upon knowledge of the strength of the players; this can be performance in previous tournaments or a guess based upon rating maintained in many application areas of Swiss pairing.
First, all players having the same score are grouped together in what is commonly called a scoregroup. When pairing the second round this could be three groups (1 point, 0.5 points and 0 points). In subsequent rounds the number of possible scoregroups increases with two. The second and subsequent rounds are determined based primarily upon the score of players. Pairing the scoregroups proceeds following a simple sequence. First a median scoregroup is determined. The median scoregroup is the scoregroup which consists of players having a score equal to half the number of rounds played. When there is no such scoregroup, the group with a score equal to half the number of rounds played with an offset

of 0.5 is used. Rounds/2 - offset or Rounds/2 + offset. Incrementing the offset with 0.5 if no matching scoregroup is found.
First the scoregroup with the highest score will be paired, pairing downwards until just before the median scoregroup is reached. Next the scoregroup with the lowest score will be paired pairing upwards until just before the median scoregroup. Finally the median scoregroup will be paired. Before the actual pairing starts, players are tested for available opponents in their scoregroups. Players who do not have a suitable opponent will be placed in their alternative scoregroup. This being the scoregroup below when pairing proceeds downwards, or above when pairing proceeds upwards. A number of requirements exist when testing for suitable opponents. These requirements can differ per application Swiss pairing is used for, but commonly include the rules:

- Competitors cannot play the same opponent twice
- It is not allowed to have a scoregroup with an odd number of players.

Once scoregroups have been made and necessary transfers have been done, players inside the scoregroups are sorted according to rating. Then the top half is paired with the bottom half of the group, called the proposed pairing. If the players in a scoregroup are numbered: 1, 2, 3 ... n, then the proposed pairings are:
1 v (n/2 + 1), 2 v (n/2 + 2), 3 v (n/2 + 3) ... n/2 v n.
For a scoregroup with six players this would result in the proposed pairing of
1 v 4
2 v 5
3 v 6

Once proposed pairings are done the pairs undergo another batch of tests. These will be the same tests as performed earlier, but performed on the proposed pairs. If a pair is found violating the set rules, the weaker player is exchanged with a different player until a valid pair is formed. These tests are done according to the rules:

- When pairing downwards the pair with the highest ranking player is tested first. If the pair violates the defined rules, the lower ranked played is exchanged until a valid pair is formed.
- When pairing upwards the pair with the lowest ranking player is tested first. If the pair violates the defined rules, the higher ranked player is exchanged until a valid pair is formed.

When we have a scoregroup containing six players [1...6] proposed pairing would be
1 v 4
2 v 5
3 v 6
Should 1v4 be invalid because player 1 already played against player 4, the following proposed pairing would be considered

1 v 5
2 v 4
3 v 6

Here player 1 would meet a player from its own scoregroup (player 5 being slightly weaker than player 4)), though still having the advantage of pairing a player from the bottom half of the scoregroup. Should the pairing 1v5 also be invalid, the next player is considered and so on. The complete sequence of proposed pairings for player 1 would be from left to right;

1	v	4	1	v	5	1	v	6	1	v	3	1	v	2
2	v	5	2	v	4	2	v	4	2	v	5	3	v	5
3	v	6	3	v	6	3	v	5	4	v	6	4	v	6

When a player can play different sides in a game, for instance play black or white in an Othello game, often special rules concerning color preferences are enforced. These rules usually forbid players from playing the same color three games in a row, or have a difference in played colors of three or more.
Advantages of the Swiss tournament system are:

- All players play all rounds (except for the bye some players may receive when having an odd number of players)
- The final ranking gives a relative ranking for all contestants of the tournament, not just the winner
- Less matches need to be played than in a round-robin tournament to determine a clear winner.

5.3 Multi-Stage tournaments

Multi-Stage tournaments are a tournament type much like knockout tournaments. Multi-Stage tournaments are divided into several stages, where each participant or team can progress to the next stage when performing well. Often there are many multi-stage tournaments held simultaneously serving as a qualifying tournament for access to the final tournament. During a stage of a multi-stage tournament the top team(s) progress to the next stage, and any type of tournament can be used for each stage. Common applications are the FIFA World Cup (www.fifa.com/worldcup), where first there are several qualifying stages before the final tournament starts with often 32 remaining teams. Other applications include the cricket world cup (http://icc-cricket.yahoo.com/icc-events/cricket-world-cup.html).

5.4 Promotion and relegation tournaments

Should a tournament have more competitors than a tournament format permits, it is often required to split the competitors into several sub-tournaments being held in parallel. Promotion and relegation tournaments are commonly used in sports throughout the world. Most soccer leagues use this kind of tournament where teams are promoted or relegated

once a year.
Players are grouped together into these sub-tournaments based upon ranking or playing strength. As a result tournaments with high skilled competitors or teams are formed as well as sub-tournaments with lower skilled competitors or teams. Each sub-tournament can have a different type of tournament, but usually these are the same for all. After all sub-tournaments have finished, top ranking competitors are moved from their current sub-tournament group to a higher group and are as such promoted. The lowest ranking competitors are relegated and are moved to a lower group. Once all competitors have been moved, each sub-tournament is started again.
Advantages of this type of tournaments are the multiple tournaments being held at the same time, reliable sorting of competitors throughout the groups, and maintaining importance of matches between lower ranking competitors near the end of the season. A disadvantage is the time required to sort all competitors.

5.5 Used tournament methods

Many types of tournaments exist for determining playing strength of players or teams, and all have their strengths and weaknesses. Obviously not all methods are suitable for learning a game such as Othello with neuroevolution. The goal of using tournament type learning is to test if competitive learning has an advantage over cooperative learning. Using competitive learning allows a different and possibly superior way of creating the ranking of individuals. An important requirement is to have every individual receive a reliable fitness measurement. This implies having all learning players play a sufficient amount of games. In Part I players were allowed to play 50 games against predesignated opponents in order to receive a reliable fitness. It is clear all players in the competitive learning environment need to have an equal amount of measurements in order to make a good comparison between the cooperative and competitive learning methods.
Tournament types such as knockout have the characteristic of removing players from the tournament after having lost a set number of matches. This results in only the highest players receiving a reliable fitness, and a lot of the lower players all being assigned the same fitness value. This may be desirable for determining what player or team is the most skilled, but this clearly is not wanted when measuring fitness.
Multi-stage tournaments are, like knockout tournaments, a good way to determine the best player of the competitors, but fails to assign proper ranking and fitness to the players who are knocked out of the tournament in early stages.
Promotion and relegation tournaments are among the best types of tournaments for assigning a reliable fitness to individuals or teams. However, a large number of matches would still be required in order to sort a division and this means having another tournament for each division. Often a round-robin tournament is used in order to sort each division. When using round-robin tournaments it is more advantageous to have multiple divisions using round-robin than having one large pool of players. Although this results in less matches being played it means having a fixed playerbase during the experiments as players move upwards and downwards through the divisions. Obviously one could also choose to hold a complete promotion and relegation tournament between each generation in which the neuroevolution technique learns, but this will again result in a large amount of matches

required in order to reach a reliable ranking. Hence promotion and relegation is unsuitable for these tournaments.

First a simple tournament type is used in order to test whether basic tournament play is advantageous in comparison to cooperative play. Basic tournament play means having a fixed number of rounds in which each competitor plays one assigned opponent. These assigned opponents are chosen at random. By playing sufficient opponents a reliable fitness should be created for each participant in the tournament.
The second type of tournament used will be the Swiss tournament type. Swiss tournaments have the advantage of having one large pool of competitors rather than divisions. But the main advantages of Swiss pairing are, as described in *section 5.2*, having each participant playing an equal amount of matches and requiring less matches to be played in total in order to reach a reliable fitness.

Both the random and Swiss tournament types will be held while training against a random moving player (*subsection 2.3.1*) as well as a deterministic player (*subsection 2.3.2*) created during the experiments in ([Jac05]). The next chapter describes the experiments performed using tournaments.

Chapter 6

Random Pairing Experiments

6.1 Experiment setup

In order to test the capabilities of learning by using tournaments several different experiments need to be performed. For these experiments two different opponents were chosen to measure performance of competitive learning: a random moving player (*chapter 2.3.1*) and a deterministic player (*chapter 2.3.2*). These two opponents are the same as were used in *chapter 4.5* and chapter *4.7* and so a clear comparison can be made between cooperative learning as was done in those two chapters, and the competitive learning as is done for this chapter.

