PSO-Based Coevolutionary Game Learning

by

Cornelis J. Franken

Submitted in partial fulfilment of the requirements for the degree

Magister Scientiae (Computer Science)

in the Faculty of Engineering, Built-Environment

and Information Technology

University of Pretoria

Pretoria

May 2004

PSO-Based Coevolutionary Game Learning

by

Cornelis J. Franken

Abstract

Games have been investigated as computationally complex problems since the inception of artificial intelligence in the 1950's. Originally, search-based techniques were applied to create a competent (and sometimes even expert) game player. The search-based techniques, such as game trees, made use of human-defined knowledge to evaluate the current game state and recommend the best move to make next. Recent research has shown that neural networks can be evolved as game state evaluators, thereby removing the human intelligence factor completely. This study builds on the initial research that made use of evolutionary programming to evolve neural networks in the game learning domain. Particle Swarm Optimisation (PSO) is applied inside a coevolutionary training environment to evolve the weights of the neural network. The training technique is applied to both the zero sum and non-zero sum game domains, with specific application to Tic-Tac-Toe, Checkers and the Iterated Prisoners Dilemma (IPD). The influence of the various PSO parameters on playing performance are experimentally examined, and the overall performance of three different neighbourhood information sharing structures compared. A new coevolutionary scoring scheme and particle dispersement operator are defined, inspired by Formula One Grand Prix racing. Finally, the PSO is applied in three novel ways to evolve strategies for the IPD – the first application of its kind in the PSO field. The PSO-based coevolutionary learning technique described and examined in this study shows promise in evolving intelligent evaluators for the aforementioned games, and further study will be conducted to analyse its scalability to larger search spaces and games of varying complexity.

Keywords: Machine Learning, Games, Particle Swarm Optimisation, Evolutionary Computation, Neural Networks, Iterated Prisoner's Dilemma, Checkers, Coevolution, Computational Intelligence.

Supervisor: Prof. A. P. Engelbrecht
Department of Computer Science
Degree: Magister Scientiae

— *For Surina* —

Preface

Problem Statement and Overview

The study of games as computationally complex systems is not new, and have involved a large number of researchers as well as professional game playing experts in the past [117]. Apart from the game rules, a computer has always largely relied on human knowledge to play a game, with standard data structures and intricate search-based enhancements aiding the man-made evaluation functions to correctly predict the best possible move to make next.

In his original work on the game of Checkers in the early 1950's, Arthur Samuel [114] expressed his hope that future computers will be able to formulate their own evaluation functions and learn to play the game without human intervention. With the modern progression in computing hardware and software performance, Samuel's dreams are now becoming a reality.

Fogel and Chellapilla [53] [27] [26] recently applied neural networks as Checkers evaluation functions in a population of players that were trained through coevolution, with great success. These authors' work combined a number of fields in the computational intelligence domain in order to address a well-known machine learning problem, and formed the inspiration for the experimental work in this study.

The advent of swarm intelligence research, and specifically Particle Swarm Optimisation (PSO) [76] in 1995, have yielded very positive results in a variety of application areas [30]. The investigation on the use of PSO in a coevolutionary game learning context have been limited to comparisons against traditional Evolutionary Programming (EP) approaches [97], as applied by Chellapilla and Fogel [53] [27] [26]. The use of PSO has shown to hold definite advantages over the GP approach to game learning.

This study aims to thoroughly investigate the exact nature of the PSO-based coevolution technique, with specific application to both the zero sum (Tic-Tac-Toe and Checkers) and non-zero sum (Iterated Prisoner's Dilemma) game domains. The results of the study will illustrate the impact on performance of specific PSO parameters choices, neighbourhood information sharing structures and game-specific representations. A number of enhancements to the study of game learning are also presented, including new additions to the areas of benchmarking and coevolution.

Objectives

The main objective of this thesis is to study the application of PSO to the coevolutionary game learning domain. In reaching this goal, the following sub-objectives are identified:

- To provide an overview of existing techniques in the field of game learning, covering both search-based and knowledge-based enhancements.
- To provide an overview of the relevant computational intelligence techniques that form part of the coevolutionary training process, including neural networks and particle swarm optimisation among others.
- To establish a game learning algorithm that makes use of particle swarm optimisation to train neural networks as game state evaluators in a coevolutionary environment.
- To initially investigate the performance of the application of the posited learning algorithm on a computationally modest game such as Tic-Tac-Toe, and later extending the investigation into the much larger problem space provided by the game of Checkers.
- To thoroughly investigate the performance factors involved with the various elements of the game learning algorithm, possibly introducing enhancements to the existing game learning methods in order to improve performance.
- To investigate the versatility of the coevolutionary algorithm by applying it to the interesting non-zero sum game of the Iterated Prisoner's Dilemma.

Contribution

The main contributions of this thesis are:

- The first application and analysis of the Von Neumann PSO neighbourhood information sharing structure in the game learning domain.

- Establishing a consistent testing criteria to determine the performance of an evolved player, by benchmarking against a random-moving player.
- Introduction of a new performance measure that accurately takes into account the number of games drawn, in addition to the number of games won or lost. This is done in order to quantify the playing performance of the evolved player, irrespective of the benchmarking opponent.
- Introduction of certain extensions to the traditional coevolutionary scoring schemes, based on Formula One Grand Prix. The posited Grand Prix methods include a racing scheme that rewards consistent playing behaviour towards the end of training, and a generic scoring system that forms part of a particle dispersement operator that aims to alleviate swarm convergence on suboptimal solutions.
- The first application of a PSO-based coevolutionary game learning algorithm to evolve strategies for the Iterated Prisoner's Dilemma.
- The first application of the Binary PSO algorithm to evolve IPD strategies, as opposed to neural network weights in the traditional PSO-based algorithm.
- The introduction of a third novel scheme to IPD strategy generation, exploiting the symmetrical structure of man-made strategies and making use of the PSO function optimisation abilities.

Thesis Outline

Chapter 1 provides an overview of different types of games, and the core elements found in a game playing software model. The history of these elements is covered as well as different performance improvements that have been discovered during the last couple of decades. The focus then shifts from search-based techniques to knowledge-based techniques, as game learning and various evaluation methods are discussed. A section is dedicated to the best human and machine game players in the world, before lastly examining some of the possible future directions of game research.

The core computational intelligence paradigms that are used in and/or influenced the experimental work for this thesis, are presented in chapter 2. Typical neural network architectures and learning approaches are covered. The core evolutionary computation constructs based on Darwinian evolution are presented, followed by an in-depth discussion of particle swarm optimisation and the influence of its various parameter settings. Finally, coevolution is discussed as a competitive training strategy.

Chapter 3 builds on the background established by the first two chapters, and introduces an algorithm that utilises particle swarm optimisation to train feed-forward neural networks as board-state evaluators. Different methods to estimate the training performance are also presented. Finally, the influence of an increased ply-depth is investigated and the viability of using larger game trees during training is examined.

The training algorithm derived in chapter 3 is applied to the computationally modest problem of evolving intelligent Tic-Tac-Toe players in chapter 4. An overview of the problem is followed by a series of experimental work, analysing various PSO architectures and a selection of basic PSO parameter choices. The training technique is shown to be successful, with the Von Neumann architecture showing promise as neighbourhood structure of choice.

Chapter 5 extends the experimental analysis of the previous chapter to the Checkers domain, which inherently poses a much more difficult optimisation problem due to its much larger search space. The same training algorithm is once again applied, and the neighbourhood structure performance analysed. The Von Neumann architecture proves to dominate once again, but overall results are disconcerting and require further in-depth analysis.

Chapter 6 aims to conduct a detailed investigation into the possible causes for the weak playing performance of evolved Checkers players. The influence of all known PSO parameters are experimentally examined, followed by a look at neural network related performance issues, and various coevolutionary algorithm extensions. In the process, a particle dispersement operator and coevolutionary performance measure based on a scoring system inspired by Formula One Grand Prix are introduced. The chapter concludes with an analysis of stricter training criteria.

After resolving the reasons for the initial poor playing performance, chapter 7 takes the benchmarking of the evolved Checkers players one step further, by analysing playing performance against two 'intelligent' evaluation functions. The impact of training and playing on deeper tree depths are also investigated, ending with a discussion on the possibility of improving the coevolutionary training partner.

Building on the knowledge gained from experiments with zero sum games in the previous chapters, chapter 8 applies the PSO training technique to the interesting non-zero sum problem of the Iterated Prisoner's Dilemma. An overview of the problem and historic work is followed by an investigation of three different strategy generation approaches – all applying PSO in a different context.

The thesis concludes in chapter 9, with a summary of the major experimental findings and a list of future work resulting from the experiments conducted in this study.

A list of appendices provide a glossary of frequently used terms, a summary of symbols, and a list of publications that followed from the presented work.

Acknowledgements

- **Prof Andries P Engelbrecht** for his mentorship and invaluable guidance during the progression of work completed for this thesis.
- **Janneman Franken** for his mathematical insight, MatLab expertise and always stopping whatever he was doing to come and look at my computer screen upon request in the middle of the night.
- **Adi Eyal** for the insightful talks and always questioning my results.
- **Andrew Cooks** for setting up a Linux-based cluster. Without his technical help, a large portion of the simulations would not have finished on time.
- **My friends inside and outside CIRG** for mostly being interested in what I was doing, or asking when the thesis will be finished!
- The financial assistance of the National Research Foundation towards this research is hereby acknowledged. Opinions expressed in this thesis and conclusions arrived at, are those of the author and not necessarily to be attributed to the National Research Foundation.

Contents

Preface **i**

List of Figures **xi**

List of Tables **xiv**

1 Game Learning **1**

1.1 Introduction 1

1.2 Game Types 2

1.3 Game Architecture 3

1.3.1 Game trees 3

1.3.2 Improvements 5

1.3.3 Knowledge representation 8

1.4 Evaluation techniques 10

1.4.1 Hand-crafted evaluations 10

1.4.2 Intelligently weighted evaluations 11

1.4.3 Neural network-based evaluations 11

1.5 Game Learning 12

1.5.1 Data Mining approach 12

1.5.2 Temporal Difference Learning (TDL) 13

1.5.3 Self-play 14

1.5.4 Training with a superior opponent 14

1.6 World Champions 14

1.6.1 Great human players . . . 14
1.6.2 Computer players . . . 15
1.7 Future directions . . . 16
1.8 Conclusion . . . 17

2 Computational Intelligence Techniques 18
2.1 Introduction . . . 18
2.2 Neural networks . . . 19
2.2.1 Structure . . . 19
2.2.2 Training methods . . . 21
2.2.3 Application areas . . . 23
2.3 Evolutionary Computation . . . 23
2.3.1 Data representation . . . 24
2.3.2 Evolutionary operators . . . 25
2.3.3 Models of evolution . . . 26
2.3.4 Application areas . . . 28
2.4 Particle Swarm Optimisation . . . 28
2.4.1 Information sharing structures . . . 30
2.4.2 Parameters: restrictions and influence . . . 32
2.4.3 GCPSO . . . 35
2.4.4 Binary PSO . . . 35
2.4.5 Particle repelling . . . 36
2.4.6 Application areas . . . 38
2.5 Coevolution . . . 38
2.5.1 Population dynamics . . . 38
2.5.2 Credit assignment . . . 39
2.5.3 Application areas . . . 42
2.6 Conclusion . . . 42

3 Training with PSO 43
3.1 Introduction . . . 43
3.2 Basic algorithm . . . 44
3.2.1 Population structure . . . 45
3.3 Measuring performance . . . 46
3.3.1 Playing position in turn-based games . . . 46
3.3.2 Incorporating win, lose and draw . . . 47

3.3.3 Messerschmidt performance measure 48

3.3.4 Franken performance measure 49

3.4 Tree depth 50

3.4.1 Increasing the depth 51

3.5 Final algorithm 52

3.6 Conclusion 52

4 Tic-Tac-Toe **55**

4.1 Introduction 55

4.2 Game rules 56

4.2.1 Scoring structure 57

4.3 Choosing an opponent 57

4.4 Choosing a PSO architecture 58

4.4.1 Parameter selection 58

4.5 Experimental procedure 59

4.5.1 Training algorithm 59

4.5.2 PSO configuration 59

4.5.3 Neural network configuration 59

4.5.4 Setting the benchmark 60

4.5.5 Statistical soundness 60

4.6 Experimental results 60

4.6.1 Comparison of architecture performance matrices 61

4.6.2 Surface plots 64

4.6.3 Increase in hidden nodes 65

4.6.4 Increase in swarm size 66

4.6.5 Convergence 66

4.6.6 Optimising performance 67

4.7 Conclusion 68

5 Checkers **69**

5.1 Introduction 69

5.2 Game rules 70

5.2.1 Scoring structure 72

5.3 Choosing an opponent 72

5.4 Choosing a PSO architecture 73

5.4.1 Parameter selection 73

5.5 Experimental procedure . . . 74
5.5.1 Training algorithm . . . 74
5.5.2 PSO configuration . . . 74
5.5.3 Neural network configuration . . . 75
5.5.4 Setting the benchmark . . . 76
5.5.5 Statistical soundness . . . 76
5.6 Experimental results . . . 76
5.6.1 Architecture comparison matrices . . . 77
5.6.2 Surface plots . . . 79
5.6.3 Increase in hidden nodes . . . 80
5.6.4 Increase in swarm size . . . 82
5.7 Conclusion . . . 83

6 Investigating performance factors **84**
6.1 Introduction . . . 84
6.2 Particle Swarm Parameters . . . 85
6.2.1 Influence of maximum velocity (VMax) . . . 86
6.2.2 Influence of c1 and c2 . . . 87
6.2.3 Influence of inertia weight . . . 88
6.2.4 Influence of LBest neighbourhood size . . . 88
6.2.5 Updated neighbourhood structure comparison matrices . . . 89
6.2.6 Increase in swarm size . . . 92
6.2.7 Increase in hidden layer size . . . 92
6.2.8 Updated surface plots . . . 94
6.2.9 Overall PSO comparison . . . 95
6.3 Neural network input representation . . . 96
6.3.1 Numeric input variations . . . 96
6.3.2 Windowed input . . . 97
6.4 Coevolutionary techniques . . . 100
6.4.1 Introducing Grand Prix methods . . . 100
6.4.2 Race performance . . . 100
6.4.3 Particle dispersement . . . 104
6.4.4 Implementing 'Hall of Fame' . . . 107
6.4.5 Summary of coevolutionary methods performance . . . 110
6.5 Stricter training conditions . . . 111
6.5.1 Influence of training duration . . . 112

6.5.2 Winning under pressure 113
6.5.3 Varying the maximum number of moves 114
6.6 Conclusion 116

7 Assessing intelligence 118
7.1 Introduction 118
7.2 Intelligent evaluation functions 119
7.2.1 Piece-count based evaluation 119
7.2.2 SmartEval – hand-crafted evaluation 120
7.3 Benchmarking Intelligence 121
7.3.1 Self-play 121
7.3.2 Evolved players 122
7.4 Increasing tree depth 124
7.4.1 Q1: Performance against a dominating ply-depth 126
7.4.2 Q2: Does increased complexity outperform increased ply-depth? 127
7.4.3 Q3: Scalability of trained evaluation function 127
7.4.4 Q4: Do deeper training ply-depths require increased complexity? 129
7.4.5 Q5: Compressing game tree knowledge into a neural network 129
7.5 Observations 130
7.6 Improving the training partner 131
7.7 Conclusion 132

8 Iterated Prisoner's Dilemma 133
8.1 Introduction 133
8.2 Historic overview 134
8.2.1 The prisoner's dilemma 134
8.2.2 Related work 137
8.3 Choosing an opponent 140
8.4 Training algorithm 141
8.5 Strategy generation 142
8.5.1 Evolving neural networks 143
8.5.2 Evolving strategies through Binary PSO 145
8.5.3 Evolving strategies by exploiting symmetry 146
8.6 Experimental procedure 150
8.6.1 Measuring performance 150
8.6.2 Noisy environments 152

8.7 Experimental results 153
8.7.1 Parameter choices 153
8.7.2 Population performance 156
8.7.3 Strategy performance 163
8.8 Conclusion 167

9 Conclusion and future work **172**
9.1 Conclusion 172
9.2 Future work 174

Bibliography **177**

A Glossary **190**

B Definition of Symbols **193**

C Derived Publications **195**

List of Figures

1.1 A simple game tree 4

2.1 Abstract representation of a simple ANN structure. 20
2.2 Various activation function plots used by ANN nodes. 21
2.3 Various PSO neighbourhood information sharing structures. 30

3.1 Abstract training algorithm. 44
3.2 Detailed training algorithm. 54

4.1 Typical end-game sequence for a Tic-Tac-Toe game. 56
4.2 Comparison graph between Messerschmidt *et al.*'s implementation performance and performance values computed in this thesis. Larger M-values are preferred. 63
4.3 Surface plot for architecture comparison. 64
4.4 Increase in hidden nodes for specific LBest architecture configuration. 65
4.5 Increase in swarm size for specific LBest architecture configuration. 66
4.6 Convergence of neural network weights. 67
4.7 Optimised performance for a sample LBest configuration. 68

5.1 Dimensions, numbering and setup of a standard Checkers game board. 70
5.2 Basic rules of Checkers explained. 71
5.3 Surface plot for PSO structure comparison in Checkers. 81
5.4 Increase in hidden nodes during Checkers simulations. 81
5.5 Increase in swarm size during Checkers simulations. 82

6.1 Influence of maximum velocity with trend-line. 86
6.2 Influence of c_1 and c_2 parameters. 87
6.3 Influence of inertia. 88
6.4 Increase in swarm size. 92
6.5 Increase in hidden layer size. 93
6.6 Improved performance surface plot. 93
6.7 Individual gain by each neighbourhood structure. 94
6.8 Overall PSO neighbourhood structure comparison. 95
6.9 Various window formation schemes. 98
6.10 Influence of window formation. 99
6.11 Adapted training algorithm to include GPX racing. 102
6.12 Adapted training algorithm to include particle dispersement. 106
6.13 Influence of dispersement interval. 107
6.14 Influence of ‘Hall-of-Fame’ survival rate. 109
6.15 Performance of various scoring structures. 110
6.16 Influence of training duration. 112
6.17 Winning in less than 50 moves. 113
6.18 Influence of a restricted move count. 114
6.19 Win/Draw/Lose relationship. 115

7.1 Benchmark results for the intelligent evaluation functions. 121
7.2 Performance of evolved players against intelligent evaluation functions. 123
7.3 Increasing configuration as a substitute for tree depth. 126
7.4 Training and playing on deeper ply depths. 128

8.1 Complete IPD training algorithm. 142
8.2 First 32 bits of the TFT strategy represented as a sinusoidal wave. 148
8.3 First 32 bits of the TFTT strategy represented as an adjusted sinusoidal wave. 148
8.4 Combination of sin and cos function to be optimised. 149
8.5 Constructing symmetry in four different ways. 149
8.6 Increase in hidden nodes versus swarm size for Neural Network approach. . . . 154
8.7 Influence of inertia term for BinPSO and symmetry approaches. 154
8.8 Influence of the maximum velocity for BinPSO and symmetry approaches. . . . 155
8.9 Catastrophic collapses occurring in neural network-based population. 156
8.10 Average personal performance of best agent using the BinPSO approach over 500 epochs. 159

8.11 Average personal performance of best agent using the Symmetry approach over 500 epochs. 159
8.12 Average personal performance of best agent using the Neural Network approach over 500 epochs in a noisy environment. 161
8.13 Average personal performance of best agent using the BinPSO approach over 500 epochs in a noisy environment. 162
8.14 Average personal performance of best agent using the Symmetry approach over 500 epochs in a noisy environment. 163

List of Tables

3.1 Probabilities for a random-moving Tic-Tac-Toe player. 47
3.2 Probabilities for a random-moving Checkers player. 47

4.1 Architecture performance matrix - standard GBest. Larger M-values preferred. 61
4.2 Architecture performance matrix - Local Best. Larger M-values preferred. . . . 62
4.3 Architecture performance matrix - Von Neumann. Larger M-values preferred. . 63

5.1 Architecture performance matrix - standard GBest. Larger M and F-values preferred. 78
5.2 Architecture performance matrix - Local Best. Larger M and F-values preferred. 79
5.3 Architecture performance matrix - Von Neumann. Larger M and F-values preferred. 80

6.1 Performance of various methods to restrict the maximum velocity. 86
6.2 Performance of various LBest neighbourhood sizes. 88
6.3 Architecture performance matrix - Updated standard GBest. Larger M and F-values are preferred. 89
6.4 Architecture performance matrix - Updated standard LBest. Larger M and F-values are preferred. 90
6.5 Architecture performance matrix - Updated Von Neumann. Larger M and F-values are preferred. 91
6.6 Performance of various GCPSO search term sizes. 95
6.7 Performance of sliding-window formation techniques. 99

6.8 Performance of various GPX scoring systems. 104
6.9 Performance of various re-dispersement sizes over 50 intervals. 105
6.10 Performance values for various 'Hall of Fame' sizes. 108
6.11 Performance of various invocation intervals for the HOF. 110

8.1 General form of the IPD payoff matrix. 135
8.2 Payoff matrix for the IPD as used by Axelrod. 136
8.3 Binary representation of strategies, illustrating symmetrical properties. ('Op' refers to the opponent, and 'My' to the local player) 147
8.4 'Man-made' strategy benchmark – no noise. 151
8.5 'Man-made' strategy benchmark – 5% noise. 152
8.6 Neural network cooperation results for 20 simulations – noiseless environment. 158
8.7 Binary PSO and Symmetry Approaches' cooperation results for 20 simulations – noiseless environment. 158
8.8 Neural Network approach's cooperation results for 20 simulations – 5% noise applied. 160
8.9 Exceptions to superb performing Binary PSO and Symmetry Approaches' cooperation results for 20 simulations – 5% noise applied. 162
8.10 Best performing individuals, averaged over 20 runs and 10000 games per run – no noise. 164
8.11 Performance of individual man-made strategies - no noise. 169
8.12 Best performing individuals, averaged over 20 runs and 10000 games per run – 5% noise applied. 170
8.13 Performance of individual man-made strategies - 5% noise. 171

CHAPTER 1

Background on Game Learning

"The programs may indeed consider a lot of moves and positions, but one thing is certain. They do not see much!"
- Dr Marion Tinsley, 1980

This chapter presents an overview of different types of games, and the core elements found in a game playing software model. The history of these elements is covered as well as different performance improvements that have been discovered during the last couple of decades. The focus then shifts from search-based techniques to knowledge-based techniques, as game learning and various evaluation methods are discussed. A section is dedicated to the best human and machine game players in the world, before lastly examining some of the possible future directions of game research.