In [Jac05] several experiments using TD(λ), the deterministic player used for these experiments, have been performed. One big problem with the results in that research is that the random player that was used, contained a bug. This bug resulted in random selection from all possible moves minus 1. So if in a state there were 5 possible moves, it picked a move from the first 4 possible moves. This resulted in a random player that was not capable of using all possible moves, and thus covered less possible game states and in the end this player was easier to defeat. Therefore new experiments against both random and deterministic players were performed in this research using correct working randomness.

All experiments done use Enforced SubPopulation as neuroevolution technique as described in *chapter 3.3.2*. For this research ESP is used rather than NEAT. In tests done in *chapter 4.6.2* NEAT loses complexity when training against a random moving opponent, reducing its number of hidden neurons to as little as two. This may be a problem when learning in a tournament against oneself as the few skills required to defeat a new ESP player are low. This will most likely cause NEAT to evolve its networks into small topologies as were present in *chapter 4.6.2*. Secondly ESP was a lot easier to implement and monitor. When making changes to ESP or NEAT to make them work with tournaments rather than the standard cooperative learning method, a large change is needed in the software.

All experiments done use tournament play as a way of obtaining fitness for neurons. After each tournament all neurons should have a fitness assigned, and the neuroevolution is allowed to learn from the last tournament, calculating the next generation. Once the next generation is calculated a set amount of Othello players is generated from the population of neurons and another tournament is played. By default 200 tournaments are played, and thus 200 generations are calculated. After each tournament two measuring matches are played to monitor progress during the evolution process. For *chapter 6.2* this match is

done versus a random moving opponent, and for *chapter 6.3* done versus the deterministic player created during the research done in [Jac05]. In both matches a percentage of wins is calculated over a total of 1000 games played, 500 playing white and 500 playing black. These percentages give a good progress indication and can also be used as comparison with other researches into learning methods for playing Othello.
Tournaments in this research are either random pairing tournament or Swiss tournament types. A random pairing tournament is used in this chapter for experiments. A random pairing tournament is chosen over a round-robin tournament as a full round-robin would require too many evaluations to be performed to finish a tournament. An evaluation is in this case playing a game of Othello. This means two players can get an evaluation at the cost of only one game played as they are paired against each other.
A tournament lasts for a set number of rounds in which all paired players play 2 games against each other, 1 as white, 1 as black.
In each round a player can earn points. When a player loses the match, he receives 0 points, a draw yields 0.5 points and a win 1.0 points. *Figure 6.1* shows the pseudocode for the score. These points carry over to the next round. At the end of the tournament all players will have been assigned a fitness equal to the score they obtained playing matches during the rounds.

```
fitness(x) ← 0
for i ← 1 to r:
 if win(x):
  fitness(x) ← fitness(x) + 1
 if draw(x):
  fitness(x) ← fitness(x) + 0.5
```

Figure 6.1: Game scoring Pseudo-Code for a match of two games

Where:
x : A player in the tournament
r : Number of rounds of the tournament

Due to the restriction of having 2.000.000 evaluations per experiment it was necessary to find a proper balance between the number of players in a tournament, the number of rounds played and the number of generations calculated. More players in a tournament means having more evaluations required per generation. All these factors can be balanced using the formula in *figure 6.2*

$$\boxed{2.000.000 \leftarrow t * r * p/2 * g}$$

Figure 6.2: Total number of evaluations performed

Where:

t	: Number of tournaments per experiment: Default 200
r	: Number of rounds of a tournament: Default 50
p	: Number of players in a tournament: Default 200
g	: Number of games played during a match: Default 2

These parameters approach the parameters used in *chapter 4.5.2*, the parameters not listed here remain unchanged.

Since each neuron in the population needs to have a reliable fitness assigned, it is required that all neurons are evaluated at least 5 times during each tournament. Every generation every ESP player will receive one fitness assignment, the score they received from the tournament they participated in. Meaning for every ESP player, one neuron per subpopulation will receive one fitness assignment. If there are 10 ESP players in the tournament, and we have subpopulation sizes of 10, every neuron will receive on average 1 evaluation. In essence one will need at least five times as many ESP players than subpopulation size. Figure *6.3* shows the equation.

$$\boxed{e \leftarrow p/s}$$

Figure 6.3: Evaluations performed per neuron

Where:

e	: Number of evaluations per neuron
p	: Number of players in the tournament from this ESP
s	: Size of the subpopulations

6.2 Random opponents

6.2.1 Tuning for Random Moving Opponent

During the random pairing experiments a large number of different parameters were tested in order to get a better set of parameters for benchmarking. As an initial setup the most optimal parameters found during research done in *chapter 4.5.2* are used, being number of hidden neurons, subpopulation size and mutation rate. These parameters were complemented with parameters for the number of tournaments, number of rounds for each tournament, number of players for each tournament and type of tournament.

Number of hidden neurons	40
Subpopulation size	100
Mutation rate	40%
Number of ESP instances	1
Tournament type	Random pairing
Percentage random moving players	0
Percentage deterministic players	0

Experiments done with these default parameters are shown in *figure 6.4*. Along the y-axis the percentage of games won versus the random player is shown, and along the x-axis the number of evaluations performed is shown. The number of evaluations is chosen rather than the number of generations because using tournaments, different settings mean performing a different amount of evaluations per generation. For instance having more players in a tournament or playing more rounds per tournament influences the number of evaluations performed each generation.
Previous experiments done in *chapter 4.6* show ESP winning up to 87%, while these default parameters converge at 82%. This 5% performance loss can be explained due to the use of a different learning method for which the parameters are not optimized. So will we get better performance using different parameters for our experiments? First parameters and settings are tested for feasibility. Parameters to be tested are mutationrate, subpopulationsize, number of ESP instances and the percentage of random moving players in the tournament. These last two are not default parameters used by ESP. Multiple ESP instances might very well yield better results than the single instance supplying all the players. This is expected as in the standard tournament format for these experiments only ESP players are added, meaning the players have to evolve against themselves. When using a single ESP instance it might very well converge to a single strategy and fail to get out of local optima. Multiple instances of ESP are likely to converge to different optima and force other instances to leave their reached optima in order to defeat the others.
Adding random moving players to the tournament is a way of allowing ESP players to play against the same opponent that is used for the actual performance measurement every generation. This is another way of allowing a single ESP instance to leave its local optimum.

Subpopulation sizes: 5, 20 and 100
The experiments done with a subpopulation size of 5 show a clear decrease in performance,

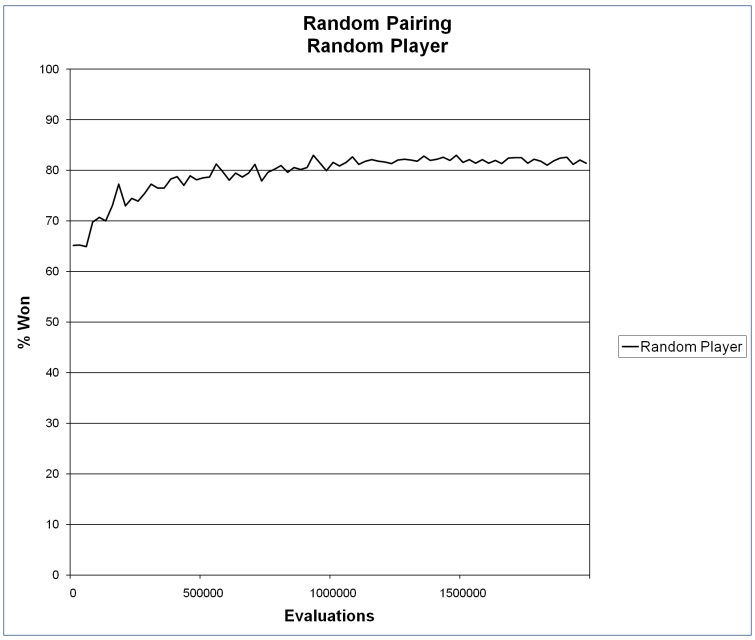

Figure 6.4: ESP, competitive learning using Random pairing and learning vs random moving opponent

scoring up to 73% wins against a random moving opponent. Experiments done with subpopulations 20 and 100 score 80% and 82% respectively The lower score obtained when using a subpopulation of 5 can be attributed to the fact there are less neurons available and thus less diversity in the population. This in turn leads to earlier convergence. The experiments done with a subpopulationsize of 20 show equally good results as were obtained with the experiments done with a subpopulationsize of 100. Although the results show the same performance in the end, the experiments done with a subpopulation of 20 show a faster convergence. But as convergence with a subpopulationsize of 100 happens fast enough and shows more learning potential a subpopulationsize of 100 is preferred.