1.1 Introduction

The aim of this chapter is to provide the reader with a solid frame of reference regarding the history of game playing technologies, starting with the fundamental distinction between *perfect information* and *imperfect information* games in section 1.2. The game tree as core data structure is introduced in section 1.3, along with various enhancements to search techniques resulting from more than four decades of research.

An overview of some of the more state-of-the-art knowledge-based techniques is also provided in sections 1.4 and 1.5, which covers the different approaches to intelligent game state evaluations as well as some of the more mainstream game learning approaches.

Section 1.6 looks briefly at two of the best human game players of recent times, *Dr Marion Tinsley* (in Checkers) and *Garry Kasparov* (in Chess). Their machine counterparts, *Chinook* (in Checkers) and *Deep Blue* (in Chess) are also acknowledged.

The chapter concludes with section 1.7 briefly discussing some possible future directions for research in games, mentioning some of the well-known search space constraints. A selection of games that exploit these constraints is listed, that while remaining unsolved, also forms the 'holy grail' for games research.

1.2 Game Types

The majority of research conducted in the game playing domain can be subdivided into two distinct application areas, namely two-player *perfect information* and *imperfect information* games. The term *perfect information* refers to the fact that all the information about the game is available to the game players at any moment in time during the progression of the game. For instance, all the Chess-pieces that are currently in play during a standard Chess game is visible at all times. There is no hidden component or any form of randomness in perfect information games. Other examples of perfect information games include Checkers, Tic-Tac-Toe, Othello and Go. *Perfect information* games are usually modelled using a game tree architecture and some form of intelligent evaluation function.

Imperfect information games have a hidden and/or random element in their nature. For instance, even though all the pieces necessary to play Backgammon is visible to both players at all times during the game, it has a random element – introduced by the throw of a dice that determines the speed of progression around the board. Other examples include the games of Poker and Bridge, where the players are not able to see what cards their opponents are holding. *Imperfect information* games have been successfully modelled using simulation based techniques, such as Monte Carlo methods [117] as well as temporal difference learning algorithms in the case of Backgammon [130] [131].

It is also important to note that single-player games (or puzzles) have received a certain amount of research attention, with advances made in solving Sokoban problems, Rubik's cube and completing crossword puzzles [117].

Lastly, some games have been 'solved' by expert programmes and dedicated algorithmic research [117]. A solved game implies being able to play as either the first (starting) or second player, and never losing a match. The computer is able to select the best move in any given situation, thereby playing a 'perfect game'. A number of games have already been solved, including Nine Men's Morris, Connect-4, Go Moku, Qubic and Tic-Tac-Toe [117].

The remainder of this thesis focuses on game playing techniques pertaining to two player *perfect information* games, with specific application to Tic-Tac-Toe in section 4.2 and Checkers in section 5.2.

1.3 Game Architecture

Ever since the seminal papers in the early 1950's by Claude Shannon [121] and Alan Turing [135] on how to program computers to play games, a great deal of work has been conducted in using games as a test-bed for artificial intelligence and computational research. The basic architecture used by the majority of programmers to implement the game simulator (apart from the game rules) includes the use of a game tree and corresponding optimised search method, some way to represent the knowledge about the game (playing strategies), as well as an advanced intelligent evaluation function (originating from either human or machine expertise).

Section 1.3.1 introduces the game tree data structure, its history and numerous related search improvements. Various knowledge representation methods are discussed in section 1.3.3. Evaluation techniques are covered in section 1.4.

1.3.1 Game trees

John von Neumann first described the basic structure of the game tree in 1928, while doing research in game theory [143]. It is based on a general tree data structure with some branching factor, a root node, children and leaf nodes. Given a game played by players A and B, and the current board state with player A having to make the next move, the game tree (see figure 1.1 for a visual representation) has the following structure:

- Root node: the current board state.
- Immediate child nodes (1 ply depth): board states generated from all possible move combinations for player A.
- Grandchildren nodes (2 ply depth): board states generated indicating all the possible moves made by player B in reply to each of the previous player A moves.
- Repeat interleaving of generated moves on subsequent depths until the complete tree is constructed, or some stopping criterion is met.

Each of the nodes in the game tree thereby contains a specific possible board state in the future progression of the game. In order to determine the best possible choice for player A, each of the

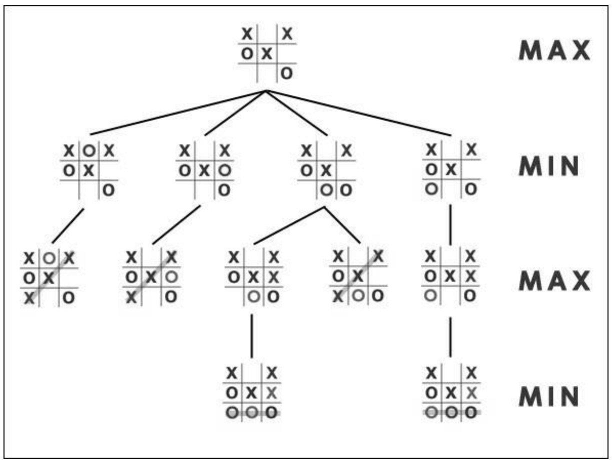


Figure 1.1: A simple game tree

board states in the tree has to be evaluated according to an evaluation function. A high value returned by the evaluation function indicates 'good' moves, whereas low evaluations indicate 'bad' moves.

Following a bottom-up approach and starting with the leaf nodes of the tree, it is the aim of player A (also referred to as the MAX player) to maximise the possible evaluation values that gets passed up the tree, and the aim of player B (also referred to as the MIN player) to minimise the possible evaluation values passed up the tree. The result is a strategy where MAX makes all the moves that best improve his position, taking into account that MIN will reply with the best possible move (limiting the damage that MAX can inflict on MIN). This sequence of moves in a game tree is also referred to as the *principal variation*. A typical Minimax game tree algorithm can be found in [101].

Building a Minimax game tree for a game such as Tic-Tac-Toe is not very resource dependent, since the complete search space for all possible moves remains quite small and can easily be generated and maintained in system memory. For more complex games such as Chess though it is estimated that the total number of legal moves in a typical game amounts to 10^{40} moves [29]. Clearly, this is too much information to store in system memory, and will take immensely long to calculate. The initial attempts to limit the number of nodes that has to be calculated and evaluated (through pruning of the search tree) appeared independently and seemingly 'unknown' in 1963 by Brundo, a Russian scientist [21]. It was not until the seminal paper by Knuth and Moore in 1975 [81] that a technique now known as *Alpha-Beta pruning* was formalised and its performance analysed.

Various other search techniques were also introduced during this time period, all of them improving on the standard Minimax and some even on standard Alpha-Beta. The best-first search SSS* algorithm was developed by Stockman in 1979 [127], and was theoretically proven to dominate Alpha-Beta, resulting in fewer node expansions. Its implementation however required a sorted OPEN list, similar to the traditional A* graph search algorithm, and apart from the immense memory requirements it also demanded a lot of overhead for inserting/deleting nodes. It has not been widely adopted by researchers.

Pearl introduced the Scout algorithm in 1980 [102], which improved on Alpha-Beta's performance due to the fact that it doesn't compute the final Minimax node values when initially traversing the tree. Instead, it uses a boolean function to approximate the final node values and thereby eliminates unnecessary moves further down the tree. The remaining tree is then re-searched in order to compute the final node values.

Due to the fact that game tree searching is usually time limited (a player is only allowed a certain amount of time to 'plan' the next move in a regulation match), a technique called *iterative deepening* has been introduced [84]. It allows for successive searches of the tree at increased depth bound - ensuring a complete search of the partial tree (up to the previous depth) before a decision is made. This avoids the problem of searching too deep into the tree and missing some 'vital' moves that are located in the opposite (and as yet unsearched) portion of the tree.

1.3.2 Improvements

Further improvements on the existing base of Alpha-Beta or Scout-related algorithms can be categorised into two main types. Section 1.3.2 examines some *move ordering* techniques used inside the game trees. Section 1.3.2 lists more *search-based enhancements*, which improves performance on the already mentioned algorithms.

Move Ordering

The aim of move ordering is to enable a depth-first search algorithm to search through the 'best node' first, thereby causing a cut-off on sibling nodes and limiting the number of unnecessary nodes that are searched. The obvious problem with move ordering is that no accurate measure of 'best node' is available. If it was available, then a game tree had no purpose and the algorithm could be used on its own to solve the game. Various move ordering enhancements exist, including:

- **Hash-tables (Transposition tables) [62]**: While searching through the game tree, the result of searched nodes (actual moves leading to this position) and their evaluations are stored in a hash-table data structure. When re-searching the tree (due to iterative deepening or on subsequent turns) the hash-table is first queried to find the best possible move from the current position (based on previous searches), before an actual tree search is undertaken. When new areas of the tree are encountered, the positions in the hash-table are updated. Hash-tables also allow for move ordering due to the availability of node evaluations before actually re-searching them – thereby allowing deeper searches into more interesting parts of the tree.

- **Killer moves** [94] [1]: Certain situations in a game may require immediate response, such as a threat of checkmate in 1 move. It is almost guaranteed to assume that whatever move is able to refute the checkmate threat, is also a good move to continue searching from. Killer moves are computed and executed, but not stored.

- **History heuristic** [116]: The history heuristic is similar to the use of hash-tables, but instead of storing the actual move and its score (the move context), only a counter is updated to indicate that a move proved to work 'well'. The counters are stored in a matrix, indexing '*source square*' and '*destination square*' move fields. This heuristic keeps track of the impact of killer moves and other 'good moves' in a long-term memory data structure, but it doesn't store the context of the moves as is done in hash-tables.

- **Static ordering**: Static ordering is a game dependent heuristic to ensure that the best move is searched first. For example, in Checkers you may want to order moves so that promotions are searched first, or that piece-captures have preference – ordered by the number of pieces captured in a single turn.

Search Enhancements

The previous optimisations focused on the occurrence of nodes in the game tree, and how to order them optimally to increase the efficiency of Alpha-Beta. The following enhancements make actual changes to the Alpha-Beta algorithm, opting to efficiently search through the game tree by visiting the fewest number of nodes due to pruning. Various enhancements exist, including:

- **Aspiration search (Windowing)**: The traditional Alpha-Beta algorithm starts its search with a search-window of size [–INFINITY : +INFINITY]. It is possible to resize this window to [–WINDOW : +WINDOW] before searching, assuming the new bounds will approximate the eventual optimal window size, thereby causing many more cut-offs earlier on in the search. Obtaining this window size beforehand could be a problem, with the end result being a repeated search of the game tree with increased window size.

- **Principle Variation Search (PVS [93], NegaScout [110])**: Principle Variation Search makes use of the previous assumption that the window size can be effectively reduced by extending it to adapt the window for interior nodes in the game tree. PVS relies on the effectiveness of the aforementioned move ordering techniques to cause the best move to be searched first. A very small window around the best move is thus necessary. If it can be shown that the first node is indeed inferior, the search window is increased to include the new nodes and the search is repeated.

- **MTD(f) [104]**: MTD(f) uses a succession of null window calls on the Alpha-Beta algorithm to determine the true value of the root node. It stores the results of previous searches and through iterative deepening is able to *guess* the window size and thereby compute the true Minimax value, evaluating less nodes.

- **Enhanced Transposition Cutoffs (ETC) [103]**: Enhanced Transposition Cutoffs come into play when implementing a hash-table. Normally the program will first check whether a move on a move-list has a hash-table entry and if not, conduct a search to determine the node value that could possibly result in a cutoff. ETC simply dictates that you search through your *complete* move list for hits in the hash-table, before conducting any new searches. This only improves performance when there is a relatively large part of the tree that can be pruned in this manner – otherwise it proves to be too costly.

- **Singular Extensions (SE) [3]**: Some nodes in the game tree are more promising than others, which leads to the desire to extend these lines of play further and investigate their use. The aim of SE is to identify these interesting lines by looking at forced moves of play. It identifies forced moves by adjusting the search window to evaluate if the best move is significantly better than the second-best move. The search depth is then increased along the more interesting/significant nodes.

- **Game specific enhancements (Null moves) [13]**: The concept of null moves is based on the premise that every move improves your position in the game. During the progression of the search the opponent is given the opportunity to make two consecutive moves (instead of the usual one). If the new position is still in the current player's favour, even after the opponent's two moves, the search for that line is terminated – indicating that it is already 'good enough' and need not be completely investigated.

Another improvement on game tree search is the use of parallel or distributed search algorithms, running on large clusters of processors and parallel architectures. The interested reader is referred to [49] for more information on the successful application of parallel search in the ZUGZWANG Chess program, using 1024 processors.

1.3.3 Knowledge representation

Up to now, the discussion has focused on the game tree and its related optimisations. The game tree is able to generate a large selection of potential successor moves from the current board state, but does not consider existing theoretical knowledge (such as opening theory) derived by educated scholars through centuries of analysis, in order to select the best node. Very large annotated databases of games played by Grand Masters exist around the world, for various popular games [28]. It is possible to utilise this knowledge and construct different playing strategies depending on the current phase of the game.

Two types of knowledge structures are typically used by high standard game playing simulators, namely the *opening book* and an *end-game database*. Researchers working on Deep Blue also recently introduced the concept of an *extended book*[23]. Each of these structures is discussed in turn below.

Opening books

Most games can be divided into three main phases, namely the opening, middle game and end-game [118]. The purpose of the opening book is to establish a solid start to the game being played. It encompasses a vast number of board positions and includes a series of well-known and expert opening moves. This should provide the computer with some advantage over amateur players, but keep it on par with expert players who also study so-called 'opening theory'.

While working to create structures that add knowledge to Chinook (see section 1.6 for more information on this world champion Checkers program), the team from the University of Alberta [118] realised that there are a couple of sure-losses (moves that lead to definite failure after 20 or more moves in play) as early as the 4th move in the game. This clearly falls in the domain of the opening book, but by sticking to only making moves from the opening book caused Chinook to lose some of its creativity, which was a serious advantage against world-class players. However, by refraining from closely following the opening book Chinook was in danger of losing the game early on.

They constructed what they called an anti-book [118]. Instead of just maintaining a database of 'good' moves as defined by Checkers experts, they maintained a database of moves that lead to forced failure later on in the game. This allowed them to maintain creativity, while avoiding any serious pitfalls.

Some efforts have also been made to learn the contents of the opening book from scratch, as opposed to manually adding the entries by hand. Buro [22] presents a dynamic growing opening book, based on games played against humans and other computers on-line (he used Othello as implementation, but this could easily scale to other games). A number of difficulties were encountered, including the use of varying sizes of opening books. When using a large opening book and competing against a smaller opening book, it is difficult to estimate exactly what the smaller book lacks by only playing against it. The possibility that the larger opening book may contain some unique superior strategies (a huge advantage) is thereby not used to its full potential. He proposes to model smaller opening books in order to make effective use of the large opening book. Other problems included the physical size of the book in memory, as well as handling the occurrence of directed acyclic graphs.

Finally, Buro makes the interesting distinction between public and private draws. Private draws can become winning situations if the opponent makes a mistake, whereas public draws are unavoidable (draws). The relevance of draws in games is covered in more depth in section 3.3.2.

Extended book

The dedicated team of researchers at IBM that worked on the 'Deep Blue' project introduced the concept of an extended book [23]. The extended book is used to guide the program in selecting its opening book moves, as opposed to using some random measure. This is done through retrieving knowledge from a database of Grand Master games, and constructing a new small database. This allows 'Deep Blue' to determine the opening playing style of its opponents by actually 'studying' their past games, and consequently use the correct counter opening moves to ensure a solid start to the game. The team reported the extended book to be a huge success while playing against Garry Kasparov in 1997.

End-game databases

Another thoroughly studied theoretical aspect of games is the end-game. Players spend hours training to resolve predefined problem positions in order to win the game. Comprehensive knowledge of end-game positions and their successful resolution is a serious requirement in any game played at master level.

For simulation programmers, the use of end-game databases has become obligatory. A typical end-game database contains the complete search tree for any board position with six pieces or less. This has been achieved for Chess [134], with the 8-piece database available for Checkers [118]. The larger databases require enormous storage space, and are usually maintained off-line (not in system memory). Various compression schemes are under investigation to try and reduce the size of the databases [118]. As soon as the end-game database comes into play, it is possible to play a near perfect game. However, depending on the entry state, a win is still not guaranteed.

Chinook is able to extend its search from the last opening-book moves, through the middle-game (usually 20 or 30 ply deep) and into the end-game database. This allows Chinook to use perfect knowledge very early on in the game, and is a major contributor to Chinook's very good overall playing performance.

1.4 Evaluation techniques

Probably the largest single element that allows for 'intelligent' behaviour during gameplay is the evaluation function. It is the evaluation function that makes decisions in the game tree, chooses the correct opening moves from the opening book, and approximately defines the principal variation path during initial searches reaching the end-game database. In fact, Chinook had four different evaluation functions for four different phases of the game (opening, middle-game, early end-game, late end-game) [118].

There exist different methods of constructing an evaluation function. The following sections summarise the most popular approaches, namely hand-crafted functions in section 1.4.1, intelligently weighted functions in section 1.4.2 and finally, neural network-based evaluations in section 1.4.3.

1.4.1 Hand-crafted evaluations

Arthur Samuel listed in his seminal paper in 1959 [114], an evaluation function consisting of 26 distinct game features that could be used to evaluate the current Checkers board state. These features could be weighted according to their importance for the current phase of the game. One of the important questions therefor is how to pick a set of features that can accurately describe the 'value' of the current board state.

Traditionally, authors would refer to existing game literature and hard-code certain important defensive/offensive positions, rewarding the computer player when maintaining them. For example, in Checkers, as in Chess, it is beneficial to control the centre squares, because it provides the most mobility. Immediately, two features have been defined: *centre-control* and *mobility*. Many other features exist [114].

More recently the aid of Grand Master level players have been used to construct evaluation functions for Checkers [118] (Chinook) and Chess [23] (Deep Blue). They rely on their concise theoretical knowledge to identify a set of features that are manually fine-tuned to achieve the desired style of gameplay.

It should be noted that a trade-off exists between speed of searching and complexity of the evaluation function. As already mentioned, a limited time is awarded to plan a move in a regulation match. The program typically aims to search as deep into the game-tree as possible. By using a very complex evaluation function, valuable computation time is spent on intricately evaluating board states that might never be used, thereby restricting the depth that can be searched. Section 3.4.1 examines this trade-off more closely.

1.4.2 Intelligently weighted evaluations

Up to now, all the 'intelligence' attributed to the game simulator is actually a direct result of human knowledge/expertise translated into code form. Recent studies have examined techniques that allow game-playing programs to benefit from their own experience. The first category of improvements aims to automatically adjust the weight values for predefined features in the evaluation function.

Kendall and Whitwell [73] used evolutionary strategies to evolve a population of Chess evaluation functions. They seeded the population with initial piece-value weights as provided by Shannon [121]. These weights were adapted through the use of basic evolutionary principles, where the strongest evaluation function of the current generation influenced the future genetic make-up (weight values) of the population. Both the 'undeveloped' and 'developed' players were tested against commercial software, with decent results.

Another approach has been to adjust the weights according to Temporal Difference Learning (TDL). TDL is discussed in more detail in section 1.5.2. The result of the application of TDL to adjusting weights is best described by an extract of correspondence between Don Daily and Jonathan Schaeffer [117]:

> *'Much to my surprise, TDL seems to be a success. But the weight set that comes out is SCARY; I'm still afraid to run with it even though it beats the hand-tuned weights. They are hard to understand too, because TDL expresses Chess concepts any way that is convenient for it. So if you create a heuristic to describe a Chess concept, TDL may use it to "fix" something it considers broken in your weight set.'*

1.4.3 Neural network-based evaluations

The second category of improvements has focused on using neural networks *as* evaluation functions. This evaluation method has successfully been used by Tesauro [130], Blair and Pollack[18] in their approaches to play Backgammon. Fogel and Chellapilla [53] [27] [26] pioneered the *evolutionary training* of neural network-based evaluation functions within the game learning domain with their Anaconda (also known as Blondie24) Checkers-playing program. The board state (or permutations thereof) would be directly input to the network, and the output corresponded to an evaluation value. Anaconda competed against human players through an on-line games portal, beating master ranked players [53]. It also proved to be superior against a commercially available software program [27].

The training methods employed in this thesis are loosely based on the groundwork provided by Fogel and Chellapilla. The details of their implementation will be discussed in the various relevant sections throughout this thesis. The neural network paradigm is covered in section 2.2. The interested reader is referred to an exciting account of the work that went into developing Blondie24 in [53].

1.5 Game Learning

Even though Arthur Samuel provided his own set of features for the evaluation used by his Checkers program [114], he still believed it should be possible for the computer to learn/discover its own set of features from repetitive play. After all, Samuel published all his Checkers papers under the *Machine Learning* subject heading, another hint at his yearning for true intelligence in games. The hardware limitations at the time sadly restricted these ideas, but the surge in development of faster processors and larger storage capacities in recent years have made this ideal a reality.

The following subsections deal with various approaches to learn new features to be used in an evaluation function, or new methods of play that induce learning behaviour. The interested reader is referred to a much broader and in-depth analysis of the most notable approaches in game learning, in a chapter by Fürnkranz and Kubat [59].

1.5.1 Data Mining approach

Data Mining usually involves the analysis of very large collections of information in order to extract knowledge in the form of rules, decision trees or clusters – indicating common trends or interrelationships in the data. As was already mentioned, large collections of annotated grand master matches exist for various games. These databases provide the ideal platform for Data Mining experiments in retrieving important features, or adjusting weight values of existing features, that humans have not yet discovered or understood. Not a lot of successful research has however been conducted in mining game databases.

The Chinook team [118] used a database of 800 grand master games to tune their evaluation function. They tried to compare how many times the computer would make the same move a grand master made in a historical game, and reward it appropriately. It did however occur that the program discovered moves that were *better* than the grand master moves, which was initially attributed to a programming error – since the moves were not well-known. This points out that the human understanding of the game is still limited in a lot of ways, where a 'good' move is only considered to be good if it was played by a grand master.