Mutation rates: 1%, 10%, 40% and 50%
The second parameter being tested is the mutation rate of the neurons. Four different values were tested, 1%, 10%, 40% and 50%. All tested values give roughly the same result of 79% wins. A mutationrate of 40% seems to be slightly better scoring 80% and as this was also the optimal value in *chapter 4*, this will be used.

ESP instances: 1 and 5
All experiments done so far in this chapter result in suboptimal win percentages of 82%. As the goal of tournament play is to learn while playing oneself learning is more difficult. This gets increasingly more difficult as the diversity in the population is reduced due to conversion. When more diversity is maintained throughout the learning process, different strategies may be evolved. To this end experiments were done using multiple instances of ESP, each having its own population, and each providing players to the same tournament.

Each instance of ESP can learn its own strategy and each instance can develop a new strategy to defeat the dominant one. Experiments are done using one instance of ESP, and experiments are done using five instances of ESP.
While learning against a random moving player the single instance of ESP scored 81% while the experiments with five ESP instances scored 79%. It is likely not much skill is required to defeat a random moving player and as such not a great many strategies need to be explored. No faster or better convergence is present when using five instances of ESP.

Random players: 0%, 50% and 80%
Another cause of learning sub optimal is the fact tournament play thus far meant playing only against other learning opponents. These opponents lack any decent strategy at the start, and although learning like this may lead to successful new and previously unknown strategies, this has not happened. In fact only suboptimal results have been witnessed. In order to reach better results random moving players are added to the tournament. This should give the learning ESP players a target to learn toward and to exploit its weaknesses. When playing a random moving opponent more states are visited which allow the learning to be more focused on defeating the random moving player which is used as benchmark. Experiments are done using a tournament with 0% random moving players, 50% random moving players and experiments were done using 80% random moving players.
Adding 50% random moving players resulted in a performance of 80%. For the experiment with 80% random moving players the same results of 80% wins were reached. Having random moving playing in the tournaments clearly does not add needed skill and diversity resulting in better learning when using random pairing and a random moving opponent as benchmark.

6.2.2 Final Experiments versus Random Moving Opponent

After tuning experiments have finished, a final batch of experiments is done using the best performing parameter values obtained. These final tests will be used to compare random pairing tournament play using a random moving opponent with the ESP experiments performed in *chapter 4.5*. These comparisons include; score obtained, speed of convergence and whether the best players perform well against other opponents rather than only the ones trained against.

For these experiments the following parameter values are used:

Number of hidden neurons	40
Subpopulation size	100
Mutation rate	40%
Number of ESP instances	1
Tournament type	Random pairing
Percentage random moving players	0
Percentage deterministic players	0

The experiments with these parameters are performed ten times to get more reliable results. During the tuning experiments a benchmark was done using 1000 games against a random moving opponent, these experiments also include a benchmark of 1000 games against a

deterministic opponent. This way it can be determined if the best performing ESP player performs well against only the random moving player it is learning against, or if it performs good against other players as well. *Figure 6.5* shows the results of these experiments.
Using random pairing and a random moving opponent allows ESP to reach a score of 82% after 1.000.000 evaluations. When placing the same ESP player against a deterministic opponent it manages to reach a score of 39% after 2.000.000 evaluations. The improvements have not yet converged onto an optimal solution and better results might be obtained when allowing more evaluations to be performed. These results can be compared with experiments done with cooperative ESP as done in *chapter 4.5.2*. *Figure 6.6* shows the same results as shown in *figure 4.6*, but has an extra graph plotted. This extra graph is the performance against a deterministic opponent (TD Greedy from *chapter 2.3.2*) while learning versus a random moving opponent. While evolving using cooperative learning, the playing strength versus the random moving opponent increases rapidly, with a score of 87% obtained after 1.000.000 evaluations. The playing strength versus the deterministic opponent starts at 10% wins, and climbs to a maximum of 24% after 1.500.000 evaluations.
It is clear the cooperative learning method manages to learn a better strategy versus the random moving opponent, also achieving this in a smaller amount of evaluations. The players evolved using cooperative learning are only specialized against the random moving player, exploiting the weaknesses present in this player. The overall playing strength in the cooperative learning fails to reach the level obtained when using competitive learning.

6.3 Deterministic opponents

As a second type of experiment the same setup and experiments done in *chapter 6.2* were done while using a benchmark against a deterministic opponent. This deterministic opponent (*chapter 2.3.2*) was trained during research done in [Jac05] and has proven to be a much stronger opponent than the random moving opponent. The advantage of the deterministic opponent is the less amount of states needed to be learned by ESP. In *chapter 4.7* the experiments were shown with NEAT and ESP learning against a deterministic opponent.
As all tuning experiments were done simultaneously, the same initial values are used as default while tuning.
For these experiments the following variables were used as initial values:

Number of hidden neurons	40
Subpopulation size	20
Mutation rate	40%
Number of ESP instances	1
Tournament type	Random pairing
Percentage random moving players	0
Percentage deterministic players	0

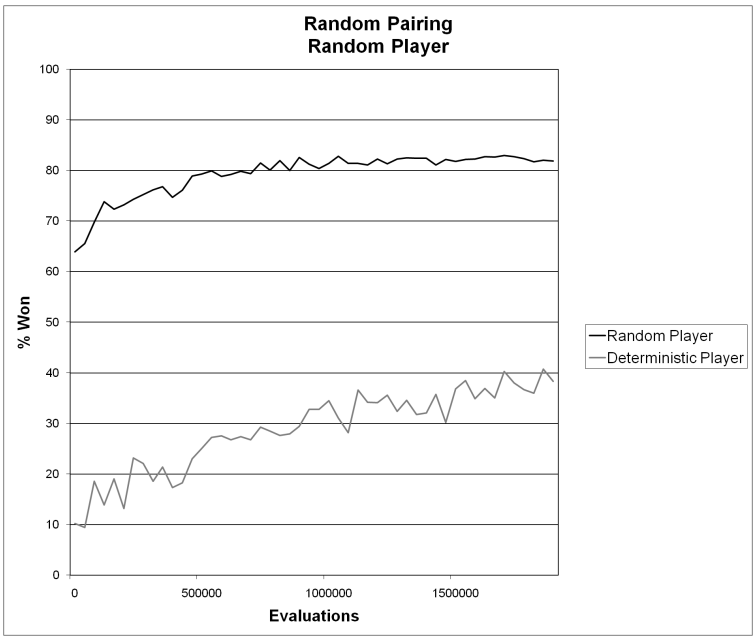

Figure 6.5: ESP, competitive learning using random pairing and learning vs random moving opponent

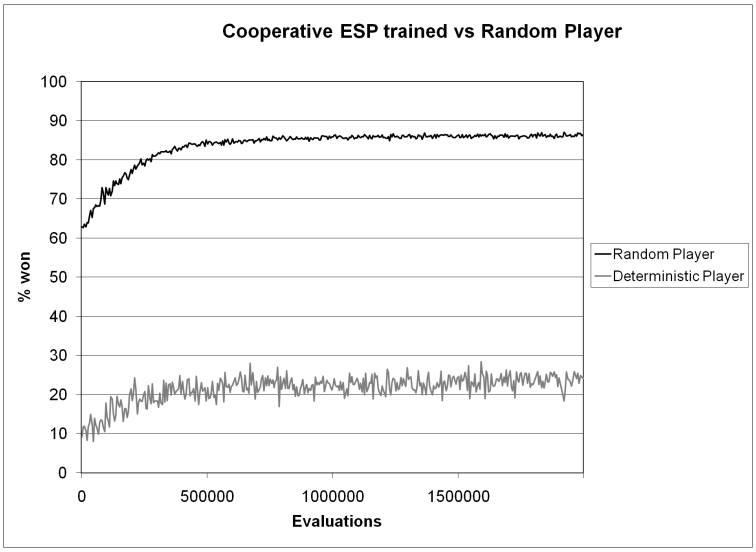

Figure 6.6: ESP, cooperative learning vs random opponent

6.3.1 Tuning for Deterministic Opponents

For learning against a deterministic opponent the same variables were tested; mutation rate, subpopulation size, ESP instances and adding the same deterministic player used as benchmark to the tournament in different proportions. The same experiments were done for the Swiss tournament as were done for the random pairing experiments.