As was previously mentioned in section 1.3.3, the Deep Blue team [23] mined the information contained in large Chess databases to construct an *extended book*. The extended book aided the program to select the appropriate opening move from the opening book, while still adhering to good opening theory.

Van Rijswijck conducted experiments on the ancient African game of Awari [141]. Using perfect information databases (end-game databases of states containing up to 35 pebbles), he was able to evolve decision trees that acted as evaluation functions. It performed very well against software with hand-crafted evaluation functions, winning each of its games convincingly.

Kojima *et al.* [83] used a Data Mining approach to retrieve patterns and sequences from large Go databases. They were able to extract useful knowledge that was examined by a Go expert. The expert randomly selected and analysed 27 (1%) of the extracted rules, and classified them to either be 'good', 'average' or 'strange'. Of the 27 rules, 14 (50%) were classified as 'good', 6 (22%) as 'average' and 7 (26%) as 'strange'. However promising these results may be, a lot of Go research still has to be completed in order to properly compete with human world champions.

1.5.2 Temporal Difference Learning (TDL)

The concept behind TDL was first described by Samuel [114]. It was formalised by Sutton [129] and applied to the game learning paradigm by Tesauro in his world-champion Backgammon program [130] [131]. TDL is a reinforcement learning algorithm. It maps inputs (typically the board state) to a set of outputs (board state evaluations), and given the final result of the game (win, lose or draw) reinforces the move sequences that lead to a positive result, and punishes the move sequences that lead to a bad result. Tesauro's TD-Gammon program used only a single-ply game tree, in combination with the TDL algorithm to evolve into a world-champion Backgammon player – beating its human counterparts.

TDL has also been applied to the game of Go [34], where it was used to train neural networks to estimate the safety of groups, as well as to estimate the territorial potential of unoccupied points – with encouraging results.

1.5.3 Self-play

It is possible to train an agent to play a game from zero knowledge – apart from the basic game rules. This is the culmination of Samuel's ideal. Fogel and Chellapilla [53] [27] [26] trained a population of Checkers playing agents through coevolutionary techniques. The agents used neural networks to evaluate the nodes (board states) of a game tree. Through self-play in a basic tournament scheme, the agents were able to evolve from mediocre novices into skilled adversaries, able to defeat master level players. The inner workings of coevolution is covered in more detail in section 2.5.

1.5.4 Training with a superior opponent

As an alternative to self-play, it is possible to train an agent against an already *intelligent* opponent. Various studies in coevolutionary environments [18] have been conducted to determine what types of adversaries promote the biggest opportunity for learning [6] [47]. Section 7.6 takes an in-depth look at the different types of training partners, and provides some experimental results to illustrate its application in Checkers.

One of the observations made by researchers working on training Go agents against the open-source GnuGo software [91], is that the evolutionary process stops when the intelligent player is able to defeat the hard-coded opponent. Coevolved players (through self-play) exhibited much better performance against the GnuGo software and was able to beat it a lot quicker than their custom-trained counterparts.

1.6 World Champions

Throughout this chapter various references have been made to world-champion human and computer players. It is perhaps fitting to dedicate a small section of this thesis to the people and programs that continuously drive research in games, while setting new standards of skill and computational power in the process.

1.6.1 Great human players

Kasparov

Garry Kimovich Kasparov [147] was born in 1963, and already at the age of 7 showed promise as a Chess prodigy. He became the youngest player to win the Soviet Junior Championship – aged 12. He won the World Junior Championship in 1980, and became the youngest World Chess Champion in 1985, beating Anatoli Karpov. Between December 1981 and February 1991

Kasparov did not lose a single Chess event he participated in. He has remained the world's best ranked Chess player for approximately 18 years.

Kasparov has invested a lot of time and effort introducing Chess to schools and children – founding the World Schools Chess Championships, while also supporting the Mentor Foundation and various other charities. He has always been interested in the use of computers to play and analyse Chess. This interest lead to his involvement in the creation of ChessBase [28] (a very large commercial database of Chess games). He participated in various Man vs Computer (most notably against Deep Blue from IBM), Man & Computer vs Man & Computer, Internet ('*Kasparov against the World!*') and other highly publicised computer tournaments.

Tinsley

Dr Marion Tinsley [115] is unanimously accepted as the greatest Checkers player of all time. His unblemished record of losing only 7 games in 45 years remains unparallelled in any modern sport or competition. This mathematics professor from Florida was universally liked and respected for his kind and gentle manner, responding enthusiastically to Checkers masters and novice players alike.

After returning from 12 years of retirement – due to a lack of competition – he joined the Chinook team from the University of Alberta to aid in the creation of the world champion Checkers program. He resigned his title as World Champion in order to play the initially unsanctioned challenge match against Chinook in 1992. He won the contest (4 wins, 2 losses and 33 draws). A repeat challenge was hosted in 1994, but after 6 games (all draws) Dr Tinsley had to resign for health reasons. He was diagnosed with cancer that following week and sadly passed away 7 months later. He is remembered fondly.

1.6.2 Computer players

Chinook

Developed by the University of Alberta in the early nineties, Chinook [118] became the first computer program to receive a world championship title after playing against a human. Chinook's strength relies on its ability to search very deep into the game tree, sometimes even reaching the pre-computed 6-piece end-game databases, thereby allowing it to make use of 'perfect knowledge'. Chinook also made use of an opening book and fine-tuned evaluation function – partially obtained through the assistance of grand master players, including the great Dr Marion Tinsley. Chinook has since been retired, but has set a new standard for world-class Checkers playing programs.

Deep Blue

Deep Blue [71] was developed by IBM, and received widespread public attention during the two exhibition tournaments against Garry Kasparov in 1996 and 1997. Even though the ultimate result of their first encounter was a win for Kasparov, Deep Blue came back to level the score in the second tournament. Deep Blue made use of purpose built hardware, capable of evaluating approximately 200 million board states per second. The development team was supplemented by grand master Chess players that aided in the design of Deep Blue's theoretical knowledge (evaluation function and move databases). IBM decided to dismantle Deep Blue after the 1997 match, but its legacy still continues to inspire Chess enthusiasts all over the world.

Deep Junior

What initially started out as a hobby for two Israeli programmers, turned into a very successful commercial product in 1997. Deep Junior [41] has been hailed as the best Chess program available today, having won the title of Absolute World Computer Chess Champion in Maastricht in 2002. Garry Kasparov used it as an analysis tool for his Microsoft-sponsored exhibition tournament on the Internet ('*Kasparov against the World!*'). It has beaten numerous grand master human players, and recently played against Kasparov in an exhibition tournament. Due to its commercial nature, not a lot of information on the inner workings of Junior is publicly available. Its strongest feature is its focus on intelligent play rather than brute force methods. Deep Junior's playing style has been positively described as being 'daring', 'creative' and sometimes appearing as if it was playing on a 'hunch' – regularly catching its opponents off-guard. It remains the latest benchmark for state-of-the-art computer Chess software.

1.7 Future directions

The applicability of computational Chess research has been questioned after the Deep Blue matches [40], and it seems as if researchers are focusing on other types of more 'interesting' games. With the advent of games like Arimaa [7] that purposefully exploit the general inability of computers to handle very large search spaces, the future for games research luckily remains bright. More and more research efforts are poured into the challenging ancient oriental game of Go (estimated to have a search space upward of 10^{170} moves), which with the current hardware limitations remains the 'holy grail' for simulation writers.

As a final comment, it is interesting to note the reply of Kasparov in an interview with New Scientist magazine[80], when asked whether he can distinguish between different high-end Chess programs by just playing against them:

'I can identify immediately Deep Junior or Deep Fritz. Every machine has its own character, undoubtedly. The machines' evaluation process is based on the priorities, and each set of priorities is different. If you place emphasis on one particular mode of play in the program as Deep Junior's program does, the machine gets more adventurous. If you have another style, you have different machine personalities. Today, any professional could tell Deep Fritz from Deep Junior in 10 to 15 moves. It's as easy as differentiating between Kramnik and Kasparov.'

1.8 Conclusion

This chapter aimed to provide background information on existing game playing technologies. The focus of this thesis was established as presenting research on two-player perfect-information games, with specific application to the games of Tic-Tac-Toe and Checkers. The game tree as core data structure was presented, along with a wide range of enhancements to improve its performance – spanning more than 40 years of research efforts. Different evaluation methods were covered, including a neural network-based approach that will be experimentally examined in this study. A discussion followed on knowledge representation and the concepts of opening-books, extended-books and end-game databases. Various game learning techniques were examined, with self-play (coevolution) identified as the main training method for experimental testing. A small section was dedicated to the human and computer world champions, ending with a look at games with large search spaces as the future for games research.

CHAPTER 2

Background on Computational Intelligence Techniques

"I not only use all the brains that I have, but all that I can borrow".
- Woodrow Wilson (1856 - 1924)

The core computational intelligence paradigms that are used in and/or influenced the experimental work for this thesis, are presented in this chapter. Typical neural network architectures and learning approaches are covered. The core evolutionary computation constructs based on Darwinian evolution are presented, followed by an in-depth discussion of particle swarm optimisation and the influence of its various parameter settings. Finally, coevolution is discussed as a competitive training strategy.

2.1 Introduction

The previous chapter provided an insight into the game-specific techniques that are implemented in a typical game simulator. In order to add a more intelligent approach to board state evaluations and player training in general, a series of computational intelligence techniques are required. This chapter aims to provide an overview of these relevant techniques.

Basic neural network theory is presented in section 2.2, covering aspects such as architecture, weight initialisation, different training approaches and application areas. Evolutionary computation is introduced in section 2.3, which lays the groundwork for a more detailed discussion on particle swarm optimisation (PSO), its parameters and different information sharing structures in section 2.4. Coevolution is covered in section 2.5 as a competitive training strategy.

2.2 Neural networks

The human brain can easily be described as the most advanced information processor in the world today. It handles 'complex' tasks such as visual scene analysis, intricate acoustic positioning as well as speech recognition seemingly instantaneously. Translating these capabilities to modern day computer systems have been marginally successful, but in comparison have not yet reached the human brain's incredible efficiency.

The brain is predominantly constructed out of countless interconnected neurons that each individually 'fire' (or activate) a signal along a network of synapses as soon as a particular internal chemical threshold is exceeded. Repeated 'firing' strengthens certain connections, while a lack of 'firing' results in weakened connections. In a very simplified and abstract sense, this process can be regarded as 'human learning'. Artificial neural networks (ANN) aim to mimic the structural characteristics of actual neurons and synapses – albeit on a significantly smaller scale – in order to construct a framework for 'learning' certain features from a given set of inputs.

The remainder of this section discusses a selection of the relevant ANN themes pertaining to the experiments and comparisons conducted throughout this study. The specialised field of neural networks is extremely vast, and the interested reader is referred to [99] or [15] for a broader investigation of this interesting topic.

2.2.1 Structure

In its traditional form, the ANN has three layers of interconnected nodes (or neurons). Figure 2.1 shows an abstract graphical representation of such a network. The first layer is called the *input layer*. The input layer's size (number of neurons) is greater than or equal to the number of features constituting each input pattern. The input layer may sometimes be augmented with an additional constant neuron – referred to as the bias unit – with a constant input value of –1 [46].

The second layer is referred to as the *hidden layer*. The hidden layer is also constructed out of neurons. Each input neuron is connected to every hidden layer neuron through a set of weighted connections. Each hidden layer neuron makes use of an activation function. The activation function is employed to calculate the 'firing strength' (output) of each neuron, given the weighted sum of all connected input neurons [101]. The hidden layer may also be augmented

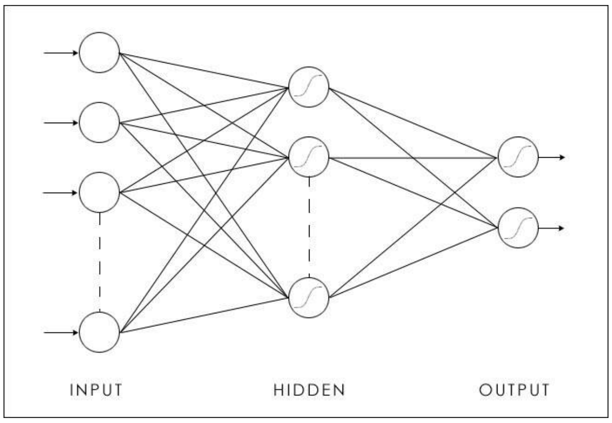


Figure 2.1: Abstract representation of a simple ANN structure.

with an additional constant neuron with the value of -1. In simple ANN architectures only a single hidden layer is constructed, but multiple hidden layers of varying sizes may also be used. Some of these well-known multi-layer architectures include the Elman recurrent neural network and the time-based neural network [46].

An ANN's last layer is called the *output layer*. Once again, the hidden layer is connected to the output layer through a set of weighted connections. Each of the neurons in the output layer also makes use of an activation function. The output from the network is used to make a decision regarding the training of the network, or in the case of an already trained network provides the correct response to the given set of inputs.

Weight initialisation

Before training, an initial state has to be constructed for the neural network. Different methods of initialising the weights of an ANN exist. The use of random numbers in the range (-0.5, 0.5) is often cited, but it remains largely dependent on the specific activation function in use. Wessels and Barnard [146] derived range–equations for weight initialisation that proved to facilitate learning with the most success. The range is defined as:

$$\left(\frac{-1}{\sqrt{fanin}}, \frac{+1}{\sqrt{fanin}}\right) \tag{2.1}$$

where the term *fanin* refers to the number of incoming connection weights to the specific neuron. The weight initialisation range specified by equation 2.1 is used for the experimental work in this study.

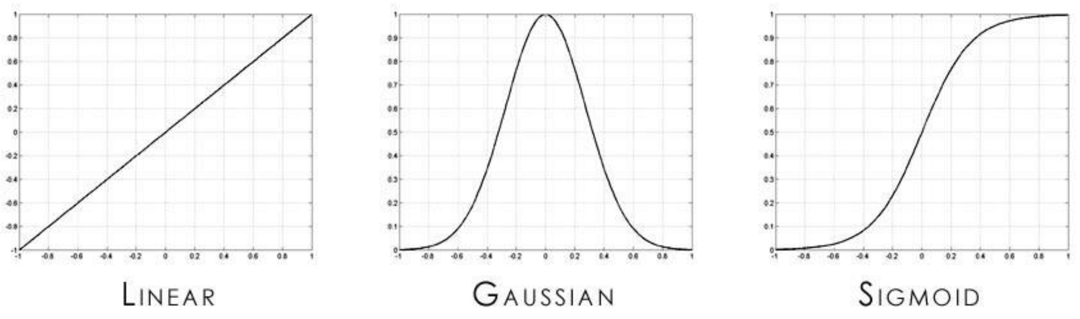


Figure 2.2: Various activation function plots used by ANN nodes.

Activation functions

As mentioned in section 2.2.1, the different neurons in the hidden and output layers each make use of an activation function to calculate the output from a given weighted sum of inputs. Figure 2.2 graphically depicts plots of various popular activation functions. Previous work in game learning by Chellapilla and Fogel [26] [27] [53] showed the successful use of the hyperbolic tangent as an activation function for evaluating Checkers board states. Tesauro [130] used a sigmoidal function for his temporal difference approach to learning Backgammon. The sigmoid activation function is used in all of this studies' experimentally constructed ANNs.

2.2.2 Training methods

The act of training an ANN boils down to the process of updating the weighted connections between neurons in the different layers of the network. In order to train, a series of input data is fed into the network and the resulting output compared to an expected output. This process is also known as *supervised learning*, since it uses pre-built data sets that accurately describe the input/output relationships. A popular application of ANNs that makes use of supervised learning, is that of the classifier system.

The main reason for using an ANN as a classifier system lies in the network's ability to detect and approximate underlying correlations in the provided input data. By training an ANN on a representative set of input data, it should afterwards be able to correctly classify a *new* pattern that has not been seen by the network before. This ability to classify previously unseen patterns correctly is known as *generalisation*, which also characterises a well-trained network.

It is possible to over-specialise a network by choosing an overly large architecture, using a non-representative set of training data containing large amounts of noise, or continuing to train the network well after optimal generalisation has been achieved. This limits the ANN's ability to generalise on the data, and is known as *overfitting*. It is the aim of the network designer to optimise an ANN's ability to generalise and avoid overfitting [45].

It is also possible to train networks using an unsupervised approach. Unsupervised learning is suited to situations where it is difficult or impossible to calculate an accurate error measure between the given inputs, and an expected output. In most cases, there is no expected output to compare against. The self-organising map [82] is a good example of a neural network-based approach that relies on unsupervised learning to cluster data.

It is possible to train populations of neural networks through competitive coevolution [53]. By representing the weights of the neural network as an individual in the population's genetic structure, it is possible to adapt the structure (train the network) to more closely resemble individuals with higher fitness. Fitness is usually calculated through some competitive tournament scheme. Training neural networks through coevolution is discussed in greater detail in chapter 3, but is mentioned here for completeness.

Feed forward networks

In order to start any training process there needs to be input data. Providing the input data to the network (also referred to as 'feeding' the network) constitutes assigning the value of each of the input pattern's features to a specific input neuron. These input values are then 'fed forward' through the network. The nodes residing in the hidden layer each calculates a weighted sum of the inputs and corresponding network weights. The node's activation function (see figure 2.2 for example plots of activation functions) receives the weighted sum as input and produces a numeric value that is in turn fed through to the output layer. After calculating the weighted sum for the output nodes, and discovering the final output value after activation function processing has completed, the network is able to produce a set of outputs. It is the job of the network designer to analyse the output values (possibly comparing them with expected output values) in order to start updating the network weights – thereby allowing the network to start 'learning'.

Gradient descent learning

The feed-forward phase produces an output based on the given set of inputs. In supervised learning the network output can be compared against a correct expected output, and the mean squared error between the outputs computed. This error represents a position on the search function landscape that needs to be minimised. As the error declines, the network is said to be descending the gradient of the landscape until it converges on a local minimum.

Werbos [145] introduced a set of weight adjusting equations based on the result of the error function that will allow the network to perform gradient descent learning, and called it back-propagation. An in-depth exploration of gradient-based learning is beyond the scope of this study, but the interested reader is referred to a book by Bishop [15] that has optimised examples of training based on the gradient descent paradigm, namely conjugate gradient descent, scaled conjugate gradient descent and Levenberg-Marquardt methods.

Since no error can be computed during training for the provided inputs to the neural network in the game learning environment of this study, gradient descent learning is not applied. It is mentioned in this chapter for completeness only.

2.2.3 Application areas

Neural networks have been extensively applied to a wide range of specialised areas. As previously mentioned in chapter 1 on game learning, neural networks have been trained as board-state evaluators (see section 1.4.3), and a learning method based on temporal difference has used neural networks to successfully train intelligent Backgammon players (see section 1.5.2 on TD-learning).

One of the other well-known application areas of the neural network is its use as a highly developed classification tool. Examples of typical data sets that are classified by neural networks can be found at the UCI Machine Learning repository [19]. In addition, neural networks can be trained to perform time forecasting on historical financial data, to analyse and recognise objects in images (image analysis), perform real-time classification of biometric data and aid in medical diagnosis, to name but a few [46].

2.3 Evolutionary Computation

Evolutionary computation is a broad term that refers to a population-based [12] stochastic search and optimisation method, derived from the principles of ‘survival of the fittest’ as defined by Darwinian evolution [39]. The algorithmic processes involved in standard evolutionary computation was defined by Holland [67], and a thorough treatment of this very broad field can be found in [60].

The biological cycle of life follows the familiar pattern of birth, reproduction and death over time. The rate at which these events occur depends on a large number of factors, including the living environment, the size of the existing population, the overall fitness of the members of the population and the distinctive diversity of genetic material available in the population.

These same concepts can be directly mapped to the algorithmic approach to evolution. An evolutionary computing simulation is performed over a finite period of time, with each time step referred to as an *epoch* and the current population also called the *current generation.* The evolutionary environment is represented by an optimisation problem, with the population of individuals representing possible solutions to the optimisation problem. Individuals that represent more optimal solutions are said to be more fit than their neighbours. A number of evolutionary operators are applied to the population at the end of each epoch. The aim of the operators correspond to the aforementioned Darwinian principle of 'survival of the fittest'. A percentage of the fittest solutions in the population are selected to survive to the next generation – a process referred to as *elitism.* The surviving individuals are involved in replacing the void formed by the culled individuals through *reproduction.* A slight probability usually exists that mutation will occur during reproduction, in order to maintain diversity and counteract premature convergence. The fitness of the individuals are recomputed and the cycle restarted to construct the next generation.

The evolutionary process is usually continued for a predetermined length of time, or terminated as soon as the population converges on a suitable solution. The manner in which a potential solution is individually represented is discussed in section 2.3.1, followed by a more detailed exposition of various evolutionary operators in section 2.3.2. An overview of the broad range of existing evolutionary computation models is provided in section 2.3.3, after which a short list of the applications of evolutionary computation is given in section 2.3.4 to conclude this part of the chapter.

It should be noted that the pure form of evolutionary computation as it generally applies to problem solving is not applied to the experimental work in this study. Instead, a more specialised subfield – called particle swarm optimisation – is used to evolve potential solutions, driven by a competitive coevolutionary scheme. Both particle swarm optimisation and coevolution is discussed in more detail later on in this chapter. This section on evolutionary computation only serves to introduce the basis of inspiration for these methods.

2.3.1 Data representation

As already mentioned, each individual in the population represents a candidate solution to the optimisation problem. Two main distinctions can be made when discussing the solution representation. A *genotype* encodes the genetic information required to obtain the fitness of the problem (the parameters of a mathematical function for example). The genotype is usually comprised of real-valued numbers, or in the case of genetic algorithms may even correspond to a binary bit string. A fixed length bit string is usually applied, but a variable length string has also had some success [61]. The second distinction refers to the phenotypic representation of the solution. The phenotypic representation aims to model the behaviour of the individual, rather than the problem composition.

An intricate relationship can exist between the two representations [95]. The first is called *pleiotropy*, and occurs when random genetic mutations can cause unexpected variations in the phenotypic behaviour of an individual. The second relationship is termed *polygeny*, and occurs when a specific set of genes contribute to a certain phenotypic trait. Removing the trait from the individual requires removing or replacing the associated genes.

2.3.2 Evolutionary operators

The evolutionary process updates the data values presently represented by the individuals in the population, through the use of various operators inspired by Darwinian evolution. These operators are briefly discussed below.

Selection

Selection in the evolutionary computation context can have a two-fold meaning. The first involves the selection of the particular individuals that survive and form part of the next generation. The second involves selecting individuals as parents for reproduction in order to construct the new individuals for the next generation. The details of reproduction in an evolutionary context is discussed below.

The first commonly used selection scheme is called *random selection*. Random selection involves choosing individuals in the population with no regard to their current fitness, but instead relies on a pure stochastic method of selection. Another popular selection scheme is *tournament selection*, where small groups of individuals are randomly selected and the overall best performing individual determined through independent tournaments between the small groups.

Research by Ficici *et al.* [50] indicated that normal selection schemes might not work as well in coevolutionary environments as was hoped. Further research into selection schemes for coevolutionary environments need to be conducted, and a series of new selection schemes inspired by Formula One Grand Prix are presented in chapter 6.

Reproduction

Reproduction in an evolutionary computing context is largely dependent on the specific evolutionary model employed. In the majority of cases, reproduction requires the selection of two existing individuals as parents before spawning offspring. The exceptions to this rule are the evolutionary programming model that produces offspring through mutation, and differential evolution that allows for more than two parents to contribute to the reproduction process.

Assuming a bit-level data representation, genetic algorithms allow for operations on bit-level to create offspring by identifying one or more pivot points and performing the appropriate cross-over operations. In genetic programming, reproduction involves swapping subtrees between the individuals selected as parents [101].

Reproduction allows for the genetic material of the more fit individuals in the population to propagate through multiple generations, hopefully combining with other fit individuals to evolve into a globally optimal solution.

Mutation

It is possible for a population of individuals to stagnate on locally optimal solutions, with all the individuals sharing a large portion of identical genetic material. Mutation aims to add an element of noise to the search process, hopefully guiding the individuals to more optimal solutions without causing premature convergence. This is also referred to as the need to maintain *diversity* within a population.

The application of mutation is also dependent on the particular model of evolution employed. In GAs it may involve randomly changing bit values to their compliment representation, or randomly selecting and deleting subtrees in the case of genetic programming.

The rate of mutation is also an important factor to consider while evolving a population of individuals. A low mutation rate may be insufficient to combat premature convergence, while a too large mutation rate may not result in locating optimal solutions that require fine adjustment. It is accepted that a declining mutation rate provides the best results, with the initially large mutation rate supporting global exploration of the search space, ultimately specialising on a particularly promising area of the search space as the rate declines [46].

2.3.3 Models of evolution

A wide variety of evolutionary models exist to implement the aforementioned evolutionary constructs. A selection of the more popular approaches are discussed below. The interested reader is referred to [106] for more information on differential evolution and [46] for a more detailed algorithmic treatment of the following models.

Genetic algorithms (GA)

The genetic algorithm has been a very popular model of evolutionary computation. Originally described by Fraser [58] and formalised by Holland [67], genetic algorithms represent individuals as *chromosomes* – usually some bit string representation in genotypic space. The traditional evolutionary operators are applied to the bit string representation, and evolution continues until a suitable solution has been found. A more detailed explanation of the specific application of the evolutionary operators to genetic algorithms can be found in [46] and [119].

Genetic programming (GP)

Genetic programming was invented by Koza [85] and makes use of a tree representation for each individual. An individual's fitness corresponds to traversing the tree, executing the various operations described therein and recording the output. Reproduction between individuals involves randomly swapping subtrees to form new individuals. Mutation may involve randomly changing an operator or value defined in the tree, deleting certain portions of the tree, or extending the tree in a random fashion by adding more child nodes [46] [119].

Evolutionary strategies (ES)

Evolutionary strategies model the evolution of evolution with the aim of optimising the processes involved [109] [46]. Originally defined by Rechenberg [108] and Schwefel [120], evolutionary strategies evolve both the genetic and phenotypic representations of individuals. A difference to the traditional evolutionary algorithm includes the application of mutation only if it results in a more fit individual [46] [119].

Evolutionary programming (EP)

Evolutionary programming was invented by Fogel [54] and focuses on the phenotypic behaviour of an individual. It does not make use of the normal reproduction operators as defined by evolutionary algorithms, but instead produces offspring through mutation only. In addition it also makes use of the elitism operator as described in [46] and [119]. Evolutionary programming was used to successfully train the Checkers program Anaconda (Blondie24) by Chellapilla and Fogel [53] [27] [26].

Cultural evolution

Cultural evolution [68] aims to positively bias the search process by introducing domain-specific information in the form of a belief space. In society, cultural evolutionary processes progress much quicker than biological evolutionary processes, as indicated by the rapid adoption of fashion and other lifestyle trends. The simulated cultural evolutionary process involves evolving both a belief space, and the traditional population space – hopefully resulting in faster discovery of an optimum solution to the optimisation problem. The interested reader is referred to [46] for a more complete algorithmic discussion on the topic.

Coevolution

Coevolution [42] does not rely on an explicit fitness function to determine the fitness of the individuals. Instead, individuals are evaluated in relation to the other individuals in the population, or against the individuals in other populations. Competitive coevolution induces an arms race that results in overall improved performers. Coevolution is discussed in greater detail in section 2.5.

2.3.4 Application areas

Evolutionary algorithms have been applied to a very broad range of fields, from scheduling and control applications to classification and data mining operations [46]. In the game learning context, evolutionary programming has been successfully applied by Chellapilla and Fogel to evolve intelligent game playing agents [53] [27] [26], and GA-based niching algorithms have been used to construct various playing strategies in Checkers [79].

2.4 Particle Swarm Optimisation

In recent years a novel new population-based optimisation method was introduced, inspired by the flocking behaviour of birds. The Particle Swarm Optimisation (PSO) algorithm was first described by Kennedy and Eberhart [76] in 1995, and have since proven to be more successful than traditional evolutionary computing (EC) approaches in complex problem solving [46] [43]. The PSO technique borrows some basic elements from any EC approach, with the traditional terms such as the *population* being referred to as the *swarm* and the *individuals* referred to as *particles*. The operators traditionally associated with evolutionary computation, such as selection, reproduction and mutation are not applicable to the standard PSO algorithm. Recent extensions to the standard PSO algorithm by various researchers have however added both selection [4] and reproduction [89], with notable success. The particle repelling techniques described in section 2.4.5 can be seen as variants of the mutation operator, due to their similar aim: to increase diversity.

In particle swarm optimisation each particle is represented as an n-dimensional vector $\vec{x}$, and signifies a potential solution to the optimisation problem. An element unique to the PSO technique is the addition of a velocity for each particle. The velocity allows the particle to 'fly' through the n-dimensional search space, which constitutes the evolution of the potential solution.

Each particle maintains a copy of its best position found during the simulation thus far. The particles also share information about possible better solutions found by other particles in the swarm. Neighbourhood information sharing structures are discussed in more detail below, but in short allows a particle to compare its performance against as little as two neighbouring particles, or against the complete swarm.

The particle is proportionally steered in the direction of the neighbourhood best and previously found personal best positions, through the use of cognitive (c_1) and social (c_2) acceleration constants. Depending on the specific optimisation problem at hand, a restriction may be placed on the domain that the particle is allowed to explore in each dimension. Additionally, a restriction may be placed on the maximum velocity of a particle to counteract any velocity explosions and subsequent particle scattering into unimportant areas of the search space. Lastly, a particle may be subject to maintaining only a portion of its previous velocity – a concept referred to as inertia. The inertia term adjusts the level of local or global exploration of each particle in the swarm.

Here follows a more concise mathematical definition of the steps involved in the standard particle swarm optimisation algorithm:

1. Instantiate a swarm Φ of particles to random positions in an n-dimensional hyperspace Ω, where each particle $\vec{x}_i$ is a vector representing a potential solution to the optimisation problem.

2. Set initial personal best positions, $\vec{y}_i$ equal to original particle positions $\vec{x}_i$.

3. Set initial velocities, $\vec{v}_i$, to 0.0.

4. Repeat until converged:

   (a) Determine each particle's fitness, f, using a problem-dependent method.

   (b) Compare current fitness against previous best fitness, updating the personal best position $\vec{y}_i$ at time step t as follows:

$$\vec{y}_i(t+1) = \begin{cases} \vec{y}_i(t) & \text{if } f(\vec{x}_i(t+1)) \geq f(\vec{y}_i(t)) \\ \vec{x}_i(t+1) & \text{if } f(\vec{x}_i(t+1)) < f(\vec{y}_i(t)) \end{cases} \tag{2.2}$$

   (c) Determine the neighbourhood's best particle position $\vec{z}$ (different neighbourhood structures are described in more detail below).

   (d) Update the velocity for each particle, taking into account the personal $\vec{y}_i$ and neighbourhood best $\vec{z}$ positions. The velocity $v_{ij}(t)$ for the j-th dimension of particle $\vec{x}_i$ at time step t is updated as follows:

$$v_{ij}(t+1) = v_{ij}(t) + p_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + p_{2j}(t)(z_j(t) - x_{ij}(t)) \tag{2.3}$$

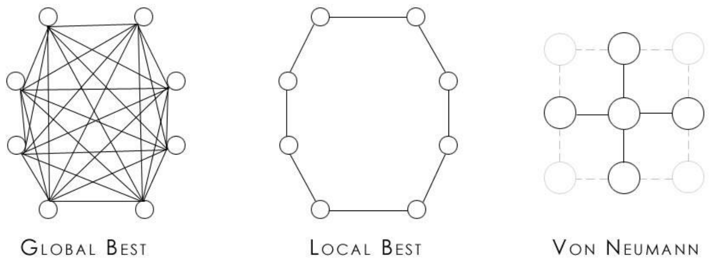


Figure 2.3: Various PSO neighbourhood information sharing structures.

where $p_{1j} = r_1 \times c_1$ and $p_{2j} = r_2 \times c_2$. As already mentioned, c_1 and c_2 are the cognitive and social acceleration constants respectively, and r_1 and r_2 are random numbers such that $r_1, r_2 \sim U(0,1)$.

(e) Update the position for each particle as follows:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \tag{2.4}$$

2.4.1 Information sharing structures

The previous section mentioned the purpose of the neighbourhood information sharing structures, namely to allow particles to propagate knowledge of superior solutions found by other particles in their particular neighbourhood. The neighbourhood best particle, $\vec{z}$, also influences the velocity update process as defined by equation 2.3. Two traditional PSO neighbourhood structures are examined in this study, namely Global Best (GBest) and Local Best (LBest). In addition, the recently developed Von Neumann structure [78] is examined and for the first time applied to the game learning domain. Each structure is defined in more detail below, with a graphical representation given in figure 2.3.

It should be noted that the neighbourhoods discussed below are constructed in variable space, and not fitness space. Various other neighbourhood structures are briefly mentioned at the end of this section.

Global Best (GBest)

The GBest neighbourhood information sharing structure was introduced in the original PSO paper by Kennedy and Eberhart [76]. It is graphically depicted in the first sub-image in figure 2.3, and can simply be described as a completely interconnected network of nodes, where each particle shares information with all the other particles in the swarm. It is the easiest information sharing structure to implement. The neighbourhood encompasses the complete swarm, with the Global Best particle representing the overall best performer in the swarm.

GBest suffers from premature convergence on suboptimal solutions that are not even guaranteed to be local optima [137]. An improvement on the basic GBest information sharing structure is presented by the Guaranteed Convergence PSO (GCPSO) algorithm, described in section 2.4.3 and originally introduced by Van den Bergh [137] [140].

Local Best (LBest)

The LBest neighbourhood information sharing structure makes clearer use of smaller predefined neighbourhoods within the context of the larger swarm. It is graphically depicted in the centre image of figure 2.3, and conceptually represents a one-dimensional lattice structure with a sliding window that is able to move freely around the lattice. The size of the window represents the neighbourhood size. As an example, an inclusive neighbourhood size of 3 particles will allow a specific particle to compare its performance (fitness) against its immediate left and right-sided neighbours. As already mentioned, a neighbourhood size equal to the complete swarm size corresponds to the GBest neighbourhood structure, allowing the LBest structure to operate as the GBest structure if the need exists.

Experimental analysis by a number of researchers have shown that LBest does take slightly longer to converge on a particular solution, but ultimately result in more optimal solutions when compared to GBest [124]. As will be seen by the experimental work in this study, the LBest neighbourhood structure consistently outperforms GBest to evolve more intelligent players in Tic-Tac-Toe [56], Checkers [55] and the Iterated Prisoner's Dilemma [57].

Von Neumann

The Von Neumann neighbourhood information sharing structure is the last image depicted in figure 2.3, and extends the one-dimensional lattice structure of the LBest neighbourhood to form a two-dimensional lattice. With this extension, each particle is able to compare its performance against its immediate left and right-sided neighbours, as well as the neighbours immediately above and below it in variable space.

The Von Neumann structure was originally introduced by Kennedy and Mendes [78], and on initial experimental work by the same authors showed to outperform the traditional PSO neighbourhood structures. This study will examine the first application of the Von Neumann neighbourhood structure to game learning, and show some definite increased performance characteristics under certain experimental conditions [56] [55] [57].

Other neighbourhood structures

It should be noted that a wide variety of other neighbourhood structures have recently been introduced by researchers in the field. The decision to use the standard GBest and LBest PSO neighbourhood structures was made in order to ease possible future comparative study. The Von Neumann network structure was chosen due to the suggested performance gain mentioned by its authors, and still relatively unexplored nature. For completeness, a selection of other neighbourhood structures that were not examined in this study are briefly mentioned below.

Hu *et al.* [70] introduced a dynamic neighbourhood structure that finds the closest neighbours in fitness space (not variable space). Kennedy [75] conducted interesting research on social network structures based on the 'small world' phenomenon [98], and introduced the modified wheel and star topologies that included some 'small-world randomisation' in connections. Suganthan [128] used a dynamically changing neighbourhood size – incorporating the calculated Euclidean particle distances – starting with only single particles (LBest neighbourhood size of 1) and gradually increasing the neighbourhood size over time to finally encompass the whole swarm (representing GBest). Mendes *et al.*'s work in [96] also resulted in the creation of the Pyramid, 4Clusters and Square architectures. More 'obscure' randomly created architectures are discussed in [78]. All of the above authors presented an increase in performance when using their respective neighbourhood structures, and a broader examination of all the different neighbourhood structures' performance in a game learning environment is left as future work resulting from this research.

2.4.2 Parameters: restrictions and influence

A large part of a successful PSO implementation lies in understanding the interaction of the various parameters present in the underlying PSO algorithm. In most cases the parameters settings are problem specific. For example, sometimes there are certain restrictions on the ranges of the problem domain that need to be enforced, and sometimes those restrictions are not applicable. The following subsections will examine a variety of parameters, their influence on swarm behaviour and the restrictions that might be applicable in certain instances. Most of the parameter settings are directly involved in either guaranteeing convergence, or resulting in accelerated convergent behaviour.

Acceleration constants

As was previously mentioned in the discussion on the standard PSO algorithm, the velocity update equation involves a comparison of both the difference between the current and previous best performance values, as well as the current en neighbourhood best performance values. The particle's velocity is proportionally updated to take into account both the differences, which are in turn independently managed by the cognitive (c_1) and social (c_2) acceleration constants.

A comparatively larger c_1 value will steer the particle in the general direction of its previous best solution, while a larger c_2 value will steer it towards the neighbourhood best solution. Some restrictions on the c_1 and c_2 values have been defined by Kennedy [74], based on the convergent behaviour of a swarm. The restriction forces particles to not exhibit escalating oscillatory behaviour, and is formally stated as:

$$c_1 + c_2 \leq 4 \tag{2.5}$$

The cognitive and social acceleration constants used for the experimental work conducted in this study adhered to equation 2.5 to avoid oscillatory behaviour.

Inertia weight

The first extension to the standard PSO algorithm to induce accelerated convergence was introduced by Shi *et al.*[122] in 1998, namely inertia. The inertia weight, ϕ, has a significant influence on convergence, and is added to the velocity update originally listed in equation 2.3 as follows:

$$v_{ij}(t+1) = \phi v_{ij}(t) + p_{1j}(y_{ij}(t) - x_{ij}(t)) + p_{2j}(z_j(t) - x_{ij}(t)) \tag{2.6}$$

The inertia weight adjusts the retained size of the previous velocity for the current time step. It primarily balances out the local and global search capabilities of the swarm. A large inertia weight value facilitates global search, while a smaller inertia weight facilitates local search. Shi *et al.* [123] extended the static inertia weight to form a linearly decreasing inertia over time, and reported a suitable improvement over previous results. The linearly decreasing inertia weight causes a global search at the start of the simulation, and ends with a local search.

It should be noted that the linearly decreasing inertia weight is sometimes not sufficient in dynamic environments, and it is due to this fact that only a static inertia weight is applied in the experimental work in this study. In order to address the occasional problem-associated inefficiency of the inertia weight, Shi *et al.* [125] developed a fuzzy system to adapt the inertia weight according to a set of fuzzy rules, resulting in yet another slight performance gain.

Finally, Van den Bergh [137] empirically examined the interaction of the various PSO parameters, with specific reference to convergent behaviour. Experimental analysis indicated that not all values for c_1, c_2 and ϕ resulted in the swarm converging on a specific solution. Van den Bergh derived an equation that represents the limits to parameter values that will result in convergence on a local solution. The equation is formally represented as:

$$\phi > \frac{(c_1 + c_2)}{2} - 1 \tag{2.7}$$

Inertia weight values used in the experiments conducted in this study adhered to the aforementioned equation for convergence.

Maximum velocity

In some optimisation problems and parameter combinations the particle velocities may explode into very large values for all dimensions, resulting in inferior performance and non-convergent swarm behaviour. A number of solutions exist to restrict the maximum velocity. The first is simply to clamp the velocity in each dimension to [-*VMax*, +*Vmax*], where *VMax* represents the maximum velocity. If a problem has a restricted domain space, the maximum velocity is usually chosen as a proportionally smaller value. The second approach to enforcing the maximum velocity recognises the fact that strictly clamping the velocity vector in all dimensions may cause a change in direction for the particle. In order to maintain the original direction of the velocity vector, it can be proportionally scaled in all dimensions instead.

The last approach to addressing the potential explosive velocity of a particle was introduced by Clerc *et al.* [31] [32], namely the use of a constriction coefficient in the velocity update equation. The velocity update method originally listed in equation 2.3 is changed accordingly to:

$$v_{ij}(t+1) = \kappa(v_{ij}(t) + p_{1j}(y_{ij}(t) - x_{ij}(t)) + p_{2j}(z_j(t) - x_{ij}(t))) \tag{2.8}$$

where

$$\kappa = 1 - \frac{1}{p} + \frac{\sqrt{|p^2 - 4p|}}{2} \tag{2.9}$$

with $p = p_1 + p_2 > 4$. In the above equations, κ is referred to as the constriction coefficient, and its use should alleviate the necessity for any implicit maximum velocity clamping or scaling in all dimensions. Shi *et al.* compared the constriction coefficient approach for managing velocity to using an inertia weight in [44], and concluded that the use of the constriction approach is preferred – unless the values for c_1, c_2 and ϕ are chosen to not show divergent behaviour. This conclusion corresponds to equation 2.7 by Van den Bergh that eliminates the use of a maximum velocity while still resulting in a converging swarm in most problem environments.

The approach of strictly clamping the maximum velocity is experimentally compared to the linearly scaled and constriction coefficient approaches in chapter 6.

2.4.3 GCPSO

The first specific improvement on the standard PSO algorithm that is applied to experimental work in this study, is the Guaranteed Convergence PSO (GCPSO), introduced by Van den Bergh [137] [140]. GCPSO improves on the poor performance of the GBest structure, by recognising that in some instances, the following relation holds for the neighbourhood (global) best particle:

$$\vec{x} = \vec{y} = \vec{z} \tag{2.10}$$

where, $\vec{x}$ represents the current position, $\vec{y}$ represents the previous best position, and $\vec{z}$ represents the neighbourhood (global) best position. The impact of this relation on the update of the particle velocity is that the last two terms of the equation become zero, and the particle is ultimately only propelled by its inertia. The GCPSO algorithm requires that if the relation in equation 2.10 holds, the position of the neighbourhood (global) best particle at index α be updated using the new equation:

$$\vec{x}_\alpha(t+1) = \vec{z}(t) + \phi\vec{v}_\alpha(t) + \delta(t)(1 - 2r_2(t)) \tag{2.11}$$

The inclusion of the δ term results in the neighbourhood best particle to perform a random search in the close proximity of its current position, hopefully leading to a more optimal solution and restricting premature convergence. In the original work by Van den Bergh [137] [140], the size of the search term was dynamically adjusted during the simulation based on the rate of convergence.

The GCPSO optimisation is applied to the problem of evolving Checkers agents later in this study. The appropriate search term size is experimentally determined in section 6.2.9, along with a performance comparison of extending the GCPSO technique to other neighbourhood structures such as LBest and Von Neumann.

2.4.4 Binary PSO

The second specific variant of the standard PSO algorithm applied for experimental work in this study, is the Binary PSO algorithm (BinPSO). BinPSO was introduced by Kennedy and Eberhart [77] in 1997, and allowed for particles to operate in a discretized search space – in contrast with the continuous search space of the standard PSO algorithm. The BinPSO algorithm can still make use of the traditional neighbourhood information sharing structures, since the only change to the standard PSO algorithm involves the position update equation. The equation is changed as follows:

$$x_{ij}(t+1) = \begin{cases} 0 & \text{if } r_i(t) \geq f(v_{ij}(t)) \\ 1 & \text{if } r_i(t) < f(v_{ij}(t)) \end{cases} \tag{2.12}$$

where

$$f(v_{ij}(t)) = \frac{1}{1+e^{-v_{ij}(t)}}$$

and $x_{ij}(t)$ is the value of the j-th parameter of particle P_i at time step t, $v_{ij}(t)$ is the corresponding velocity and $r_i(t) \sim U(0,1)$. The traditional PSO velocity update equations remain the same [76]. However, the original authors of the BinPSO algorithm recommend clamping the maximum velocity in the range [-4,4] to avoid saturation of the sigmoid function [43].

The BinPSO algorithm is applied in this study to evolve strategies for the Iterated Prisoner's Dilemma in chapter 8 – the first known application of its kind in the field of evolutionary game learning. More specific details on the IPD strategy generation process with BinPSO are given in section 8.5.2.