Mutation rates: 1%, 10%, 40% and 50%
First experiments were done using different mutation rates; 1%, 10%, 40% and 50% were performed. A clear decrease in learning capability is seen when using the competitive learning method instead of the original cooperative learning method. As the original cooperative learning method scored 87%, the best performing setting for the competitive learning method was only 25% using a mutationrate of 10%. When using mutationrates of 1%, 40% or 50% the population converges faster to a maximum but resulted in a less winning percentage than was the case when using a 10% mutationrate. For a mutationrate of 1% a score of 17% was reached. A mutationrate of 40% gives a score of 16% and using a mutationrate of 50% ESP scores 20% wins. In contrary to experiments done on mutationrates in *chapter 6.2.1* mutationrates do influence the learning capabilities. When learning versus a random moving opponent, any strategy or tactic learned gives better scores. When learning against a deterministic opponent, only a select few strategies are succesful and as such these need to be learned. When seeding an initial population the information for this strategy may not be available and mutation is needed to explore. A too low mutationrate will result in not enough explorations while a high mutationrate will hinder learning due to its destructive nature.

Subpopulation sizes: 5, 20 and 100
The different subpopulation sizes were tested next. As was the case in *chapter 6.2*, subpopulation sizes of 5, 20 and 100 were used to determine the best performance. Using the different subpopulation sizes gave promising results. The smaller subpopulations of 5 and 20 resulted barely any learning at all and managed to get a score of 15% wins against the deterministic player. The larger subpopulation of 100 neurons resulted in significantly better learning and reached a score of 33%. As was the case when testing mutationrates, a higher mutationrate gave more diversity to the population which resulted in better learning. This is also true with subpopulationsizes; a low size means little diversity where a larger subpopulations means more diversity.

ESP instances: 1 and 5
With the goal of testing multiple instances of ESP and as such maintaining more diversity and multiple hill climbing effects, experiments are done using one and five ESP instances. As was expected an increase could be seen when using five ESP instances rather than one. Five ESP instances reach a score of 34% whereas one ESP instance scores 16% wins against the deterministic opponent. Using five instances results in much faster learning than using a single instance of ESP. No genetic information is passed on between the different ESP

instances.
As total running time is important, and only evaluations are used as measure of time, the use of five ESP instances increased experiment times by 20-30% even though the same total number of 2.000.000 evaluations was allowed. Multiple instances means more memory usage and computing five new generations everytime instead of the normal one.

Deterministic players: 0%, 50% and 80%
The final setting to be tested is adding multiple deterministic players to the tournament, where these deterministic opponents are the same as being used as benchmark. Adding these players to the tournament will give the learning ESP players weaknesses to exploit and a stronger player to learn against. This in turn should result in better learning. By default the tournament had 0% deterministic players, but for these experiments this was increased to 50% and 80% and using 50% and 20% ESP players respectively.
As expected, adding the same deterministic player to the tournament as is used for benchmarking results in better learning and a significantly higher score. When using 50% ESP players and 50% deterministic players in the tournaments, a result of 34% wins was reached in contradiction to the 16% obtained when using 100% ESP players. Using 80% deterministic players resulted in a lower score reached than when using 50% deterministic players. This is unexpected as more deterministic players in a tournament would mean more matches would be played between ESP players and deterministic players. The decrease can be attributed to the fact less evaluations are done for the neurons in ESP as less ESP players are used in the tournaments.

6.3.2 Final Experiments versus Deterministic Opponent

After the tuning experiments in *chapter 6.3.1* have been completed, a final set of experiments has been performed in order to test the techniques using their optimal parameter values. Parameter values have been tested using default values and experimenting with one of the parameters. The best performing parameter values are used in these final experiments. These experiments use the following parameter values:

Number of hidden neurons	40
Subpopulation size	100
Mutation rate	10%
Number of ESP instances	5
Tournament type	Random pairing
Percentage random moving players	0
Percentage deterministic players	50

Ten experiments have been performed using these parameter values. Instead of the usual benchmarks of 1000 games against a deterministic opponent between the generations, two different benchmarks have been performed; 1000 games against the deterministic opponents, and 1000 games against a random moving opponent. These two benchmarks will show, in comparison with the same benchmarks performed upon the cooperative learning ESP, whether competitive learning learns equally good as the cooperative technique, and whether competitive learning creates ESP players who play well versus a larger diversity of

opponents. *Figure 6.7* shows the results of the experiments using these optimal parameter values. Here we can see two graphs: one for the benchmark versus the deterministic opponent, and one versus the random moving opponent. The tests versus the deterministic opponent show learning, although at a limited scale. The maximum score reached is 32% after 800.000 evaluations. When tested the same player against a random moving player a score of 71% is reached after 750.000 evaluations. These scores can be compared to the graphs plotted in *figure 6.8*. *Figure 6.8* shows the cooperative learning done by ESP against a deterministic opponent. These are the same opponent and setting as done for the experiments in *figure 4.14*. When comparing the performance versus the deterministic opponent, it is clear cooperative learning is superior to the tournament learning done using random pairing. Cooperative learning reaches a maximum score of 70% versus the deterministic opponent, while tournament play using random pairing reaches 32%. The speed of learning is faster using the traditional cooperative learning. Although random pairing tournament play reaches its maximum at 800.000 evaluations, this maximum is significantly lower than when using cooperative learning. While these experiments have used a deterministic opponent to learn, it is interesting to see how this evolved ESP population performs against a different opponent, a random moving opponent. This is tested to determine how specialised the players are. The best performing player in *figure 6.8* fails to increase performance against a random moving opponent. A score of 64% is maintained throughout the experiments. This indicates cooperative learning manages to learn against the player being trained against, but fails to increase overall performance. When using random pairing tournament play an increase of performance is seen reaching a score of 72% against a random moving opponent. This increase is performance can be attributed to the pairing which takes place during the tournaments. When pairing against another learning player means visiting more states resulting in better overall play, which is tested by the random moving opponent.

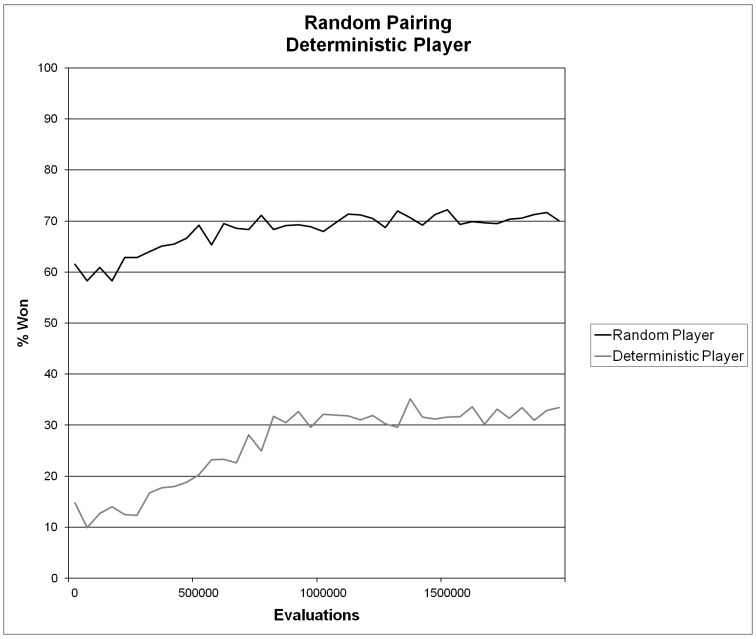

Figure 6.7: ESP, competitive learning using random pairing and learning vs deterministic opponent