2.4.5 Particle repelling

When applying PSO to multi-modal problems, the swarm encounters the problem of premature convergence on suboptimal solutions and it struggles to keep track of changing optima. Various researchers have introduced so-called repelling techniques to combat this side-effect of the traditional PSO algorithm. A selection of the techniques are described below.

Riget and Vesterstrøm [142] introduce a modification to the standard PSO algorithm that includes attractive and repulsive components, abbreviated as ARPSO (*Attractive* and *Repulsive* PSO). The introduction of these components allow for a way to control the diversity of the swarm. As soon as the diversity drops below a certain threshold, the particles' velocities are reversed (the repulsing behaviour), causing an explosion of the particles into search space. As soon as diversity is regained, the velocities are returned to normal (the naturally attractive behaviour as defined by the standard PSO algorithm). The authors indicate a definite increase in performance on a series of standard and highly dimensional mathematical functions.

Blackwell and Bentley [17] introduce similar attractive and repulsive components for the creation of musical melodies. The uniqueness of the melody is subject to the diversity of the swarm, where a converged swarm will result in a repetitive melody, which should preferably be avoided. The authors introduce collision-avoidance operators based on work done by Reynolds [111] on the simulation of animal flocking behaviour in a computer graphics context. The operators are applied to the particles in order to combat convergence, where particles in close proximity repel each other. The overall swarm is attracted to a global moving target provided by an external source, such as another musical improviser. The attraction and collision-avoidance scheme allows for close matching of a fast-changing global target, without premature convergence.

Blackwell extends the aforementioned particle collision-avoidance research by introducing a charged PSO (CPSO) algorithm, based on polarised particles that are typically found in electrostatics [16]. Blackwell considers the original PSO as a neutrally charged (uncharged) swarm, where all the particles are allowed to converge on a single global optimum. The aforementioned collision-avoidance scheme can be visualised as a swarm that is completely charged, and all the particles repel each other in a uniform manner. The CPSO technique combines these two approaches in a manner reminiscent of atomic behaviour, with the neutrally charged swarm forming the nucleus and the fully charged swarm consistently orbiting around the optimum. This approach combines the exploitation and exploration characteristics of the PSO algorithm, and is found to be successful in a variety of highly dynamic environments.

Silva *et al.* [126] follow a novel approach to particle repelling by classifying a swarm as either being a group of *predators*, or *prey*. A prey swarm's particles are equal to the standard PSO algorithm particles, and they are attracted to the global optimum. A swarm of predators are attracted to the best performing particles in the swarm of prey. The remainder of the particles in the prey swarm are repelled by the predators, which inherently results in continued diversity among the potential solutions and a wider exploration of the search space. The authors reported positive results from optimising standard benchmarking problems.

Finally, Løvberg and Krink [88] introduce a particle dispersement scheme based on the self-organised criticality (SOC) principle. Each particle maintains a critical rating, indicating its level of convergence with the best globally found optimum so far. As a particle approaches the global optimum, its critical level rating increases. Should a particle's rating exceed a predefined critical limit, the particle is repositioned in the search space through one of two methods. The first re-initialises the particle into the search space, allowing it to 'forget' its previous best position. The second relocation method only pushes the particle further in its current direction, according to a ratio with which it exceeded the critical limit. After relocation, the particle's critical rating is decreased, but its original neighbouring particles' critical limits are increased – thereby allowing for possible chain reactions (or so-called 'avalanches') of repositioning and increased diversity. The authors also link the SOC methodology to the inertia term for improved convergence, and report superior performance over the standard PSO algorithm on a selection of mathematical optimisation problems.

For experimental work in this study, a new particle dispersement operator is introduced that borrows some characteristics from the aforementioned techniques. The dispersement operator is explained in more detail in section 6.4.3, and is in its current form more suited for use in a coevolutionary training environment.

2.4.6 Application areas

Particle swarm optimisation has been extensively studied in the past nine years since its inception. A large portion of theoretical work has focused on analysing particle and swarm behaviour, including convergence and information dissemination through various information sharing structures. A selection of the theoretical PSO research applicable to this study has been mentioned in the previous sections of this chapter.

In addition to the theoretical research, PSO has been applied to train different variations of neural networks [136][72][96] – a topic covered in more detail in the next chapter. PSO has also been applied to mathematical function optimisation, multi-objective optimisation, constraint-based optimisation, training support vector machines, image analysis, data clustering and artificial immune systems to name a few. Research papers containing examples of the aforementioned applications and a wide selection of more detailed computational intelligence studies are available at [69].

2.5 Coevolution

The concept of coevolution was earlier mentioned as a specific model of evolution. Coevolution is formally defined in [42], and can easily be described at the hand of a competitive relationship in nature [46]. A plant has to adapt (evolve) in order to survive the continued attacks by native ant species. The plant excretes fluids to harden its defences, and the ant evolves stronger jaws to penetrate these defences. This tit-for-tat relationship continues indefinitely, resulting in an arms race of improved offensive and defensive strategies and subsequently stronger/better adversaries.

The above example illustrates *competitive coevolution* – the model employed to train game playing agents in this study. Another form of coevolution exists in which the two or more subpopulations in the environment cooperate in order to improve their fitness, commonly referred to as *symbiosis*. Symbiosis is not applied to the experimental work in this study, but the interested reader is referred to [137] for a summary of the main cooperative approaches to evolution.

2.5.1 Population dynamics

The coevolutionary process applies to one or more populations in the problem environment (or species in nature). Different models have been used to represent this interaction between the individuals and the different populations. The first model only relies on a single population of players that compete against themselves in order to determine the fittest individual, and is often referred to as the *global model* [24]. The various approaches to determining fitness is discussed in more detail in section 2.5.2.

The second model is referred to as the *island model* [63], and makes use of separate populations of individuals that evolve in isolation. This approach was invented to add a level of parallelism to the optimisation process, and the various populations may even exist on remote computers connected through some communication framework. Each isolated island's population evolves as described in the first model, through competition between native individuals. Individuals are additionally allowed to migrate to neighbouring islands, and by doing so increase the level of diversity due to the injection of foreign genetic material.

The third model divides a large population into overlapping subpopulations of individuals, and is referred to as the *neighbourhood model* [92]. It closely resembles the island model, but instead operates on a single machine and allows individuals to belong to more than one (sub)population. In some instances the neighbourhood model may show increased convergence speeds over the island model.

The global model is applied to experimental work conducted in this study.

2.5.2 Credit assignment

The main benefit of using a coevolutionary approach over traditional evolutionary techniques is the absence of a predetermined fitness function. Instead, the fitness of an individual is measured in relation to the other individuals in the population. This allows for a 'moving target' to be established in the fitness of the population, as evolution enhances certain individuals' performance and the remainder of the population tries to overcome the dominance of an ever-changing 'best' individual.

One obvious drawback of not having a standard universal fitness function is the occurrence of so-called 'one-shot wonders' in the population, also referred to as the 'Buster Douglas effect' by Blair and Pollack [18] in their paper on the success of a coevolutionary-trained Backgammon player. Buster Douglas was the world heavyweight boxing champion for nine months in 1990. It may be possible for an individual to outperform the majority of the population in the last generation due to a chance mutation. This may cause an individual to be selected as the 'overall best' individual, even though its playing strategy may only be exploiting the general population's genetic deficiencies. Usually, more thorough analysis through benchmarking or exposure to other playing environments will illustrate the strategy's lack of robustness against a wider selection of strategies.

In game learning it is equally difficult to identify and reward an exact sequence of decisions or moves that lead to a superior strategic position. Sometimes a 'mistake' made in the opening exchange of moves may lead to disaster 40 moves later in the game. Credit is instead usually assigned based on the outcome of a game, and not the sequences of moves themselves. It is the accumulated credit in a specific generation that determines the performance of the player, and may result in the aforementioned problematic situation of a 'one-shot wonder'.

Rosin and Belew [112] [113] have derived a selection of improved credit assignment techniques in order to more accurately determine an individual's overall performance. The following sections will look at the use of relative fitness evaluation, fitness sampling and the use of a 'Hall of Fame' to improve performance assessment. This study experimentally examines a selection of these techniques in chapter 6, and subsequently introduces a new coevolutionary performance measuring technique based on Formula One Grand Prix [48] in section 6.4.1.

Relative fitness evaluation

The first improvement to more accurately reward individuals for their performance was introduced by Rosin and Belew [112] in 1995. *Relative fitness evaluation* aims to measure the performance of a primary population's individuals in relation to a competing population's individuals. This can be achieved by using one of three approaches.

The first approach is referred to as calculating the individual's *simple fitness*, and involves taking a selection of individuals from the opposing population to serve as a benchmark suite of opponents. The simple fitness of a particular individual in the primary population corresponds to the number of benchmark opponents it was able to beat.

The second approach to relative fitness evaluation is known as *fitness sharing*. Fitness sharing extends the aforementioned simple fitness measurement and compares the number of structurally similar opponents in the primary population. The simple fitness value is subsequently divided by the number of similar individuals in the primary population, thereby rewarding individuals with a larger diversity.

The last approach aims to reward individuals that beat opponents very few other individuals could beat, and is known as *competitive fitness sharing*. An individual's fitness is inversely calculated to the number of fellow individuals that could beat a particular opponent, thereby rewarding more points to an individual that was able to beat an opponent no or few other primary population members could beat.

Fitness sampling

The second major approach to more accurately determine an individual's fitness was also introduced by Rosin and Belew [112] in 1995. The previous discussion only mentioned the 'selection of a set of benchmark opponents', without detailed reference to how the selection is performed. The manner in which the opponents are chosen can greatly influence the overall performance value for a particular individual. A number of selection methods exist to perform the required *fitness sampling*, and a few of these are also experimentally examined in this study.

The first is the obligatory *all versus all* scheme, in which each individual in the primary population competes against each of the individuals in the opposing population. This method provides the most accurate assessment of fitness, but in comparison with other approaches induce a significant performance penalty – especially with large population sizes. A variation of this scheme is called *all versus best*, in which all the individuals in the primary population is tested against the fittest individual of the opposing population.

The next scheme simply involves composing the benchmark suite out of a random selection of one or more individuals from the opposing population, and is formally defined as *random sampling*. This method is drastically less computationally expensive when compared to the *all versus all* method, and is applied to the initial experimental work in this study.

The last two schemes make direct use of the relative fitness measures defined earlier, with the first applying the relative fitness of the individuals to construct a *tournament sampling* scheme. The last method is called *shared sampling* and selects the opponents with the largest shared competitive fitness.

Hall of Fame

Rosin and Belew [113] extended their contribution to optimising coevolutionary approaches by introducing the concept of a 'Hall of Fame' (HOF). The HOF maintains a finite list of unchanged copies of previous best performing individuals. The aim of the HOF is to combat the negative effects of the 'moving target' introduced by the coevolutionary approach, by ensuring that more recent population-best particles still compete against the HOF entrants for a position in the list. This approach allows the evolutionary system to maintain possible characteristics that represented good individuals in the past, but that have been 'lost' due to the process of evolution. It stands to reason that the competitive exposure of these earlier characteristics will increase the overall robustness of the evolved solutions.

2.5.3 Application areas

Coevolution is specifically applied in this study to train intelligent game playing agents in the games of Tic-Tac-Toe, Checkers and the Iterated Prisoner's Dilemma. The applications to game learning have been extensive, with groundbreaking work by Fogel and Chellapilla [53] [27] [26] in combining coevolution with evolutionary programming to train neural networks as game state evaluators. Angeline and Pollack [5] used coevolution to evolve a high-level language to play the game of Tic-Tac-Toe. Further applications of coevolution include the evolution of military strategies, path planning and structural optimisation [46].

2.6 Conclusion

This chapter presented an overview of the different computational intelligence paradigms that are applied in or influenced the implementation of the experimental work in this study. The structure of a neural network and the relevant training approaches were discussed, followed by a broad overview of traditional evolutionary computation techniques. A more recent population-based optimisation method inspired by the flocking behaviour of birds, namely particle swarm optimisation, was discussed thereafter. The different parameters involved in the PSO algorithm were presented, along with recommended settings to induce accelerated convergence. Finally, an overview of coevolution was given to illustrate the type of learning/training environment that the agents will be exposed to for the experimental work conducted in this study.

A training algorithm that incorporates neural networks, particle swarm optimisation and coevolution is presented in the following chapter.

CHAPTER 3

Training with PSO

"Experience teaches only the teachable".
- Aldous Huxley (1894 - 1963)

This chapter builds on the background established by the previous two chapters, and introduces an algorithm that utilises particle swarm optimisation to train feed-forward neural networks as board-state evaluators. Different methods to estimate the training performance are also presented. Finally, the influence of an increased ply-depth is investigated and the viability of using larger game trees during training is examined.

3.1 Introduction

The first chapter provided an overview of traditional game learning paradigms, illustrating the strong focus on game trees as core data structure. The evaluation function employed by the game tree was predominantly developed by human expertise in the specific game. It was noted that one way of adding intelligence to the evaluation process was to replace the man-made evaluation function with a neural network, hoping that it could correctly classify a board state as being favourable or unfavourable. The traditional training process applied to Feed Forward neural networks (as discussed in chapter 2) is no longer valid, since the principal variation of the game tree (the perfect move sequence) is not available, and no error measure can be computed. This chapter provides more detailed information on the use of Particle Swarm Optimisation to train the neural network as board state evaluator, driven by a competitive

```
1. Instantiate particles in swarm.
2. Repeat for 500 epochs:
    (a) Determine each particle's performance.
    (b) Compute neighbourhood's best particle.
    (c) Update velocity and position of particles accordingly.
3. Determine overall best performing particle.
4. Measure playing performance against a random player.
```

Figure 3.1: Abstract training algorithm.

coevolutionary environment. The reader is referred back to chapter 2 for detailed information on the computational intelligence paradigms applied in the algorithm.

The basic algorithm is discussed in section 3.2, after which the details behind the population structure and various performance measures are covered in sections 3.2.1 and 3.3 respectively. The game tree as core data structure is not ignored, and insightful research results are discussed in section 3.4, while considering the use of the game tree during training. The final algorithm is listed in section 3.5.

3.2 Basic algorithm

The use of PSO to train neural networks as game playing agents is not entirely new. Messerschmidt *et al.* posited a training algorithm to learn how to play Tic-Tac-Toe in [97]. Figure 3.1 lists an abstract version of the original training algorithm. In its abstract form, the algorithm can be subdivided into three main sections. The first section requires the implementation of a PSO algorithm and subsequent representation of game agents as particles in the swarm. The second section requires the game engine and a specific way to determine a particle's performance during training. Lastly, the algorithm requires a way to validate the performance of the most successful particle after evolution has taken place.

The general theory behind the PSO algorithm was introduced in chapter 2. The specific aspects surrounding particle representation and other population matters are described below.

3.2.1 Population structure

The use of any PSO algorithm relies on the presence of a population of particles, each represented in a uniform manner, each possessing a velocity as well as some historical information regarding previous best solutions that have been found in the search space.

Each game playing agent relies on the use of a neural network to evaluate game states. Different playing strengths (or varying strategy 'intelligence') are directly related to the strength of the evaluation function. It is logical that each game playing agent therefore contains its own unique neural network configuration – allowing for individual playing behaviour in the context of a larger population.

The training of neural networks using PSO has been very successful in past research, with Van den Bergh illustrating the training of summation unit neural networks [136] and Ismail *et al.* training product unit neural networks [72] with positive results. Van den Bergh also introduced the concept of *cooperative swarms* that break up the neural network into multiple parts, allowing for more specialised optimisation [138] [139]. Mendes *et al.* [96] investigated the performance of a selection of PSO architectures to train neural networks for regression and classification tasks, comparing them with standard back propagation (and some enhanced) methods.

The process of training a neural network through PSO involves the conversion of the complete set of weights that connect the neural network layers into a single augmented vector. This results in each single weight vector to represent a complete neural network, which in turn represents a unique game playing agent. The dimensionality of the particle usually plays a significant role in the performance of an optimisation algorithm, as well as the degree of difficulty associated with finding an optimum solution. For experiments conducted in this study, the neural network weight sets ranged in size from 34 values in the most simplistic configuration to 1531 values in the most complex configuration.

Since a particle is only a different representation of a neural network, the weight initialisation schemes associated with neural networks that were mentioned in the previous chapter still apply. The initialisation process corresponds to positioning the particle in an n-dimensional search space, where n corresponds to the number of weights in the network. The scheme introduced by Wessels and Barnard [146] has proven to work well, but it is also possible to use Faure sequences [132] or some other pseudo random number generator such as Sobol sequences [105] for proper initialisation.

In addition, it is necessary to address the PSO algorithm requirement for a 'personal best' performance comparison. Since each particle represents a unique neural network that competes against fellow members of the population, it stands to reason that the only way of accurately comparing a historic configuration's performance is by forcing both the current and historic configurations to compete against the same individuals. This results in the personal best configuration of a particle to be inserted as an additional player in the population – thereby doubling the population size. The numbers quoted to represent the swarm size in the experimental sections of this study *exclude* the presence of the personal best particles, even though they remain present and form a critical part of the training process.

The individuals in the population each play a game against 5 randomly selected opponents, always starting as player one. The performance of each player is determined, which in turn drives the evolutionary process. The following section looks more closely at various performance measures.

3.3 Measuring performance

Any evolutionary process requires the evaluation of the fitness of an individual in the population. The fitness drives the individuals to inherit traits common to the more successful members of the population. Measuring the performance (fitness) in two-player perfect information games is not as simple as keeping track of the number of games won by each individual. The following sections deal with the intricacies associated with playing position and scoring structures in traditional games, after which two distinct particle performance measures are introduced.

3.3.1 Playing position in turn-based games

Most two-player perfect information games are turn-based. The game rules usually specify how to determine the player that starts the game, be it through some random method (such as the roll of a dice) or based on historical game results (winner of the previous game starts first, or vice versa). One aspect of playing position in turn-based games is the fact that some games suffer from severe playing-side imbalances.

This implies that should both players be aware of the specific game's principal variation, it will be possible for a player to start the game, play a perfect series of moves and never lose – even though the opponent is making the best possible counter-moves. This also assumes that the principal variation of the game does not result in a draw. The only definitive way of accurately measuring the imbalance of a game is by constructing the principal variation through the complete game tree, thereby solving the game. The concept of solving a game was mentioned in section 1.2, and the process is by no means an easy one.

There may be a way to approximate the probabilities for a game tree in order to give an

Table 3.1: Probabilities for a random-moving Tic-Tac-Toe player.

Games won if playing first:	0.58
Games won if playing second:	0.28
Games drawn:	0.14

Table 3.2: Probabilities for a random-moving Checkers player.

Games won if playing first:	0.43
Games won if playing second:	0.43
Games drawn:	0.14

indication of any potential balance problems. Messerschmidt *et al.* achieves this by competing two random-moving players for an extended period of time (usually in the order of 1 million or more games) and noting the number of games that are won, lost or drawn [97].

The games of Tic-Tac-Toe and Checkers are examined as examples of perfect information games in the experimental sections of this study. Tables 3.1 and 3.2 list the results of 1 million games between random-moving players for Tic-Tac-Toe and Checkers respectively.

It is interesting to note the severe balancing problems present in Tic-Tac-Toe, and the almost even playing field present in Checkers.

3.3.2 Incorporating win, lose and draw

The easiest measure of the success of a player in any particular game is simply the number of games won. Some games however do include the possibility of draws and usually assign a fraction of the payoff normally associated to winning a game instead, should a draw occur between two players. Each player builds up a collection of 'points' during the progression of the round-robin tournaments. The total points awarded to each player forms part of its fitness calculation, and has a direct impact on its location (or importance) in the population during evolution.

Previous work by Chellapilla and Fogel [53] [27] [26] on evolving intelligent Checkers playing agents assigned +1 for a win, 0 for a draw and -2 for a loss. The larger negative value assigned to losing (as compared to the positive value for winning) is attributed to the fact that the system should clearly 'punish' the player for losing. Messerschmidt *et al.* [97] followed the exact same scoring structure for training Tic-Tac-Toe playing agents.

A new approach to game-by-game performance analysis is introduced in section 6.4.1, based on the well-known Formula One Grand Prix scoring structure – including the previous and most recent editions introduced by the Fédération Internationale de l'Automobile (FIA) [48].

It should be noted that all of the aforementioned scoring measures are only used during the evolutionary training process, and do not factor into the benchmarking of a specific game playing agent. The next section takes a closer look at the technique employed by Messerschmidt *et al.* to accurately determine a game playing agent's performance. A refined version developed by the author for use in Checkers agent benchmarking is described thereafter.

3.3.3 Messerschmidt performance measure

After training has been completed, the global best particle is determined. This particle represents the pinnacle of playing behaviour achieved after 500 epochs of evolution. Since coevolutionary training is employed to drive the PSO process, the fittest individual is based solely on its originating environment. Different populations will result in different 'intelligent players', mainly due to the constantly adapting optimum (also referred to as a 'moving target') common to all coevolutionary environments. It is important for comparative study to benchmark the final evolved player against a common and consistent player not vulnerable to the underlying genetic deficiencies introduced by the training process.

The use of a player making moves at random (a 'random player') has previously been employed by various researchers [47] [6] [8] [97] as a benchmark player. After approximating the probabilities for the Tic-Tac-Toe game tree (as illustrated in table 3.1), Messerschmidt *et al.* derives a suitable performance measure that takes into account the number of games won by the agent when starting the game, as well as when playing second. The performance measure, M, is defined as follows:

$$M = (w_1 - 58.8\%) + (w_2 - 28.8\%) \tag{3.1}$$

where w_1 represents the percentage of games won against the random player when the evolved strategy play as player one, and w_2 represents the percentage of wins when playing as player two.