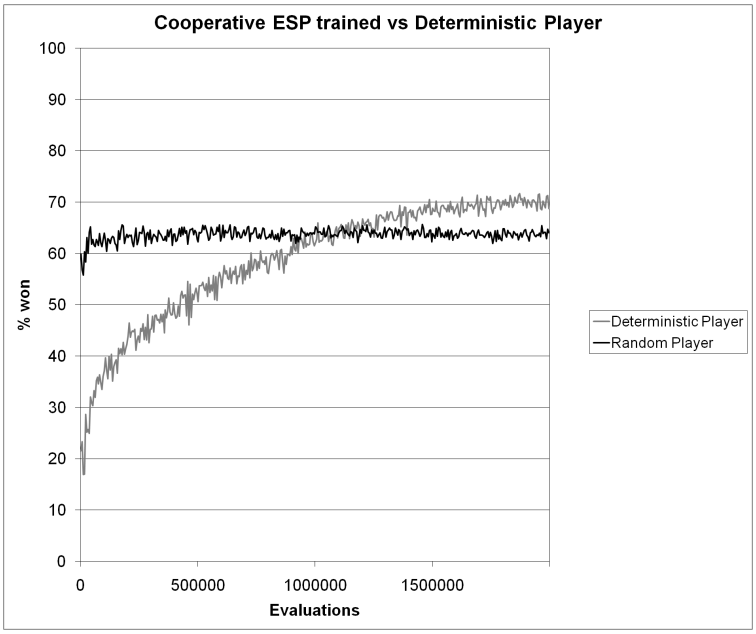

Figure 6.8: ESP, cooperative learning vs deterministic opponent

Chapter 7

Swiss Pairing Experiments

This chapter shows the results obtained while using a Swiss tournament instead of the random pairing experiments done in *chapter 6*. Although the experiments in the previous chapter were successful while training against a random moving opponent, ESP failed to learn a good strategy against the deterministic opponent while using random pairing. Swiss tournament is a more advanced tournament type where players only play opponents of roughly the same strength. Swiss tournaments are described in *subsection 5.2.2*. Using Swiss tournaments, a better learning curve is expected than was the case in *chapter 6*.

7.1 Experiment setup

As was the case in chapter *6*, all experiments here use tournaments in order to receive their fitness measurements. The Swiss pairing algorithm is used here to provide a better and more reliable way of measuring these fitnesses. The Swiss pairing tournaments will take slightly more time to complete as the algorithm itself is more complex than the random pairing used previously. This extra time, however, is negligible in contrast to the time required for evaluating the players. Every experiment is limited to doing 2.000.000 evaluations, and is repeated five times each. For the experiments the same default parameters are used as in *chapter 6* due to simultaneous execution.

7.2 Random opponents

7.2.1 Tuning for Random Opponents

The first type of experiments done are learning versus a random moving opponent. Once again a large amount of experiments were done to determine the best possible parameter values for learning. A different pairing technique may very well require different values for its parameters than was the case with random pairing. As a starting point the same parameters were used as was done in *section 6.2*.

Number of hidden neurons	40
Subpopulation size	20
Mutation rate	40%
Number of ESP instances	1
Tournament type	Swiss
Percentage random moving players	0
Percentage deterministic players	0

Tuning of the parameters was done exactly the same as it was done in *subsection 6.2.1.*

Mutation rates: 1%, 10%, 40% and 50%
As a mutation rate the values 1%, 10%, 40% and 50% were tested. The experiments showed no difference when using the four different mutationrates. All variables resulted in a winning percentage of 79% When comparing the results to the results obtained at *subsection 6.2.1* equal results are obtained.

Subpopulation sizes: 5, 20 and 100
Subpopulation sizes have once again been tested at values of 5, 20 and 100. Using a subpopulation of 5 shows little to no learning. A very short learning period results in a winning percentage of only 70%. Using a subpopulation of 20 resulted in less fast learing but a better end result of 79%. A subpopulation of 100 meant having more gradual learning due to having to replace more bad performing neurons in order to have a larger amount of good performing networks. A subpopulation of 100 resulted in a winning percentage of 83%. The results from using a Swiss tournament rather than a random pairing tournament do not differ significantly.

ESP instances: 1 and 5
When comparing different numbers of ESP instances, one and five have been tested to maintain more diversity and keep different populations trying to outperform each other. The results of testing one and five instances of ESP training against a random moving opponent using Swiss pairing are equal. Both the single instance and the five instances of ESP reached a winning percentage of 79%.

Random players: 0%, 50% and 80%
When using Swiss pairing the placing of random moving players to the tournament might have a positive effect on learning. Once again experiments were done using 0%, 50% and 80% random moving players in the tournaments. Using different percentages of random moving players in the tournaments resulted in different learning behaviour. On contrary to what was expected using 0% random moving players in the tournaments resulted in the best results of 79% wins. Using 50% random moving players resulted in a winning percentage of 78% and using 80% random moving players resulted in 75% wins. Adding random moving playes to the tournaments does not add the required information to the tournaments which the ESP players can exploit. Adding a higher percentage of random moving players actually harms the learning process as the ESP players receive less evaluations and thus a less reliable fitness.

7.2.2 Final Experiments versus Random Moving Opponent

Final experiments were performed using the best parameter values obtained during the tuning performed in *subsection 7.2.1.* These parameters are tuned changing one parameter and keeping all others the same. This shows what parameters are important and what values work better than others.

During the tuning process in *subsection 7.2.1* these parameter values are found best performing:

Number of hidden neurons	40
Subpopulation size	100
Mutation rate	40%
Number of ESP instances	1
Tournament type	Swiss
Percentage random moving players	0
Percentage deterministic players	0

Using these parameter values, experiments have been performed in order to determine learning capabilities. Instead of the usual benchmark of 1000 games against a random moving opponent, again two benchmarks are performed during the experiments: one versus the random moving opponent and one versus the deterministic opponent. This experiment will show learning capabilities using Swiss tournaments and a random moving player as benchmark. The experiment has been repeated ten times and results from these experiments can be seen in *figure 7.1*.
These experiments will be compared to the experiments done using random pairing. The random pairing experiments are plotted in *figure 7.2* and are the same graphs as *figure 6.5*. Using Swiss pairing, a score of 84% is achieved after doing 1.200.000 evaluations, where random pairing reached a score of 82% after 1.000.000 evaluations. Random pairing has a stronger convergence although fails to reach the score of 84% as was reached by the Swiss pairing experiments.
In both the random pairing experiments and the Swiss pairing experiments done versus a random moving opponent, the global skill of the evolved ESP population increases steadily. Using random pairing a score against the deterministic opponent of 39% was reached, where the Swiss pairing experiments reached 35%. Both experiments show the same learning rates over the same courses of time.

7.3 Deterministic opponents

7.3.1 Tuning for Deterministic Opponents

During the tuning process of the parameters for Swiss tournaments and deterministic opponents, the same experiments are performed as done in the other three tuning tests.

Number of hidden neurons	40
Subpopulation size	20
Mutation rate	40%
Number of ESP instances	1
Tournament type	Swiss
Percentage random moving players	0
Percentage deterministic players	0

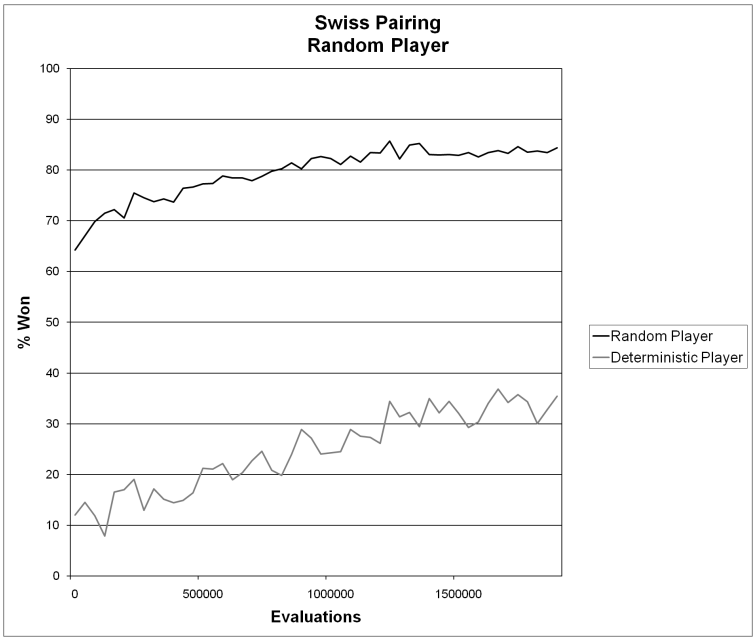

Figure 7.1: ESP, competitive learning using Swiss pairing and learning vs random moving opponent

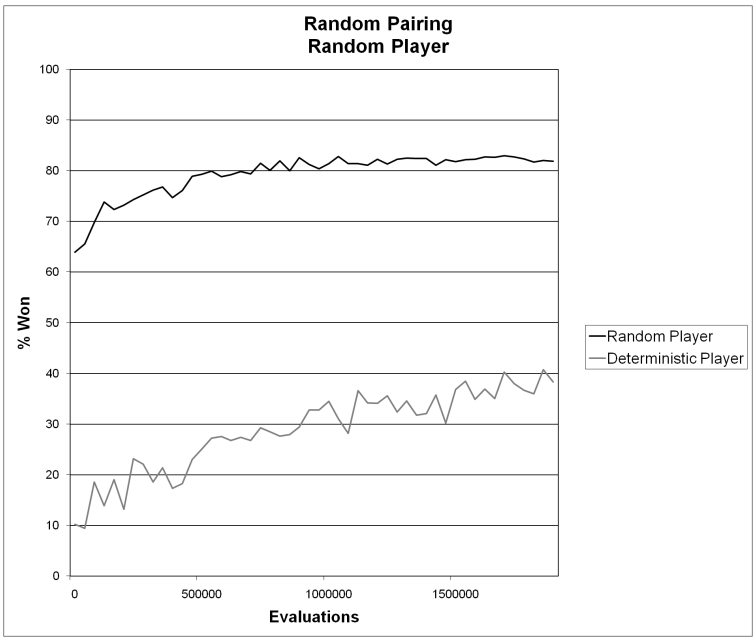

Figure 7.2: ESP, competitive learning using random pairing and learning vs random moving opponent

Mutation rates: 1%, 10%, 40% and 50%
Different mutation rates have been tested in order to determine the most promising mutation rate. Once again mutation rates of 1%, 10%, 40% and 50% have been tested. Using different mutationrates when using Swiss tournaments and a deterministic opponent results in equally fast learing. Mutationrates of 1% and 50% both scored 22% wins versus the deterministic opponent. When using a mutationrate of 10% a score was obtained of 25%, a mutationrate of 40% resulted in a winning percentage of 23%. Using a mutationrate of 10% returns the best results for this setup.

Subpopulation sizes: 5, 20 and 100
Subpopulation sizes 5, 20 and 100 have once again been tested. It is expected to see a subpopulation of 100 to perform better than subpopulations of 5 and 20.
When using a subpopulation size of 5 hardly any learning takes place, and ESP manages to reach a score of 13%. This was expected as tuning experiments done in *subsections 6.2.1*, *7.2.1* and *6.3.1* also showed little to no learning when using a subpopulationsize of 5. Obviously a small subpopulationsize means too little diversity for learning although each neuron gets a better and more reliable fitness assigned as every neuron has a larger chance of being selected for use in a network.
A subpopulation size of 20 resulted in a score of 25%. Again a fast convergence is the result although suboptimal. The experiments done with a subpopulation of 100 showed a more gradual learning curve and resulted in a winning percentage of 33%. using a subpopulationsize of 100 has the best results when using Swiss tournaments and a deterministic opponent.

ESP instances: 1 and 5
Again two different setups are tested concerning the number of ESP instances. More instances should result in more diversity maintained as well as different hill climbs being performed at the same time. Higher scores are expected when using multiple ESP instances rather than a single one. For these experiments two setups are tested: a single ESP instance, and five ESP instances. When using a single ESP instance a result is obtained of 25% wins versus the deterministic opponent. Five ESP instances result in 29% wins against the deterministic opponent.

Deterministic players: 0%, 50% and 80%
Three different proportions of deterministic players have been added to the tournaments to add information for the ESP players to use and exploit. When using 0% deterministic players in the Swiss tournaments, a winning percentage of 25% was reached. Adding deterministic players will add required information to the tournaments, but at the cost of evaluations and so less evaluations can be performed on ESP players. When using 50% deterministic players in the tournament a result of 24% wins was reached. Using 80% deterministic players in the tournament did mean an improvement. 27% Wins against the deterministic opponent are reached. A reason for the decrease in performance when using 50% deterministic players in the tournaments might be the fact the added information does not outweigh the fewer evaluations performed on the ESP players. One would expect this to be even more true when using 80% deterministic opponents, although the higher percentage means a much larger chance to pair up with a deterministic player. Apparantly this happens often enough

to outweigh the fewer evaluations performed for ESP players.

7.3.2 Final Experiments versus Deterministic Opponent

Experiments have been performed using the tuned parameters for comparison to the experiments performed in *section 4.7* and *subsection 6.3.2*. Expectations are to see a better performing competitive learning ESP population than was the case in *subsection 6.3.2*. The best performing parameter values at *subsection 7.3.1* are used here, and are listed.

Number of hidden neurons	40
Subpopulation size	100
Mutation rate	10%
Number of ESP instances	5
Tournament type	Swiss
Percentage random moving players	0
Percentage deterministic players	0

These parameter values are equal to the values used for the experiments using random pairing, except for the percentage of deterministic opponents entered into the tournaments. Although using 80% deterministic opponents in the parameter tuning proved to be slightly more succesful in learning, it was chosen not to use this value for the final experiments. As can be seen in *figure 6.2* there is a trade-off between using more players and the number of rounds and tournaments that can be played. Both using five instances of ESP and using 80% deterministic opponents in the tournament resulted in improving results. Using 80% deterministic opponents in the tournament means having a tournament size five times as large as using 0% deterministic opponents. Because every individual in the population of ESP needs to play sufficient games in order to receive a reliable fitness for each neuron in the population, reducing the number of ESP players is not an option. Using five ESP instances also require each instance to add sufficient players to the tournament, which means each instance needs to add 100 players to the tournament. When using a total of 500 ESP players, and adding 80% deterministic opponents, doing tournaments and allowing only 2.000.000 evaluations means calculating very few generations for the ESP instances, and so allowing very little evolution to take place. Because of this the best performing option was chosen to use as final experiment, this being the use of five ESP instances and 0% deterministic opponents.

The experiments using these parameters have been repeated ten times and use, like the other tuned experiments in *subsections 6.2.2*, *6.3.2*, and *7.2.2*, two different benchmarks to measure performance during these experiments. After each generation the winner of the tournament plays 1000 games against a deterministic opponent, followed by 1000 games against a random moving opponent.
Comparison is done between the results obtained using random pairing while learning using a deterministic opponent, and Swiss pairing using the same deterministic opponent for learning. As done with the other three final experiments, this comparison is done with learning speed, maximum scores reached and specialized learning. *Figure 7.4* shows the same results which were shown in *figure 6.7* while *figure 7.3* shows the results from the

Swiss pairing experiment.

Using Swiss pairing as a tournament technique, successful learning was achieved. When comparing the scores reached in the benchmarks against the deterministic opponent Swiss pairing reached a top of 38% after 2.000.000 evaluations. At this point convergence has not yet completed and it is expected to perform better after having reached more than 2.000.000 evaluations. Comparing the results to the results from using random pairing tournaments, random pairing reached a maximum of 32% wins, while Swiss pairing reached 38%. Learning speed is equally fast although using Swiss pairing allows learning past the 32% wins. Performance against the random moving opponent has also increased when using Swiss pairing. In *figure 7.4* a maximum score of 71% was achieved against the random moving opponent. Using Swiss pairing tournaments allows ESP to reach a score of 83%. The performance increase versus the random moving opponent is significantly faster than is the case when using random pairing tournaments. This shows Swiss tournaments are capable of learning better and faster and also create players which are better suited playing different opponents.