A confidence value is also computed to strengthen the statistical soundness of the reported performance value. The confidence value is computed as:

$$M \pm \left(z_{\alpha/2} \times \frac{\hat{\sigma}}{\sqrt{n}} \right) \tag{3.2}$$

where $z_{\alpha/2}$ is such that $P[Z \geq z_{\alpha/2}]$, with $Z \sim N(0,1)$ and α is the confidence coefficient. Z is a random real number that Messerschmidt *et al.* sets equal to the outcome of the benchmark games [97]. The confidence coefficient is set to be 90% (or $1-\alpha = 0.9$). The $\hat{\sigma}$ term in equation 3.2 signifies the standard deviation for a particular player, given by:

$$\hat{\sigma} = \sqrt{\hat{\pi}_1(1-\hat{\pi}_1) + \hat{\pi}_2(1-\hat{\pi}_2) + 2\hat{\sigma}_{12}} \tag{3.3}$$

where $\hat{\pi}_1$ and $\hat{\pi}_2$ represent the probabilities for winning as player one and player two respectively – computed by dividing the number of games won by the number of games played. Lastly, the term $\hat{\sigma}_{12}$ represents the covariance, defined as:

$$\hat{\sigma}_{12} = \frac{1}{m-1}\left(\sum_{i=1}^{m}(x_{1i}x_{2i}) - \frac{(\sum_{i=1}^{m} x_{1i})(\sum_{i=1}^{m} x_{2i})}{m}\right) \tag{3.4}$$

where m represents the total number of games played, x_{1i} and x_{2i} represents the outcome for game i as player one and two respectively – assuming the value 1 represents a win and 0 a loss for a particular game.

Successful game playing agents will have a high final M value when benchmarked against a random player, with lower M values signifying poor playing ability. The Messerschmidt *et al.* performance measure is applied to experimental work conducted on Tic-Tac-Toe games, discussed in the following chapter.

3.3.4 Franken performance measure

The aforementioned performance measure introduced by Messerschmidt *et al.* [97] works well in simple games where winning is a major factor in measuring player performance. When applied to more intricate (and balanced) games, such as Checkers or Chess, the performance measure does not provide an accurate representation of player performance. Taking highly publicised tournament games in Chess and Checkers as an example (such as the Kasparov and Deep Blue duels, as well as the Tinsley and Chinook encounters), the winning margins are more often than not decided over a single win, with the participants leaving a series of draws in their wake.

Drawing is an important aspect of more complex games, and should be factored into a performance measure. The mean of the observed probabilities during benchmarking is calculated using the standard mean function for discrete random variables [100] as follows:

$$F = \sum_{i=1}^{3} x_i f(x_i) \tag{3.5}$$

where x_i represents a weight associated with losing, drawing and winning the game respectively, and $f(x_i)$ represents the corresponding outcome probability. When applied to the experimental work (15 simulations of 10000 games), the mean is calculated twice, once for 150000 games played as player one, and once for 150000 games played as player two. The average of the two means is calculated and scaled to a value between 0 and 100. A value closer to 0 means that the player is losing a lot of its games, while a value closer to 100 indicates that a player is winning most of its games.

Substituting the random Checkers player's values (as listed in table 3.2) into the above equation yields a performance value of 50, indicating that the random player wins as many games as it loses, which is true. This performance measure will benefit a player that draws and wins more of its games as opposed to winning and losing equal amounts.

The 'Franken performance measure' is applied in conjunction with the Messerschmidt *et al.* performance measure during experimental work conducted with Checkers in chapters 5 to 7.

3.4 Tree depth

Before presenting the final training algorithm, a last aspect relating to training performance should be examined. The bread and butter of traditional game engine programmers are the game tree and its associated optimisations – as mentioned in chapter 1. A lot of effort goes into optimising the performance of the tree algorithm in order to be capable of analysing a state further into the future (deeper down in the tree). It should be interesting to observe the trade-off between evaluation function strength and tree depth. Before investigating this particular problem in section 3.4.1, a debate regarding the inclusion of a game tree in the intelligent evolutionary approach should first be resolved.

Even though the game tree is considered to be a 'traditional approach', its underlying structure is invaluable during training. In its primitive form (single ply), the game tree allows for the immediate expansion of all possible game states to follow the current state. This move generating process forms the basis of any perfect-information game playing system and cannot easily be argued against.

The impact on performance when increasing the tree depth does however have to be taken into account. As the tree depth is extended, the number of board states that need to be evaluated grows exponentially. The intelligent evaluation process itself (using a neural network) is also more processor intensive than most of its hard-coded human crafted counterparts.

Since the training and benchmarking processes rely significantly on stochastic techniques, a large number of simulations need to be completed in order to dampen the influence of noise induced by these methods. An increase in the number of simulations, coupled with an increase in tree depth pose a tremendous constraint on the time allocated to training and benchmarking.

Previous work by Tesauro on TD-Gammon [130] [131] limited the tree depth to single ply.

Even though the main reasons for this decision were largely due to the influence of the dice-roll in this probabilistic game, Tesauro was still able to evolve an expert Backgammon player capable of beating international computer and human champions.

For the experimental work conducted in this study, a single ply game tree is used in order to compensate for the large number of simulations required for comparative study. Since the focus of the study is to study the application of PSO and its related parameters to game learning, and not to create a world champion game playing agent, the aforementioned decision is justified. Some studies with extended tree depth (2-ply) are however described in chapter 7, including sections that cover the comparison of performance between evolved players and hard-coded evaluation functions, as well as the impact of an extended tree depth on training behaviour.

3.4.1 Increasing the depth

Even though a decision has already been made regarding the tree depth to be used for experimental work in this study, the following section briefly looks at research that aimed to determine the relationship between an increase in tree depth and the strength of the evaluation function.

A recent article by Heinz [66] provides a compact history of research in relating tree depth to playing performance – work that spans over 15 years. First pioneered by Thompson [133] with 100 self-play experiments in 1982, work conducted by Berliner *et al.* [14] is worth mentioning in more detail. Berliner and his research team are the creators of the chess program HITECH. In order to address this research problem the team crippled HITECH's evaluation function (in effect reducing its 'intelligence'), and called the new version of the program LOTECH. LOTECH was allowed to search one ply deeper into the search tree than its 'smarter' companion, HITECH. After allowing HITECH and LOTECH to compete in over a thousand round-robin games, each with its own ply-depth restriction, it became clear that the less intelligent but further seeing LOTECH consistently beat the smarter but shallow seeing HITECH. It should be within reason to suspect that a point of cross-over exists where an increase in ply-depth no longer matters – more prominently referred to as diminishing returns – that will thereby allow HITECH to overtake LOTECH. The researchers observed a slight level of diminishing returns after 6-ply, but solid proof of significant diminishing returns in chess has been hard to find.

Experimental work conducted prior to 2003 by Fierz [51] on his world-renowned Checkers program 'Cake++', did show significant signs of diminishing returns at very deep ply depths, illustrating the scientific difference between the two game application areas.

3.5 Final algorithm

The final training algorithm is listed in figure 3.2, detailing the expansion of the various parts of the algorithm as discussed in the preceding sections of this chapter. The same algorithm is applied without change to experimental work on Tic-Tac-Toe and Checkers training in the following chapters, and a slightly modified version is applied to the Iterated Prisoner's Dilemma at the end of this thesis.

In summary, the algorithm broadly approaches the learning task by taking into account the fact that traditional game tree techniques require human intelligence for board state evaluations. In order to learn its own game strategy, each game playing agent is equipped with a neural network that performs board state evaluations on a single ply game tree. Traditional neural network training approaches require the calculation of an error measure before updating the weights. Since no error measure can be computed (no perfect target evaluation exists), an evolutionary method of training neural networks is required. Each individual game playing agent is represented as a particle in a swarm, with the dimensions of the particle equal to the number of the neural network weights. By 'flying' the particle through the n-dimensional search space, the weight values are updated and the network trained. The particle swarm optimisation method requires the calculation of a fitness value in order to perform the required velocity and position updates. Once again, since no pure mathematical approach is available to examine the game playing strategy, the swarm of particles compete in a coevolutionary fashion in order to determine each particle's relative fitness. The training process continues for 500 epochs after which the best individual is benchmarked against a random-moving player.

3.6 Conclusion

This chapter aimed to bring together the various game learning and computational intelligence concepts – previously only discussed in isolation – and combine them into a training algorithm to be used for the remainder of the experimental work conducted in this study. An abstract training algorithm based on work done by Messerschmidt *et al.* was presented, and subsequently expanded. The first discussion focused on population structure, and how PSO has been used to train neural networks. It explained the representation of a neural network's weight set as a single vector – thereby forming the particle in the swarm.

The playing position in games and subsequent balance problems prevalent in two-player perfect information games were discussed alongside various ways to incorporate winning, losing and drawing in a typical game. The need for a benchmark performance measure for the eventual best evolved individual player was discussed, after which the details behind the specific measure introduced by Messerschmidt *et al.* were listed. A different approach to performance measurement, aptly dubbed the 'Franken performance measure', were introduced thereafter to cope with the occurrence and significance of draws in more intricate games such as Checkers.

The chapter ended with a discussion on the depth of the game tree, its possible influence on game playing performance as well as its impact on training time. The detailed training algorithm provided at the end of the chapter incorporated all the necessary points of discussion into a usable algorithm for training game playing agents using PSO.

```
1. Instantiate population of agents.
2. Repeat for 500 epochs:
   (a) Add each agent's personal best NN configuration to the population.
   (b) For each individual in the population:
       i. Randomly select 5 opponents and play a game against each, always
          starting as 'player one'.
          A. Generate all single valid moves from the current board state.
          B. Evaluate each board state using the neural network.
          C. Select the highest scoring board state, and move appropriately.
          D. Test if game has been won or drawn.
          E. Switch players at end of turn, until game over.
       ii. Assign +1 point for a win, -2 for a loss, and 0 for a draw after every game.
   (c) Compute best performing particle according to PSO algorithm in use.
   (d) For each agent (excluding personal best) in the population do:
       i. Compare performance against personal best.
       ii. Compare performance against neighbourhood's best particle.
       iii. Update velocity for each particle according to PSO algorithm.
       iv. Update weights according to PSO algorithm.
3. Determine single best performing agent in whole population
   (including personal best players).
4. Best agent plays 10000 games against a random moving player as 'player one'.
5. Best agent plays 10000 games against a random moving player as 'player two'.
6. Return to step 1 until 30 simulations have been completed.
7. Compute confidence interval over the 30 completed simulations.
8. Compute performance value over the 30 completed simulations.
```

Figure 3.2: Detailed training algorithm.

CHAPTER 4

Tic-Tac-Toe

"Everything should be made as simple as possible,
but not one bit simpler".
- Albert Einstein (1879 - 1955)

The training algorithm derived in the previous chapter is now applied to the computationally modest problem of evolving intelligent Tic-Tac-Toe players. An overview of the problem is followed by a series of experimental work, analysing various PSO architectures and a selection of basic PSO parameter choices. The training technique is shown to be successful, with the Von Neumann architecture showing promise as neighbourhood structure of choice.

4.1 Introduction

With all the necessary background information established in chapters 1 and 2, and a practical training algorithm constructed in chapter 3, the first set of experiments can now be conducted to determine the viability of using PSO as an approach to game learning.

The well-known computationally modest game of Tic-Tac-Toe (Noughts and Crosses) is used as an initial testbed for the posited framework. The game of Tic-Tac-Toe is introduced in section 4.2, after which a discussion on possible opponents available during training and benchmarking in section 4.3, leads to a description of one of the main experimental testing criteria – the performance of various PSO neighbourhood structures – in section 4.4.

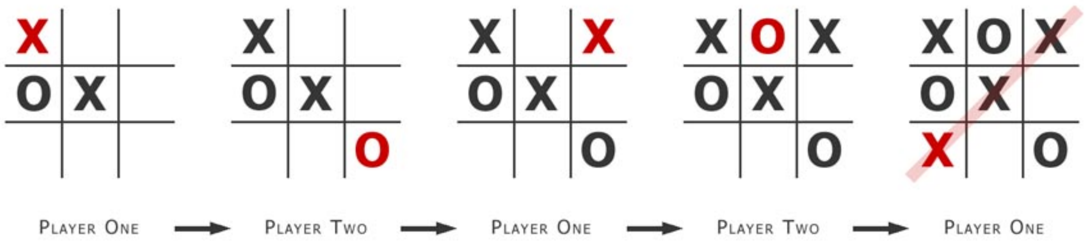


Figure 4.1: Typical end-game sequence for a Tic-Tac-Toe game.

The experimental procedure is clearly outlined in section 4.5, followed by a series of experimental results in section 4.6. The experimental results are compared to published results from Messerschmidt *et al.*'s [97] experiments that made use of GAs and traditional PSO structures to evolve Tic-Tac-Toe game playing agents, with interesting results.

4.2 Game rules

Tic-Tac-Toe is a two-player perfect-information game, played on a 3-by-3 grid of initially empty squares. Each player is assigned a set of game pieces, enabling them to play as either a circle (nought) or a cross. Starting conditions to the game are not set in stone, and a random player selection for the first move in the first game can be followed up by a 'loser plays first' or 'winner plays first' scheme thereafter.

As explained during the discussion on imbalanced games in the previous chapter, a player starting the game has a significant advantage to win in the end. Therefore, all the players in the evolving population play 5 games as player one, and are selected at random to play an additional 5 games as player two – forming the opponents for a particular individual currently playing as player one.

Each player makes a move in turn by placing a game piece (either a cross or a circle) in a specific open square. A player is not allowed to place a piece in an already occupied square, and the game ends when all the squares have been filled up. It is the aim of each player to arrange a sequence of three pieces into a straight line, either horizontally, vertically or diagonally. At the same time the player should keep its opponent from achieving the same goal – for whoever achieves it first, wins the game. In the case of all the squares filling up without a straight line being formed for either player, a draw is declared.

A typical 'end-game' sequence is depicted in figure 4.1, with the current player's move highlighted in red (the opening moves for each player are not shown individually, but can be inferred from the first diagram – assuming crosses started the game). The player playing crosses eventually wins the game by aligning three pieces in a straight line along the right diagonal.

4.2.1 Scoring structure

The traditional scoring structure for Tic-Tac-Toe simply assigns a point to the player winning the game. Other variants to scoring such as a 'doubling cube' found in Backgammon are not available in Tic-Tac-Toe. As explained in the previous chapter's discussion on how to incorporate winning, drawing and losing (section 3.3.2), a value of +1 is assigned to each player winning a game, 0 for a draw, and -2 for a loss. The larger losing factor is once again only used to clearly punish a losing player, keeping its neural network from stagnating on weight sets that produce poor game-state evaluations.

4.3 Choosing an opponent

Tic-Tac-Toe is introduced in this chapter as a 'computationally modest' game, due to the fact that it has already been solved, and the complete game tree consisting of 9-factorial nodes can be algorithmically constructed without much effort – allowing for a potential 'perfect player' as an opponent.

The use of human players as training opponents are not viable, since humans are not consistent players and are subject to lapses in both concentration and fatigue. Time is also a valuable resource during experimental testing, and will be wasted by competing against human players. Instead, the random-moving player as introduced in the previous chapter is employed as a consistent, yet unpredictable player. The Rand3 pseudo-random number generator designed by Knuth [105] is used for consistency.

Previous researchers have made use of the random-moving player as a training opponent for Tic-Tac-Toe games [47] [6]. In addition, the researchers also provide intermediate levels of more intelligent opponents, using hand-crafted evaluation functions that allow for improved intelligent playing behaviour, forming a stepping stone for continuous learning. Usually, a perfect player is also present in the population to form an upper-bound on the population performance. Since the focus of the training technique is coevolution, with a strong emphasis on the *lack* of human intelligence, no hand-crafted human evaluation functions are present in the training population, and the random-moving player is only employed to benchmark the ultimate evolved player's performance.

4.4 Choosing a PSO architecture

Section 2.4.1 introduced the three primary PSO architectures for information sharing, namely the Global Best (GBest), Local Best (LBest) and Von Neumann (VN) structures. Previous research in the application of PSO to function optimisation [96] [137] show a clear distinction in performance between the different structures. Messerschmidt *et al.* [97] compared the performance of GA and PSO-based coevolution in the game learning domain, showing a definite advantage to using PSO. It should be interesting to examine the specific impact the neighbourhood structure has on an individual's training and playing performance.

4.4.1 Parameter selection

For the experimental work conducted in this study, various parameters pertaining to PSO are tested to determine their specific influence on training and playing behaviour. The various parameters relating to the functioning of the PSO algorithm were described in section 2.4.2. Experiments on Tic-Tac-Toe will largely focus on PSO architecture comparisons, with a set of default parameter settings based on existing research.

The cognitive (c_1) and social (c_2) acceleration constants are set to 1.0 each, adhering to the convergence equation by Van den Bergh [137] as listed in section 2.4.2. The inertia weight is fixed at 1.0. A significant departure from the research conducted by Messerschmidt *et al.* is the absence of the VMax parameter in this study. The maximum velocity is not restricted for any of the particles present in the experiments to compare the three neighbourhood structures' performance. The aforementioned parameter choices adhere to the restrictions identified by Van den Bergh and described in section 2.4.2, thereby allowing the removal of the maximum velocity cap.

For each of the three different neighbourhood architectures, an 'Architecture performance matrix' is constructed, measuring the change in swarm size to the change in hidden nodes over a number of experiments. The reader is reminded that due to the need for a personal best comparison, the swarm size actually reflects only half of the true population size – due to the inclusion of the personal best configuration as an additional player in the population. The personal best configuration is only used as a performance comparison and does not get altered – only replaced if it is outperformed.

Some experiments were performed to try and optimise the aforementioned baseline configurations (parameter settings) using other values for the c_1, c_2, inertia weight and VMax parameters, with adequate success (described in section 4.6.6). The focus for these experiments however, should be stressed to be on neighbourhood structure comparison only.

4.5 Experimental procedure

As already mentioned, the first aim of the experiments in this chapter is to test the viability of using PSO in the game learning domain, on a computationally modest problem. The second major aim of the experimental work is to examine the influence of the various neighbourhood information sharing structures.

The following subsections summarise the exact experimental setup present during the performance analysis.

4.5.1 Training algorithm

The final core training algorithm listed in section 3.5 is applied to the Tic-Tac-Toe problem. As already discussed, each particle consists of an n-dimensional weight vector that corresponds to the weights of a neural network. Each particle can be seen as a single game playing agent in a population of players, with the substitution of particle information (weight values) into a neural network framework resulting in unique board evaluations for that specific player.

The use of coevolution to drive the training process, should result in more intelligent players emerging over time. The best individual after 500 epochs of evolution is benchmarked against a random-moving player.

4.5.2 PSO configuration

In summary, the PSO parameter choices are as follows: $c_1 = c_2 = 1.0$, inertia = 1.0, VMax is not applied. Each of the three neighbourhood structures are compared for performance. The swarm sizes range from 5 to 50 particles in the case of GBest and LBest, and between 15 and 50 particles for Von Neumann. The Von Neumann information sharing structure's lower bound is chosen due to the 2-dimensional lattice representation that may cause the same particle to be represented more than once in the same neighbourhood, which is not allowed. The LBest structure makes use of a neighbourhood size of 5 particles, including the particle currently being evaluated.

4.5.3 Neural network configuration

The neural network has been identified as an evolvable evaluation function for game board states in section 3.2.1 of the discussion on the training algorithm. An aspect not yet covered by the discussion on neural networks is how the board state is fed into the network. The board state representation differs from game to game, and different approaches will be shown in the later chapters on Checkers.

For Tic-Tac-Toe however, each grid (or board state) is represented as a vector of 9 spaces (numbered from left to right, top to bottom). Each space can have one of three states, namely: it can be an empty space (represented as 0.5), occupied by a 'friendly' (own) piece (represented as 1.0) or it can be occupied by an opponent piece (represented as 0.0). The use of the 'my piece' versus 'opponent piece' representation scheme, results in the exact same board-state to be inversely represented to each player. This approach differs from other approaches used by various researchers [97], and effectively doubles the number of input states available to the neural network – hopefully allowing it to better differentiate between states caused by 'starting the game' or 'playing second'. Messerschmidt *et al.* [97] instead included an additional neuron to identify the current player. In order to differentiate between piece types (noughts or crosses), a different fixed value is used for each player.

Since the piece values for the pieces in the game do not vary (a nought and cross have equal importance/value), the input values can remain constant throughout training. Chellapilla and Fogel [53] [27] [26] evolved the value for Checkers pieces, since a Checkers-king is supposed to be of higher value than a Checkers-man. The piece-values for this study's Checkers simulations are covered in the following chapter.

4.5.4 Setting the benchmark

The benchmark playing behaviour for random-moving Tic-Tac-Toe players were listed in table 3.1, with the player moving first winning approximately 58% of the time, and the player playing second winning approximately 28% of the time. These percentages directly form part of the Messerschmidt *et al.* performance measure, as detailed in section 3.3.3 and equation 3.1. The performance values calculated from the experimental work on Tic-Tac-Toe in this study will be directly compared to the values computed by Messerschmidt *et al.*

4.5.5 Statistical soundness

Confidence intervals are calculated with regard to the Messerschmidt *et al.* performance values, as clearly described in section 3.3.3. A confidence coefficient (alpha value) of 0.9 is used, and the required covariance and standard deviation values are computed.

4.6 Experimental results

The following subsections provide an analysis of experiments conducted according to the aforementioned experimental procedure. The three architecture performance matrices (one for each

Table 4.1: Architecture performance matrix - standard GBest. Larger M-values preferred.