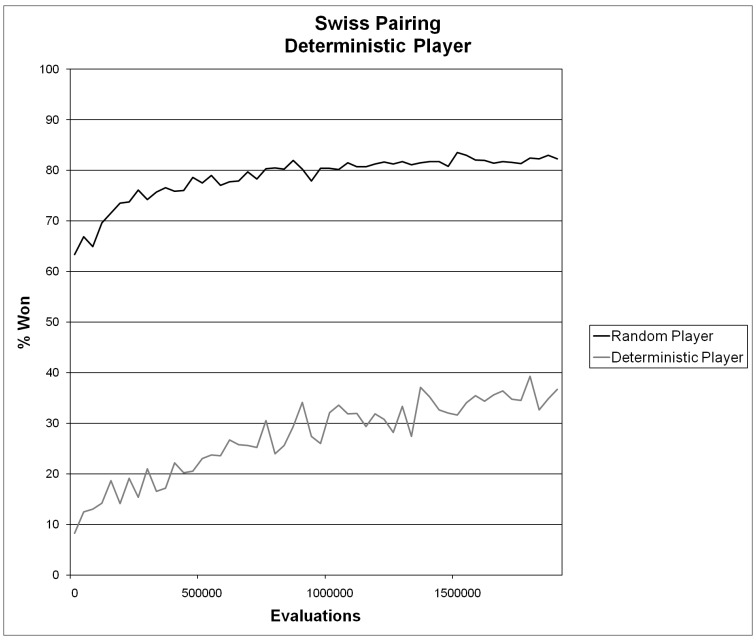

Figure 7.3: ESP, competitive learning using Swiss pairing and learning vs deterministic opponent

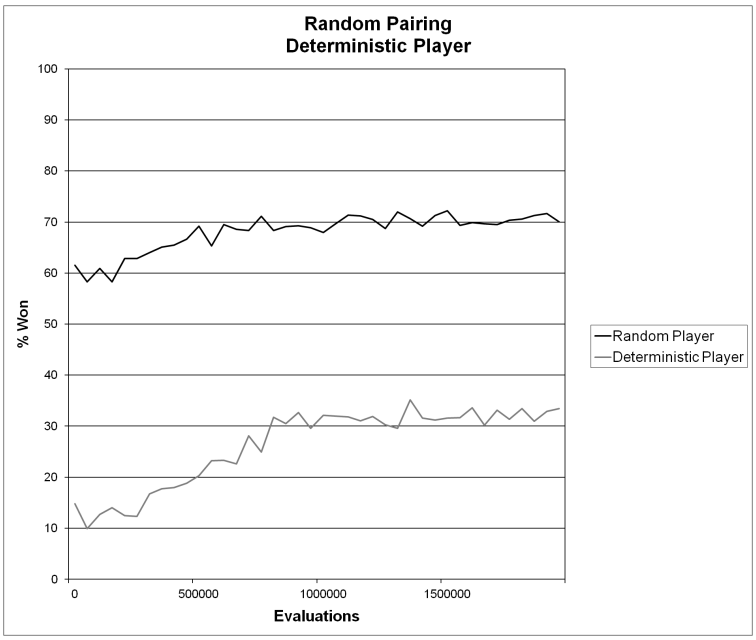

Figure 7.4: ESP, competitive learning using random pairing and learning vs deterministic opponent

Chapter 8

Conclusion

8.1 Part I - Neuroevolution and Othello

Find the neuroevolution technique that is best at learning to play Othello.

Three neuroevolution techniques, SANE, ESP and NEAT, have been tested and compared in terms of learning potential and speed.

Looking at the results in *chapter 4* the first conclusion is that both ESP and NEAT perform much better than SANE. SANE ended with a 83% win percentage against the random opponent. That is not a bad score and several other researches with reinforcement learning and Othello show percentages like that. Probably with some optimizations SANE can perform even better. But we expect that in the long run it will never be as good as ESP and NEAT.

ESP is a technique partly based on SANE, but more sophisticated in terms of specialization. ESP uses subpopulations to choose its neurons from which leads to faster specialization than SANE.

The results of ESP show a much better performance than SANE. There is a difference between ESP with and without delta coding where the experiments without delta coding show better results. Maybe delta coding can be used to increase the end result, but further experiments are needed to find out. Without delta coding, ESP reaches a 88% win percentage against a random opponent which is significantly higher than SANE.

ESP shows good gaming potential. It has smooth learning curves and shows good progression over the generations. And because of its 40 hidden neurons it probably has good potential against different opponents because the last technique, NEAT, only uses a few hidden neurons to perform even better than ESP.

NEAT differs from SANE and ESP, because it evolves both weights and topology of the neural network. The results of NEAT are very promising. A steep learning curve in the beginning and a high win percentage at the end of the experiment. After 400 generations it had reached a win percentage of 92% against the random opponent and is still (slowly) learning. This is a better result than ESP and SANE, which gives NEAT good potential.

One remarkable observation was that the best networks only use 2 hidden neurons. The strategy includes a focus on only 2 corners of the board instead of 4. So apparently very few knowledge and a simple strategy is all that it takes to beat a random opponent.

To find out whether NEAT can perform against other opponents NEAT was tested against the same deterministic opponent as ESP played against. The results of this experiment also shows slightly better results than ESP, ending with a 90% win percentage after 400 generations. The networks needed around 6 hidden neurons to beat this player, which is a lot less than the 40 that ESP uses.

It is clear that NEAT has the best results. This is probably due to the fact that NEAT evolves the topology instead of only the weights, resulting in much more different neural networks. This makes NEAT a very powerful method because it can minimize the number of neurons needed resulting in a faster learning process. Both SANE and ESP use a fixed topology that has to be designed before the experiments. This results in bigger neural networks and they take much longer to learn.
Both ESP and SANE use a fixed topology and ESP is based on SANE, but is more sophisticated. So it is not a big surprise that it performs better.
NEAT and ESP are completely different, but both show smooth learning curves and good results. So which one has the best potential to learn an overall good strategy for Othello? Both have their strengths and their weaknesses.
ESP uses a fixed topology which needs a good decision of the researcher before starting the experiments, but with a large network it is able to learn a good strategy. NEAT shows that a very small network is capable of defeating a random opponent, so ESP can probably get better and faster results with a smaller network against a random opponent, but that does not make a good Othello player.
NEAT does not need a decision on its topology, but to be able to learn a good topology and strategy, it needs several different opponents.
Because of time constraints it was not possible to do more experiments to find out how ESP and NEAT perform when learning against different opponents at the same time. New experiments are needed for that.

8.2 Part II - Neuroevolution: Cooperative and Competitive

Can learning speed in neuroevolution be increased by using a different learning method while maintaining the same level of play?

When doing experiments for Part I, good results were achieved and SANE, ESP and NEAT learned at good speed. The player created performed very well against the type of player being trained against. Although NEAT performed best in Part I, ESP was used instead due to advantages and disadvantages discussed in *chapter 6*.

In Part II, fitnesses of neurons was determined through tournament play rather than pairing every ESP player with the type of player used as benchmark. The goal is to need less games being played to receive the same amount of fitness measurements and maintain the same level of play.
Four different types of experiments were performed using two different tournament types as well as two different types of opponents. For every type of experiment tuning of the parameter values was performed in order to achieve the best results for that setup.

The first experiment done was having a random pairing tournament learn to play against a random moving opponent. The same was done in Part I using ESP where a result of 87% was reached after 1.000.000 evaluations. This experiment showed random pairing to be slightly less succesful at learning to play against a random moving opponent, reaching a score of 82% after 1.000.000 evaluations. The learningspeed itself did not differ much

between the two types of experiments. Interesting to see was the performance measured versus the determinsitic opponent while learning against the random moving opponent. The cooperative ESP experiments reached a result of 24% where the competitive experiments using random pairing resulted in 39% wins.

The second type of competitive neuroevolution against the random moving opponent tested was Swiss pairing. Swiss pairing was expected to perform better than random pairing due to its more advanced pairing algorithm. When training against the random moving opponent, random pairing reached a win percentage of 82% where Swiss pairing reached 84%, this is according to expectations. While training versus the random moving opponent random pairing reached a score of 39% versus the deterministic opponent where Swiss pairing reached 35%. It is unclear why Swiss pairing fails to reach an equal or higher result than obtained using random pairing since parameter settings were the same.

The third type of experiment done was using neuroevolution with a random pairing tournament learn to play versus a deterministic opponent. In the cooperative experiments done in Part I, ESP reached a score of 70% after 2.000.000 evaluations. Using tournament play, it was expected to see less good results against the deterministic opponent. But as was the case using random pairing, a more allround strategy was expected to be learned. Using random pairing against a deterministic opponent a result of 32% was reached which is significantly lower than using the standard cooperative learning. Although it took standard cooperative ESP 2.000.000 evaluations to reach 70% and random pairing 800.000 evaluations to reach 32%.
This score of 32% might improve if more than 2.000.000 evaluations would be allowed as no full convergence has taken place. The score itself is likely the cause of not pairing the ESP players with the deterministic opponents all the time as is the case with the cooperative learning in *section 4.7*. The experiments using cooperative learning have the ESP players each play 50 matches against the deterministic opponent, resulting in playing with good skill against this deterministic player to receive a high fitness. In the experiments performed in *subsection 6.3.2* no deterministic opponents have been added to the tournament, having all ESP players play themselves rather than the determinsitic opponent. This in turn means the ESP players receive fitness for their capability in playing a player other than the deterministic opponent.
An advantage of having all ESP players play themselves is the fact they do not specialize against a single type of opponent, but rather evolve a more widely usable strategy. This strategy might not be as effective as players specialized in defeating the deterministic opponent but are an interesting area of research nonetheless.