	Hidden Units							
Particles	3	5	7	9	11	13	15	Averages
5	23.87500 ± 0.00204	18.57200 ± 0.00206	31.30600 ± 0.00199	22.73530 ± 0.00205	18.03670 ± 0.00205	28.16830 ± 0.00201	21.00530 ± 0.00204	23.38551 ± 0.00204
10	24.68600 ± 0.00204	20.01570 ± 0.00205	30.02070 ± 0.00200	30.42130 ± 0.00199	26.02430 ± 0.00201	25.70270 ± 0.00205	21.84070 ± 0.00204	25.53020 ± 0.00202
15	29.45630 ± 0.00200	25.82200 ± 0.00202	29.74770 ± 0.00200	27.99130 ± 0.00201	28.87530 ± 0.00200	25.60500 ± 0.00201	22.53070 ± 0.00204	27.14690 ± 0.00201
20	26.86470 ± 0.00200	27.74970 ± 0.00201	27.42330 ± 0.00201	27.21170 ± 0.00201	29.45400 ± 0.00199	30.10430 ± 0.00198	26.17900 ± 0.00203	27.85524 ± 0.00201
25	25.50830 ± 0.00202	29.46430 ± 0.00200	24.14230 ± 0.00203	28.69970 ± 0.00199	26.00700 ± 0.00201	31.54570 ± 0.00196	29.01270 ± 0.00200	27.76857 ± 0.00200
30	22.26000 ± 0.00204	22.85600 ± 0.00204	25.35570 ± 0.00202	27.94700 ± 0.00200	27.61600 ± 0.00201	23.54570 ± 0.00202	28.45130 ± 0.00200	25.43310 ± 0.00202
35	21.97270 ± 0.00202	30.97430 ± 0.00199	27.88270 ± 0.00201	26.04500 ± 0.00201	24.17770 ± 0.00203	28.50270 ± 0.00200	31.13130 ± 0.00197	27.24091 ± 0.00201
40	27.72070 ± 0.00200	25.50430 ± 0.00202	24.12330 ± 0.00202	29.04930 ± 0.00200	26.66370 ± 0.00201	29.56730 ± 0.00199	26.89400 ± 0.00202	27.07466 ± 0.00201
45	30.05800 ± 0.00199	28.09400 ± 0.00200	25.32670 ± 0.00201	32.35430 ± 0.00197	29.31170 ± 0.00199	32.44800 ± 0.00198	27.37030 ± 0.00201	29.28043 ± 0.00199
50	25.82900 ± 0.00202	25.28000 ± 0.00202	26.73500 ± 0.00200	29.00330 ± 0.00200	24.69870 ± 0.00203	27.92670 ± 0.00202	22.50300 ± 0.00204	25.99653 ± 0.00202
Average M	25.82307 ± 0.00202	25.43323 ± 0.00202	27.20634 ± 0.00201	28.14582 ± 0.00200	26.08651 ± 0.00201	28.31164 ± 0.00200	25.69183 ± 0.00202	

of the neighbourhood structures) are listed in section 4.6.1, and a selection of the newly computed results are graphically compared to the experimental results previously obtained by Messerschmidt *et al.*. The performance value computed by the Messerschmidt *et al.* measure is abbreviated as an 'M-value' for the duration of the study. The matrices are visualised as heightmaps (surfaces) and analysed accordingly in section 4.6.2. The effects of an increase in swarm size and hidden nodes are discussed in sections 4.6.4 and 4.6.3, followed by a brief illustration of neural network weight convergence in section 4.6.5. The effect of using an optimised parameter set (mentioned earlier) is graphically depicted in section 4.6.6.

4.6.1 Comparison of architecture performance matrices

Tables 4.1, 4.2 and 4.3 list the newly computed architecture performance matrices for the standard GBest, LBest and Von Neumann neighbourhood structures respectively. Each matrix depicts the change in performance for an increase in swarm size (from top to bottom) and an increase in the hidden layer size (from left to right). The result of the Messerschmidt *et al.* performance measure after 150000 games as player one, and 150000 games as player two against a random-moving player is listed in each matrix entry, along with its associated confidence value.

A couple of interesting observations can be made before applying any of the additional visual

Table 4.2: Architecture performance matrix - Local Best. Larger M-values preferred.

Particles	Hidden Units							Averages
	3	5	7	9	11	13	15	
5	24.13830 ± 0.00203	27.14100 ± 0.00200	28.59900 ± 0.00200	28.78330 ± 0.00200	29.28070 ± 0.00199	27.57530 ± 0.00200	25.08970 ± 0.00202	27.22961 ± 0.00201
10	32.92700 ± 0.00195	28.65970 ± 0.00199	30.92730 ± 0.00197	29.12300 ± 0.00198	30.20130 ± 0.00198	28.58070 ± 0.00200	29.43730 ± 0.00200	29.97947 ± 0.00198
15	33.69430 ± 0.00196	28.78600 ± 0.00201	30.15430 ± 0.00199	33.61730 ± 0.00195	28.67500 ± 0.00200	30.04970 ± 0.00199	30.68630 ± 0.00197	30.80899 ± 0.00198
20	28.80500 ± 0.00199	30.26700 ± 0.00198	31.10170 ± 0.00197	31.37070 ± 0.00198	33.14500 ± 0.00195	33.04930 ± 0.00196	33.88700 ± 0.00195	31.66081 ± 0.00197
25	29.51230 ± 0.00200	32.28630 ± 0.00196	30.48330 ± 0.00198	28.60870 ± 0.00201	29.64030 ± 0.00199	34.76100 ± 0.00195	29.16870 ± 0.00199	30.63723 ± 0.00198
30	26.91530 ± 0.00201	28.94070 ± 0.00199	31.80770 ± 0.00197	28.12200 ± 0.00200	30.24300 ± 0.00198	30.59830 ± 0.00199	31.96370 ± 0.00196	29.79867 ± 0.00199
35	31.50770 ± 0.00198	29.48670 ± 0.00199	30.76100 ± 0.00198	25.98430 ± 0.00202	29.39330 ± 0.00200	31.70430 ± 0.00198	28.40900 ± 0.00200	29.60661 ± 0.00199
40	28.52870 ± 0.00199	30.09000 ± 0.00197	31.00270 ± 0.00199	30.22700 ± 0.00199	27.64700 ± 0.00200	30.73770 ± 0.00198	31.88130 ± 0.00197	30.01634 ± 0.00198
45	28.56300 ± 0.00199	27.46170 ± 0.00201	27.89400 ± 0.00201	27.40030 ± 0.00200	33.58730 ± 0.00196	30.88930 ± 0.00198	26.91370 ± 0.00201	28.95847 ± 0.00200
50	31.92370 ± 0.00195	30.25670 ± 0.00198	29.12000 ± 0.00200	26.71670 ± 0.00201	32.62700 ± 0.00196	31.38270 ± 0.00197	30.65500 ± 0.00198	30.38311 ± 0.00198
Average M	29.6515 ± 0.00198	29.3376 ± 0.00199	30.1851 ± 0.00199	28.9953 ± 0.00199	30.4440 ± 0.00198	30.9328 ± 0.00198	29.8092 ± 0.00199	

analysis techniques. A search for the best individual constructed from all the simulation runs for each neighbourhood structure, reveals the already well-established fact – GBest performs slightly worse than LBest in the majority of test problems. Looking at the specific performance ratings for Tic-Tac-Toe, the best GBest player scored 32.448 on a configuration with a swarm size of 45 particles, and 13 hidden nodes. The best LBest individual had an improved score of 34.761, with a swarm size of 25 particles and 13 hidden nodes.

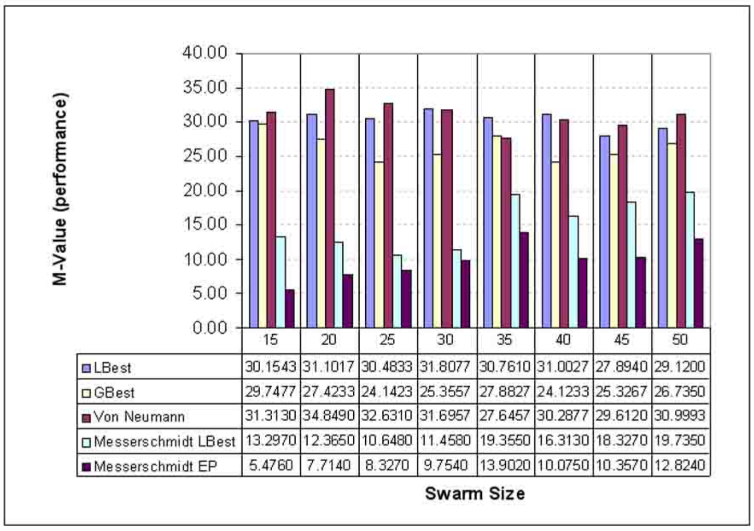
The interesting development is the introduction of the Von Neumann neighbourhood structure to the game learning domain. The best Von Neumann individual scored a superior 34.849, with the smallest swarm size of 20 particles, in addition to only using a hidden layer of 7 nodes.

The trend continues if a global average is computed across the cells of each of the different matrices. The average GBest performance is still very poor, with a value of 26.67121. LBest continues to outperform GBest with an average value of 29.90793, while the average Von Neumann performance remains superior with 30.10109.

Figure 4.2 compares the performance values computed for experimental work on Tic-Tac-Toe in this study to previously published results by Messerschmidt *et al.* [97], assuming a hidden layer of 7 hidden nodes and a swarm size varying between 15 and 50 particles. Messerschmidt *et al.* compared the performance between an evolutionary programming approach to game learning (as first introduced by Chellapilla and Fogel [53] [27] [26]) with a PSO approach to game learning. The results show a definite improvement to using PSO. The results computed

Table 4.3: Architecture performance matrix - Von Neumann. Larger M-values preferred.

Particles	Hidden Units							Averages
	3	5	7	9	11	13	15	
15	31.05670 ± 0.00198	28.75400 ± 0.00199	31.31300 ± 0.00198	31.91870 ± 0.00197	32.29270 ± 0.00196	28.37470 ± 0.00200	29.33330 ± 0.00198	30.43473 ± 0.00198
20	29.14400 ± 0.00199	29.28230 ± 0.00199	34.84900 ± 0.00195	33.33230 ± 0.00196	33.44530 ± 0.00195	32.32530 ± 0.00196	32.03730 ± 0.00196	32.05936 ± 0.00197
25	30.07170 ± 0.00198	30.00000 ± 0.00199	32.63100 ± 0.00196	32.21170 ± 0.00195	31.04230 ± 0.00197	33.52300 ± 0.00196	31.06070 ± 0.00198	31.50577 ± 0.00197
30	28.56370 ± 0.00201	30.88870 ± 0.00197	31.69570 ± 0.00198	28.65230 ± 0.00200	31.56730 ± 0.00197	29.11230 ± 0.00200	26.56270 ± 0.00202	29.57753 ± 0.00199
35	32.63000 ± 0.00196	31.08730 ± 0.00198	27.64570 ± 0.00200	27.96570 ± 0.00200	29.15730 ± 0.00199	31.69670 ± 0.00197	28.48070 ± 0.00201	29.80906 ± 0.00199
40	29.39830 ± 0.00199	28.97830 ± 0.00199	30.28770 ± 0.00198	27.30300 ± 0.00200	29.94200 ± 0.00199	29.31970 ± 0.00200	32.70270 ± 0.00196	29.70453 ± 0.00199
45	29.97770 ± 0.00198	29.61200 ± 0.00199	29.61200 ± 0.00199	29.77130 ± 0.00199	29.71600 ± 0.00200	26.83570 ± 0.00200	27.84630 ± 0.00201	29.05300 ± 0.00199
50	24.87400 ± 0.00201	26.32730 ± 0.00203	30.99930 ± 0.00199	30.71830 ± 0.00199	28.92130 ± 0.00200	32.37170 ± 0.00197	26.44130 ± 0.00201	28.66474 ± 0.00200
Average M	29.46451 ± 0.00199	29.36624 ± 0.00199	31.12918 ± 0.00198	30.23416 ± 0.00198	30.76053 ± 0.00198	30.44489 ± 0.00198	29.30813 ± 0.00199	



	15	20	25	30	35	40	45	50
▨ LBest	30.1543	31.1017	30.4833	31.8077	30.7610	31.0027	27.8940	29.1200
□ GBest	29.7477	27.4233	24.1423	25.3557	27.8827	24.1233	25.3267	26.7350
■ Von Neumann	31.3130	34.8490	32.6310	31.6957	27.6457	30.2877	29.6120	30.9993
□ Messerschmidt LBest	13.2970	12.3650	10.6480	11.4580	19.3550	16.3130	18.3270	19.7350
■ Messerschmidt EP	5.4760	7.7140	8.3270	9.7540	13.9020	10.0750	10.3570	12.8240

Figure 4.2: Comparison graph between Messerschmidt *et al.*'s implementation performance and performance values computed in this thesis. Larger M-values are preferred.

for the experimental work in this study clearly outperform even the best available individual score computed by Messerschmidt *et al.*

The increased performance can be attributed to the different manner in which a board state is presented to the neural network, along with the use of slightly adjusted PSO parameters – most notably the absence of the maximum velocity.

Furthermore, the Von Neumann information sharing structure continues its dominance in

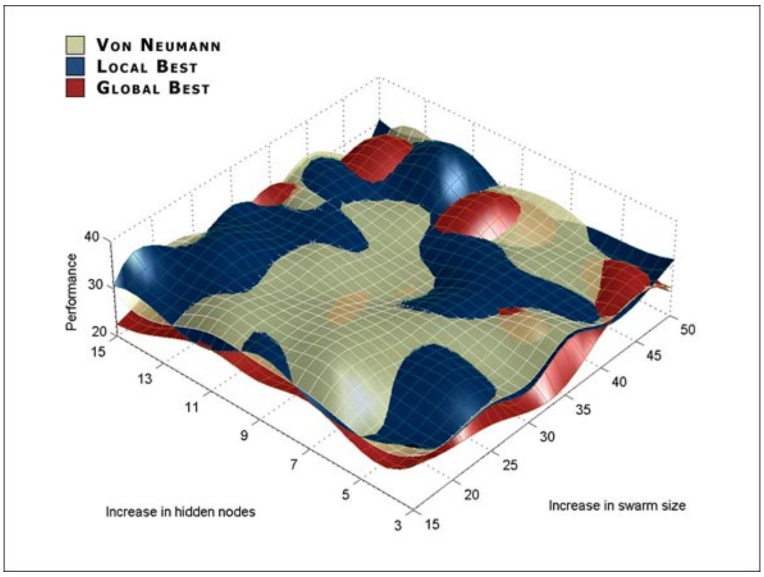


Figure 4.3: Surface plot for architecture comparison.

this specific configuration, outperforming the other neighbourhood structures in 5 out of the 8 cases. The GBest information sharing structure remains the worst performer, only achieving the 2^{nd} best performance on one occasion. The LBest structure manages to provide strong competition to the Von Neumann structure in most cases.

4.6.2 Surface plots

In order to ease global comparison between the different neighbourhood structures, the performance values in each performance matrix can be regarded as a height value, thereby forming a three-dimensional surface when plotted as a heightmap.

The heightmaps constructed from the performance values for the recently computed Tic-Tac-Toe experiments are depicted in figure 4.3, with each surface assigned a different colour for visual distinction. Higher lying areas on the various surfaces indicate increased performance, with the visible parts of a surface indicating its superiority in the specific swarm size/hidden node configuration.

From visual inspection it is clear that the Von Neumann architecture dominates a significant portion of the overall surface area, specifically in configurations with smaller swarm sizes and a fair number of hidden nodes. The LBest architecture surpasses the Von Neumann dominance in experiments with very large hidden layer sizes, while the GBest architecture manages to briefly outperform the other two structures in very large configurations of swarm and hidden layer sizes.

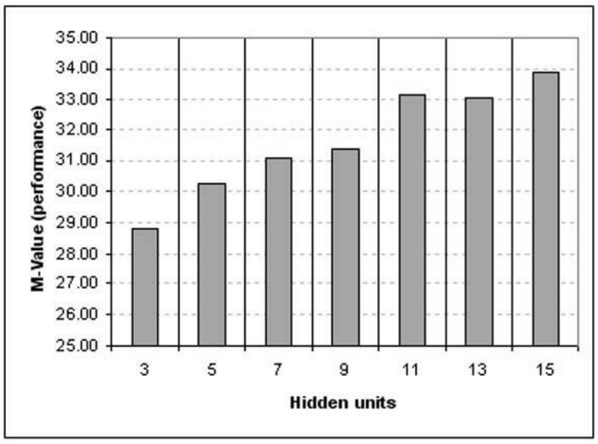


Figure 4.4: Increase in hidden nodes for specific LBest architecture configuration.

The opacity of the Von Neumann surface has been lowered to visibly indicate the difference in performance between it and its nearest underlying competitor. With the lowered opacity enabled, it is clear that the LBest neighbourhood structure remains the biggest competitor to Von Neumann, with GBest sporadically making inroads in both these structures' performance. The three large bumps visible on the surface (for Von Neumann in-line with 7 hidden nodes, GBest and LBest in-line with 13 hidden nodes) indicate the overall best performers for each neighbourhood structure. The LBest structure's 2^{nd} best performer (13 hidden nodes, 45 particles) also makes a significant visual impact.

4.6.3 Increase in hidden nodes

It is difficult to identify a definite trend among the neighbourhood structures with regards to a performance increase/decrease after increasing the number of hidden nodes for each configuration. The Von Neumann architecture seems to favour mid-range hidden layer sizes, peaking quite early on as indicated by its best performing player.

Even within a specific neighbourhood structure the trend cannot be consistently isolated. Figure 4.4 illustrates an almost perfect example of an increase in performance due to an increase in the number of hidden nodes for the LBest neighbourhood structure. The example is based on a configuration with a swarm size of 20 particles. It would seem that both GBest and LBest tend to prefer larger hidden layer sizes, as both these neighbourhood structures' best performing individuals are located at the far end of the hidden layer size spectrum.

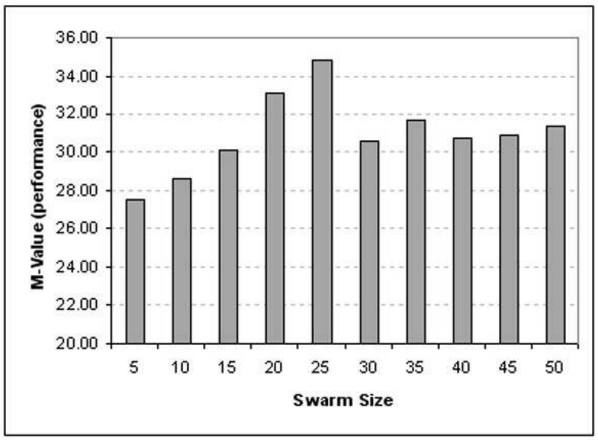


Figure 4.5: Increase in swarm size for specific LBest architecture configuration.

4.6.4 Increase in swarm size

The inability to identify a definite trend among hidden layer sizes repeats itself with an increase in swarm sizes. After some analysis it would seem as if both the Von Neumann and LBest neighbourhood structures prefer mid-range swarm sizes, which is also confirmed by the location of both these structures' best performing individuals.

Figure 4.5 depicts the change in performance for the LBest neighbourhood structure, assuming a configuration with 13 hidden nodes. The graph clearly shows the initial rise in performance, early peak, and resultant stabilising of the performance value as the swarm size is increased. It should be stressed that the aforementioned trend is not consistently repeated throughout the various neighbourhood structures.

Finally, it would seem as if the GBest neighbourhood structure tends to prefer larger swarm sizes, as it is the only instance in which it manages to make any serious attempts to overtake the LBest and Von Neumann neighbourhood structures as dominant players. This is more clearly illustrated with the aid of the surface plot (figure 4.3).

4.6.5 Convergence

The ability of the particles in the swarm to converge on a (hopefully optimum) value, is a defining characteristic of the PSO algorithm. An easy way to determine particle convergence is the plot of the absolute neural network weight values over the duration of the simulation – in this case 500 epochs. Figure 4.6 depicts such a plot for a configuration of 20 particles and 5 hidden nodes utilising the LBest structure, clearly illustrating the convergent behaviour on a stable solution.

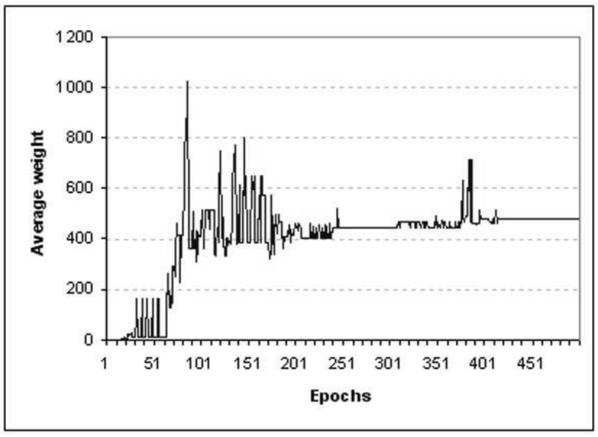


Figure 4.6: Convergence of neural network weights.

Studies by Messerschmidt *et al.* on convergence for PSO in game learning show instances of sudden oscillating behaviour arising after the duration of a seemingly stable state [97]. This can be attributed to the coevolutionary training scheme developing a player able to exploit the genetic weakness of the larger population – thereby causing the particles to move to the newly located optimum. It may also be a consequence of stray particles' maximum velocity escalating in such a way as to negatively effect the swarm – pulling particles to undiscovered and unrelated areas of the search space.

4.6.6 Optimising performance

As a final note, it should be stated that the default PSO parameters selected to perform these experiments have been chosen to cause convergent swarm behaviour on suitable solutions – in order to analyse neighbourhood information sharing structures. It stands to reason that with some level of 'tweaking', each and every simulation can be optimised to produce more intelligent players. This allows for any number of PSO parameter configurations to be successful – a fact that should not be forgotten.

An example optimisation for an LBest configuration of 5 particles is depicted in figure 4.7. The 'comparative configuration' reflects the original parameter choices as described in the aforementioned sections, and the 'optimised configuration' reflect the performance for a parameter selection of $c_1 = 1.0$, $c_2 = 2.5$ and VMax $= 170$. The performance is drastically improved, yet does not rival the overall best performing player evolved in earlier simulations for the LBest neighbourhood structure.

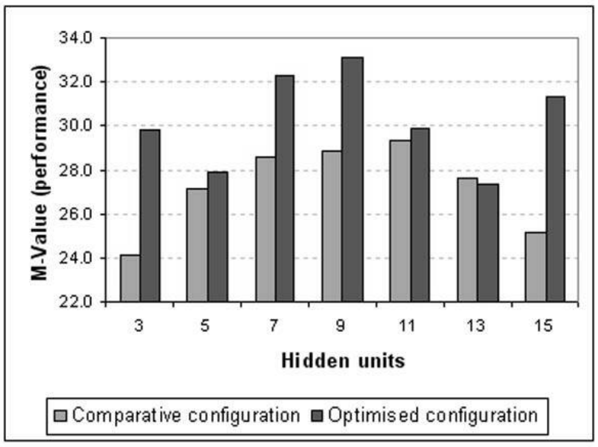


Figure 4.7: Optimised performance for a sample LBest configuration.

4.7 Conclusion

This chapter applied the training algorithm derived in chapter 3 to the computationally modest problem of Tic-Tac-Toe. The specific PSO and neural network settings were discussed, whereafter experimental analysis was conducted to measure the performance of three PSO neighbourhood structures – GBest, LBest and Von Neumann.

It was the first application of the Von Neumann structure to game learning, which yielded surprisingly good results. The Von Neumann structure seems to be superior to both the LBest and GBest neighbourhood structures, capable of evolving more intelligent players – using even smaller swarm sizes and neural network configurations. Visual performance analysis using three-dimensional surface plots of the architecture performance matrices provided more insight into the strong and weak points of each neighbourhood structure.