The fourth and last type of experiment performed was using a Swiss pairing tournament in order to evolve the ESP population into playing the deterministic opponent. Using a Swiss pairing tournament should increase the learning capabilities versus the deterministic opponent slightly.
Using Swiss pairing, a score was obtained of 38% wins against the deterministic opponent, where the random pairing tournaments reached a score of 32%. This rise in score is likely due to the fact the Swiss pairing allows a better sorting of ESP players to take place, which

in turn allows faster and better learning. it is, however still much lower than the score of 70% obtained using standard cooperative learning done in *section 4.7*. Even though the ESP population is trained versus the deterministic opponent, the performance versus the random moving opponent also increases up to a win percentage of 83% where random pairing reached 71% and the standard cooperative learning reaches a score of 64%.

From these experiments it can be concluded competitive learning results in less specialization and allows players to be evolved who are capable of playing successfully against a large number of different types of opponents. The learning capabilities are less successful when a player needs to be trained versus a single type of opponent, but is more successful when an ESP population needs to be trained to play against multiple opponents.
Learning speeds do not differ much between the two different pairing techniques.
An interesting area of research is present in the training of players in evolving more general strategies in order to defeat different playing styles offered by different opponents. Neuroevolution allows fast learning, and combined with tournament play it allows for populations to evolve these more general strategies in order to create more interesting players capable of defeating more than one type of opponent.

Chapter 9

Recommendations

9.1 Part I - Neuroevolution and Othello

Several improvements on the experiments could lead to an even better comparison of the three techniques.

To measure the performance of a technique 50 games of Othello were played by each neural network in the population each generation. At the end of the generation, the best neural network was allowed to play 1000 games to measure the champion's fitness of that generation. The problem here is that 50 games might be too few. There can be a lot of inaccurate results resulting in unjust champions, especially against a random opponent. So maybe more games are needed, or a different way of measuring the performance.

More interesting future research is learning against several different opponents. This way the result should be an overall good Othello player. Therefore more sophisticated Othello players are needed. The big disadvantage of this is that good opponents are slow which will slow down experiments dramatically.

Another interesting possibility is learning against itself. Have one technique play tournaments where neural networks play against each other every generation, and evolve the good ones. The advantage is that it is much faster than learning against slow sophisticated opponents, but the disadvantage is that no result is guaranteed because of the lack of known good opponents.

ESP showed good potential and can be optimized by experimenting with different population and subpopulation sizes and different delta coding values.

Other interesting areas are combining a good neuroevolution technique with other reinforcement learning methods.

Also optimizing the knowledge representation is an interesting area of research. Maybe put some more focus on learning strategies and use a priori knowledge of good strategies to speed up the learning process.

In this research only three techniques have been compared, while other techniques like TEAM or CoSyNe might have good gaming potential. It can be interesting to test these techniques as well.

9.2 Part II - Neuroevolution: Cooperative and Competitive

As discussed in *section 9.1*, improvements to learning using neuroevolution could be made using tournament play. Although a number of different approaches have been explored into the use of tournament play combined with neuroevolution, there are areas of interest to be researched.

As became apparent during the experiments performed, the limit of 2.000.000 evaluations was in some cases insufficient as no convergence was reached during the experiment. Better results can be obtained when a longer period of learning is allowed.

Although the tuning of the parameters has been performed, this can be done in a more precise way. The different experiments used rather similar parameter values, but also a range of values remained untested during this research. Larger subpopulations, for example, might result in even better performance.

During this research only two different types of tournaments have been tested: random pairing tournaments and Swiss tournaments. Using these types of tournaments only shows a limited part of the available tournament types and different tournaments could very well result in better learning. Using promotion and relegation tournaments in combination with longer experiments could result in better learning. Using promotion and relegation tournaments could be particularly interesting when used in combination with NEAT as neuroevolution technique. Promotion and relegation tournaments require the same players/teams being in the tournament, and as NEAT uses complete networks as population this can be achieved.
NEAT could also be coupled with the already tested tournament types random pairing and Swiss, longer learning times are expected due to likeliness of initial simplification of the networks.

Another improvement could be using many different types of players in the tournaments. Only random moving opponents and the deterministic opponents have been added to the tournaments to improve performance. Using many different players will cause the learning players to encounter many different playstyles and force them to develop more general strategies rather than exploiting a single type of opponent. Computationally expensive opponents could also be added in controlled numbers so the best performing learning players will meet them, and still have reasonable fast experiments. Also matches between non-learning players could be skipped or rated based upon statistical information of earlier matches between the two players. This depending on the type of tournament being played. Doing this might add a bias to the learning though.

Tournament play coupled with neuroevolution shows good potential, but more research needs to be done in order for better players to be evolved.

Bibliography

[Ald02] M. Alden, A. van Kesteren and R. Miikkulainen. Eugenic Evolution Utilizing a Domain Model. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002), San Francisco, 2002, pages 279-286*

[And02] T. Andersen, K.O. Stanley and R. Miikkulainen. Neuro-Evolution Through Augmenting Topologies Applied To Evolving Neural Networks To Play Othello. *Department of Computer Sciences, University of Texas at Austin, 2002*

[And04] G. Andersson. Wzebra (Othello program). *http://www.radagast.se/othello, 2004*

[Bil90] D. Billman and D. Shaman. Strategy knowledge and strategy change in skilled performance: A study of the game Othello. In *American Journal of Psychology 103, 1990, pages 145-166*

[Bur97] M. Buro. The Othello match of the year: Takeshi Murakami vs. Logistello. In *ICCA Journal 20(3), 1997, page 189*

[Del04] R. Delorme. Edax (Othello program). *http://abulmo.club.fr/edax/index.htm, 2004*

[Gom99] F.J. Gomez and R. Miikkulainen. Solving Non-Markovian Control Tasks with Neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99, Stockholm), Denver:Morgan Kaufmann, 1999*

[Gom06] F.J. Gomez, J. Schmidhuber, J and R. Miikkulainen. Efficient Non-Linear Control through Neuroevolution. In *Proceedings of the European Conference on Machine Learning (ECML-06, Berlin), 2006*

[Gro65] A.D. DeGroot. Thought and Choice in Chess. *Mouton, 1965*

[Hsu02] Feng-Hsiung Hsu. Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. *Princeton University Press, 2002*

[Jac05] S.M. Jacobs, T. Mioch, W. Tinus and M. Vuurboom. Envy, Greed and Pride - The three sins of RL Reversi. *Project report, University of Utrecht, 2005*

[Lub01] A. Lubberts and R. Miikkulainen. Co-Evolving a Go-Playing Neural Network. In *Genetic and Evolutionary Computation Conference Workshop, GECCO 2001, San Francisco, 2001, pages 14-19*

[Mor96] D.E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. In *Machine Learning 22, 1996, pages 11-32*

[Mor97] D.E. Moriarty and R. Miikkulainen. Forming Neural Networks Through Efficient and Adaptive Coevolution. In *Evolutionary Computation 5(4), 1997, pages 373-399*

[Per01] A.S. Perez-Bergquist. Applying ESP and Region Specialists to Neuro-Evolution for Go. *Honors Thesis, Department of Computer Sciences, University of Texas, Technical Report CSTR01-24, 2001*

[Pri98] J.W. Prior. Eugenic Evolution for Combinatorial Optimization. *Master thesis, Department of Computer Sciences, University of Texas, Technical Report AI98-268, 1998*

[Rus95] S.J. Russel and P. Norvig. Artificial Intelligence - a modern approach. *Prentice-Hall International Inc, 1995*

[Sta02] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. In *Evolutionary Computation 10, 2002, pages 99-127*,

[Ste96] C. Stergiou and D. Siganos. Neural Networks. In *SURPRISE 96 Journal, http://www.doc.ic.ac.uk/ñd/ surprise_96/journal/vol4/cs11/report.html, Imperial College of Science Technology and Medicine London, 1996*

[Vel99] A. Vellidoa, P.J.G. Lisboaa and J. Vaughan. Neural networks in business: a survey of applications. In *Expert Systems with Applications 17, 1999, pages 51-70*