The swarm has shown to converge on suitable solutions, even though no definite trend in increasing the swarm or hidden layer sizes could be found. The reader is reminded that the default parameter choices were made to cause convergent behaviour, and a uniquely optimal set of parameters could exist for each individual configuration.

The aim of the experiments – to apply the algorithm to Tic-Tac-Toe learning, and to investigate neighbourhood structures – were successfully achieved. The application of the algorithm to a more complex optimisation problem in game learning – that of Checkers – is considered in the following chapters.

CHAPTER 5

Checkers

"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away".
- Antoine de Saint-Exupery (1900 - 1944)

The experimental analysis of the previous chapter is extended to the Checkers domain, which inherently poses a much more difficult optimisation problem due to its much larger search space. The same training algorithm is once again applied, and the neighbourhood structure performance analysed. The Von Neumann architecture proves to dominate once again, but overall results are disconcerting and lead to further in-depth analysis in chapters to come.

5.1 Introduction

After successfully applying a PSO-based training algorithm to evolve intelligent Tic-Tac-Toe agents in the previous chapter, a more challenging problem – Checkers – allows for the analysis of the PSO-based training algorithm on a significantly larger search space.

The game of Checkers is introduced in section 5.2, describing the detailed rules as implemented for the experimental work, along with intricacies surrounding the scoring structure. The choice of benchmarking opponent is once again discussed in section 5.3, followed by a short description in section 5.4 on the specific PSO parameter choices made with regard to the various information sharing neighbourhood structures.

The experimental procedure in section 5.5 summarises the various components that have an

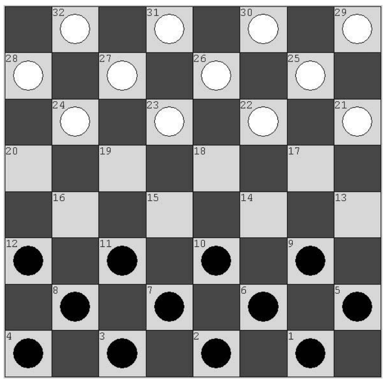


Figure 5.1: Dimensions, numbering and setup of a standard Checkers game board.

impact on the experimental work, including the training algorithm, the PSO and neural network configurations as well as the benchmark results and associated statistical computations.

Section 5.6 provides a listing of various experimental results, with graphical analysis through surface plots and graphs providing more insight into the underlying relationships between the neighbourhood structures. Section 5.7 concludes the chapter by summarising some of the major experimental findings.

5.2 Game rules

The game of Checkers (also known as Draughts) has been so collectively absorbed by the general public, that an actual set of official rules has been hard to obtain. The rules included in the commercial game sets only state the obvious use of the supplied equipment (game pieces and board), along with the tournament conditions on how to start a game and exceptions to using a clock during the game itself [90]. The rules do not even mention the restriction of the checkerboard to only 8-by-8 squares! After querying numerous sources [90] [53], including the official American Checkers Federation rules [2], the following set of rules have been selected as the internationally accepted 'Official (US) rules', and should be used to compare any future experimental work to the experimental results listed in this study.

A standard checkerboard has dimensions of 8-by-8 alternately coloured squares (depicted in figure 5.1). Checkers is a two-player game, with red (sometimes black) and white pieces. Before the game starts, the players toss a coin to determine who plays red (red always starts the game). The starting behaviour is alternated thereafter. As explained in section 3.3.1 on potential imbalanced games, Checkers has shown to not clearly favour either the first or second-moving player. During the training process, each player plays against 5 randomly selected opponents from the competition pool, always starting as player one. The current player is

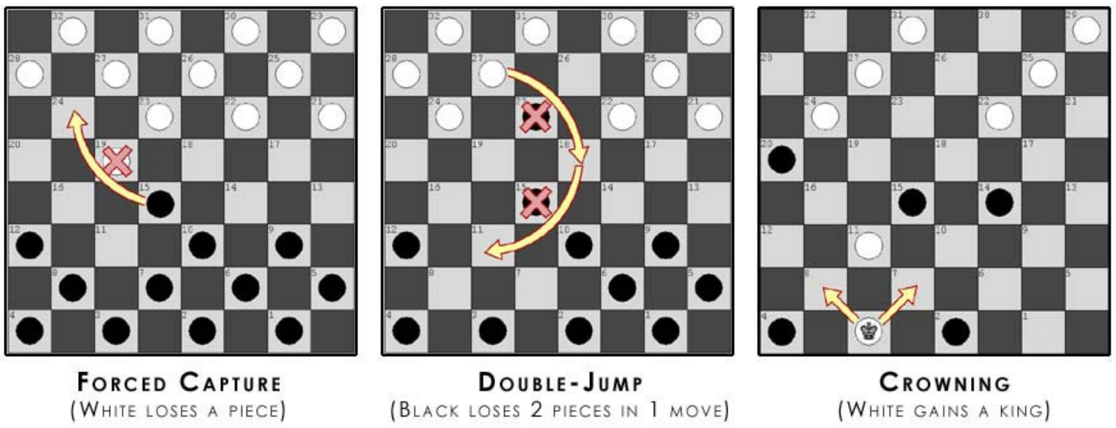


Figure 5.2: Basic rules of Checkers explained.

thereafter available to play an additional 5 games as the opponent (player two) to any of the remaining players in training.

It is the aim of the game to restrict the movement of the opponent's pieces. Each player has to make a move on every turn. The first player that is unable to move – either due to restriction or a loss of pieces – is the loser.

Each player is assigned 12 pieces, referred to as 'men' (standard value piece). The pieces are arranged according to figure 5.1. Each piece is only allowed to move in a forward diagonal direction, for a total distance of one square per turn. A piece is not allowed to move off the side of the board. A piece is not allowed to move on top of or over a friendly piece. A friendly piece is able to jump over an opposing piece, assuming that there is a free landing-spot (an open square) located directly behind the target piece and in-line with the jumping direction (see figure 5.2). The opposing piece is said to be 'captured', and removed from the board.

All captures are forced, and should be taken immediately when they arise. Should two or more captures be possible by separate friendly pieces, a player is allowed to finish a single capture this turn – irrespective of the possible higher value of the parallel available capture.

Multiple captures (a 'double jump') in the same turn are also forced, with the restriction that it should be completed by the same piece (see figure 5.2) and not involve a standard directional movement (*i.e.* it should originate from a forced capture, and immediately end after the last capture – see figure 5.2).

Should a piece reach the very last row on the opponent's side of the board without being captured, the piece is 'crowned' (the piece-name also changes from a 'man' to a 'king' and is deemed of higher value – see figure 5.2). A king is allowed to move in both a forward and backward diagonal direction as from the next turn, while still adhering to the basic rules for capturing. A man is allowed to capture a king, and vice versa. Multiple captures that result in a piece being crowned midway through the move is halted as soon as the piece reaches the back row. The newly crowned piece is allowed to complete any immediately available captures in the next round only.

Human matches have no drawing rule, and a draw is supposed to be offered by an opponent. Repeating the same three moves in succession is not illegal in Checkers (as it is in Chess), and can not be regarded as a measure for losing or drawing the game. For the experimental work in this study, a game is considered a draw if it exceeds 100 moves – a scheme followed by Chellapilla and Fogel [53] [27] [26].

5.2.1 Scoring structure

The scoring structure for Checkers in tournament play is similar to that of Chess, assigning +1 for a win, $+\frac{1}{2}$ each for a draw, and 0 for a loss. Due to the use of coevolution as the principal training method, a value of +1 is assigned for a win, 0 for a draw and -2 for a loss – identical to the previous work on Tic-Tac-Toe, as well as work by Chellapilla and Fogel [53] [27] [26]. The larger negative value of -2 for a loss, as opposed to only receiving +1 for a win, is justified to clearly punish the losing player for overall weak board state evaluations.

Once again, no variants to the standard scoring structure – such as the doubling cube found in Backgammon – are used for the Checkers simulations in this study.

5.3 Choosing an opponent

In contrast to the previous chapter on Tic-Tac-Toe, Checkers is not a 'solved' game, and no perfect player is available for either benchmarking or training. Initial work by Chellapilla and Fogel [53] used human players as training opponents, but quickly reverted to computer opponents after the significant drawback in time was observed. Chellapilla and Fogel later made use of an on-line Checkers 'game portal' to get a rough estimation of the best evolved agent's playing strength – albeit with some complications induced by the dark side of human nature (specifically sore losers).

Training a game playing agent over thousands and even millions of games is impractical with a human opponent. Apart from the obvious time drawbacks, the previously mentioned lapses in concentration and eventual fatigue play a major role in playing performance.

Computer-based players – specifically random-moving players – can consistently play at the same level, without repeating the same playing strategy. In chapter 3, table 3.2 lists the benchmark performance for two random-moving players competing in 1 million games, using the Rand3 pseudo-random number generator of Knuth [105]. Random moving players will only be used to benchmark the eventual best evolved player (as explained in chapter 3), while coevolutionary training will allow similarly constructed agents to compete in a randomly selected tournament scheme. Several advanced evaluation functions, such as a piece count-based player and an extensive hand-crafted evaluation function ('simple checkers' - provided by Martin Fierz [51], author of the well-known Cake Sans Saucy and Cake++ Checkers games), will play an important role in assessing intelligence in chapter 7.

5.4 Choosing a PSO architecture

The PSO neighbourhood structures originally introduced in section 2.4.1, and analysed in section 4.6 on Tic-Tac-Toe, is once again analysed to test the structures' performance in a more difficult problem domain. The GBest, LBest and Von Neumann architectures will be alternately selected and compared using a default set of PSO parameters (discussed below). The Messerschmidt *et al.* performance measure introduced in section 3.3.3 and utilised in the previous chapter to analyse Tic-Tac-Toe performance, is replaced by the Franken performance measure – introduced in section 3.3.4, due to the need for proper handling of drawn games as a sign of intelligence.

5.4.1 Parameter selection

The PSO parameters listed in section 2.4.2 are once again set inside a default range that will cause convergent behaviour, as illustrated by the equations originally derived by Van den Bergh and listed in section 2.4.2 for choosing the cognitive (c_1) and social (c_2) acceleration constants, as well as the exclusion of VMax. Convergence on possibly optimal solutions will enable the cross-architecture comparison required as part of the experimental work in this study.

More specifically, the cognitive (c_1) and social (c_2) acceleration constants are set to 1.0 each. The inertia weight too, is set to 1.0. The maximum velocity for each particle is not restricted. An 'architecture performance matrix' is once again constructed for each neighbourhood structure, measuring the change in performance over an increase in swarm size, and an increase in hidden layer size. Each performance matrix is represented as a heightmap for improved visual comparison and analysis.

The reader is once again reminded that due to the need of a personal best comparison in the PSO algorithm, the personal best solution of each particle is added as an additional player in the competition. The personal best particles remain unaffected by normal velocity updates (do not change), and only gets replaced if they are outperformed. The swarm size numbers quoted in the experimental configurations only represent half the true swarm size (*i.e.* excluding personal best particles).

A detailed analysis of particular PSO parameter influence is conducted in the next chapter, so no 'tweaking' on parameters are done to improve performance for this round of experiments.

5.5 Experimental procedure

The aim of the following experimental work is once again two-fold. Firstly, the coevolutionary-based training algorithm has proven to work well in the computationally modest problem space of Tic-Tac-Toe. The experiments now aim to judge its ability to scale to larger problem spaces, such as the one provided by Checkers.

Secondly, some interesting observations with regard to neighbourhood structures were made in the previous chapter, with the introduction of the Von Neumann structure outperforming both the LBest and GBest structures. It should be interesting to see if the trend repeats itself in this significantly larger and more difficult problem domain.

5.5.1 Training algorithm

The final training algorithm listed in section 3.5 is once again applied to this game learning problem, with the only change being the use of the Franken performance measure for primary performance comparison between the different PSO neighbourhood structures.

To quickly recap, each particle consists of an n-dimensional weight vector that corresponds to the weights of a neural network. Each particle can be seen as a single game playing agent in a population of players, with the substitution of particle information (weight values) into a neural network framework resulting in unique board evaluations for that specific player. Board state evaluations are limited to a single ply-depth only, with the reasoning behind this decision already discussed in section 3.4.

The use of coevolution to drive the training process, should result in more intelligent players emerging over time. The best individual after 500 epochs of evolution is benchmarked against a random-moving player.

5.5.2 PSO configuration

In summary, the PSO parameters are configured as follows: $c_1 = c_2 = 1.0$, inertia $= 1.0$, maximum velocity is not restricted. Each of the three PSO neighbourhood information sharing structures are analysed in turn, with swarm sizes ranging from 5 to 50 particles in the case of the GBest and LBest architectures, and between 15 and 50 particles for Von Neumann. The LBest structure makes use of an inclusive neighbourhood size of 5 particles.

5.5.3 Neural network configuration

In order for the agent to intelligently evaluate a board state, the board should be presented to each agent's neural network. Figure 5.1 illustrated the specific numbering of the standard Checkers game board, starting in the lower double-corner on the side of the board playing red, and finishing on the opposing player's opposite corner. Each board therefore is represented by at least 32 features – one for each playable square of the game board.

As mentioned in the rules of the game, a distinction is made between the value of a 'man' and a 'king', with a 'king' being more valuable. Chellapilla and Fogel [53] [27] [26] evolved the value for 'king'-pieces, resulting in a definite change in playing behaviour on the part of the players, as some players would recognise the importance of crowning pieces, while other would focus on restricting the opponent's playing behaviour using normal 'men'.

The 'my-piece vs opponent-piece' inverse board state representation introduced in section 4.5.3 on Tic-Tac-Toe evaluations is once again applied to Checkers. A friendly 'man' is represented as 0.75, and a friendly 'king' is represented as 1.0. An open space is represented by 0.5, with the intuitive reverse ordering for the opponent pieces – 0.25 for opponent men and 0.0 for an opponent 'king'. Alternate piece value representations are analysed in more detail in section 6.3.1 to possibly improve performance.

A 'sliding-window' is used by Chellapilla and Fogel to create various resolutions of the board state, allowing for more spatial information to be input to the network. The window would be initially sized as big as the complete board state (8-by-8), where after it decreases by a single square in both rows and columns (7-by-7, 6-by-6 etc.) and moves across the board to still cover all the possible playing squares. The result of each window is concatenated into a much larger numerical board state evaluation, and provided to the network for evaluation. The effect of using a sliding window as input to the neural network is compared in the following chapter on specific performance enhancements, along with a custom oriented window position. For the experimental work in this study, only the standard 32 playable board squares are used as input to the neural network.

The neural network implemented by Chellapilla and Fogel also made use of an additional hidden layer to (in the authors' words) improve spatial analysis on the board. In order to reduce the complexity of the neural network structure, allowing for faster board-state evaluations, only a single hidden layer (with varying size) is used.

Lastly, Chellapilla and Fogel include an additional bias to the network by providing the piece-count advantage of the current board-state as an additional input feature, fed directly into the output layer. The performance of a piece-count player is surprisingly good, and its inclusion into any neural network structure will introduce significant 'human knowledge' about the game. The inclusion of the piece-count as an additional input is justified by the authors due to the fact that most humans will intuitively recognise a superior local piece-count to be an advantage, even without a clear understanding of the game rules.

The neural network's performance against a piece-count player is analysed in more detail in chapter 7, along with other more intelligent evaluation functions. The experimental work in this study never makes use of the piece-count as an additional input feature.

5.5.4 Setting the benchmark

Table 3.2 lists the performance of two random-moving players competing in 1 million games. The Messerschmidt *et al.* performance measure directly relies on these probabilities to indicate superior playing behaviour – only taking into account wins and losses.

As explained in section 3.3.2, for more complex games (such as Checkers) a draw is often an indication of intelligent playing behaviour, and should be considered as well. The Franken performance measure described in section 3.3.4 allows for the inclusion of draws, and is applied to measure performance in the following experimental work. For all the performance matrices, the Messerschmidt *et al.* performance value (M-value) that makes use of the substituted probabilities for Checkers is listed along with the newly computed Franken performance (F-value) for completeness.

For all experimental results listed in this chapter, a Franken performance value larger than 50 indicates an increase in intelligent playing behaviour.

5.5.5 Statistical soundness

Confidence intervals are once again calculated with regard to the Messerschmidt *et al.* performance values, as clearly described in section 3.3.3. A confidence coefficient (alpha value) of 0.9 is used, and the required covariance and standard deviation values are computed.

5.6 Experimental results

The following experimental results list the various PSO neighbourhood information sharing structures' performance values, after benchmarking each particular configuration against a random-moving player for 150000 games as player one, and 150000 games as player two. Section

5.6.1 provides the detailed performance matrices for each neighbourhood structure, listing both the M and F performance values for a change in swarm size versus a change in hidden nodes.

The performance matrices are converted into heightmaps for further visual analysis in section 5.6.2, after which the specific trends relating to an increase in swarm size and increase in hidden layer size to playing performance is discussed in sections 5.6.4 and 5.6.3 respectively.

5.6.1 Architecture comparison matrices

Tables 5.1, 5.2 and 5.3 list the Franken (F) and Messerschmidt *et al.* (M) performance values (along with a confidence value) for the varying simulation configurations for each of the PSO neighbourhood structures – standard GBest, LBest and Von Neumann respectively. Each matrix illustrates the performance for a change in swarm size (increasing configurations from top to bottom) and a change in hidden layer size (increasing configurations from left to right).

Once again, a couple of interesting observations can be made before making use of the visual representation for an overall performance comparison. The generally accepted research finding that LBest outperforms GBest on most problem areas – as also illustrated in the Tic-Tac-Toe experiments – is once again repeated in the Checkers results.

The overall best GBest performer scored 62.3292 using the Franken performance measurement, with a swarm size of 40 particles and 5 hidden nodes. Compared to the location of the best GBest performer in the Tic-Tac-Toe results, it seems that the GBest algorithm still prefers the use of an increased swarm size.

The overall best LBest performer improved on the GBest score (as expected) with a highest value of 62.6382 – making use of a swarm size of 50 particles and 7 hidden nodes. Compared to the Tic-Tac-Toe results, it seems as if LBest has shifted slightly in its optimal configuration, using a larger swarm size, yet fewer hidden nodes to solve the more difficult problem.

The Von Neumann structure once again outperforms both the GBest and LBest information sharing structures, scoring 62.7935 with a configuration consisting of a swarm size of 50 particles and 7 hidden nodes. The increase in swarm size for both the LBest and Von Neumann structures can be attributed to the fact that a larger number of particles may be required to adequately cover the more complex search space introduced by Checkers.

The previous experimental results on Tic-Tac-Toe showed that LBest was able to compete closely with the Von Neumann structure. This observation is repeated in the Checkers results, even though the performance difference between the two structures' best particles is itself very small.

When computing an overall average performance value for all configurations in each performance matrix, the observations made with Tic-Tac-Toe is repeated once again. The average

Table 5.1: Architecture performance matrix - standard GBest. Larger M and F-values preferred.

	Hidden Units							
Particles	3	5	7	10	15	25	45	Averages
5 (F)	59.07150	57.99867	59.30317	59.48517	56.72700	58.23850	57.89400	58.38829
(M)	2.20000	1.97000	4.40000	3.66000	-1.80000	1.70000	1.54000	1.95286
	± 0.00609	± 0.00613	± 0.00610	± 0.00608	± 0.00618	± 0.00615	± 0.00616	± 0.00613
10 (F)	60.21550	59.71300	59.48600	58.91783	59.34783	58.10000	60.05717	59.40533
(M)	4.84000	4.89000	3.08000	2.74000	4.31000	1.04000	6.50000	3.91429
	± 0.00609	± 0.00610	± 0.00612	± 0.00613	± 0.00611	± 0.00616	± 0.00610	± 0.00612
15 (F)	58.47583	59.99500	60.09183	59.48600	58.66650	59.62033	59.98667	59.47460
(M)	3.83000	5.52000	5.08000	4.56000	1.64000	5.56000	5.94000	4.59000
	± 0.00610	± 0.00608	± 0.00609	± 0.00610	± 0.00614	± 0.00612	± 0.00610	± 0.00610
20 (F)	60.43400	59.65467	58.24433	59.09633	59.31433	57.85833	58.38467	58.99810
(M)	6.80000	5.05000	1.86000	2.32000	4.21000	1.01000	2.07000	3.33143
	± 0.00607	± 0.00610	± 0.00615	± 0.00613	± 0.00611	± 0.00616	± 0.00614	± 0.00612
25 (F)	59.20617	59.71583	60.43083	59.24317	60.13217	59.03717	58.62533	59.48438
(M)	3.93000	4.03000	7.42000	4.58000	6.58000	3.97000	3.20000	4.81571
	± 0.00607	± 0.00610	± 0.00609	± 0.00610	± 0.00609	± 0.00612	± 0.00614	± 0.00610
30 (F)	59.16283	59.08300	60.02500	57.54567	57.57767	57.73800	58.86617	58.57119
(M)	3.71000	3.14000	4.71000	0.30000	0.21000	1.20000	2.81000	2.29714
	± 0.00612	± 0.00612	± 0.00607	± 0.00615	± 0.00615	± 0.00616	± 0.00614	± 0.00613
35 (F)	59.36033	58.73350	59.30950	59.60667	59.52833	58.82267	60.55783	59.41698
(M)	2.86000	2.59000	3.74000	4.81000	4.64000	3.09000	6.36000	4.01286
	± 0.00611	± 0.00611	± 0.00611	± 0.00612	± 0.00612	± 0.00613	± 0.00606	± 0.00611
40 (F)	58.44583	62.32917	58.48617	57.63150	59.73350	57.95317	58.15433	58.96195
(M)	2.47000	10.74000	2.47000	0.66000	4.65000	0.63000	1.70000	3.33143
	± 0.00606	± 0.00601	± 0.00608	± 0.00614	± 0.00611	± 0.00617	± 0.00615	± 0.00610
45 (F)	58.72867	58.96950	58.92700	58.18833	58.46533	58.01367	58.78817	58.58295
(M)	3.53000	2.65000	4.99000	1.02000	2.16000	0.61000	3.18000	2.59143
	± 0.00611	± 0.00606	± 0.00606	± 0.00612	± 0.00614	± 0.00617	± 0.00614	± 0.00612
50 (F)	58.95217	57.72817	59.46533	59.39617	58.63617	56.04917	59.85467	58.58312
(M)	1.47000	2.06000	3.51000	4.94000	2.52000	-2.20000	5.10000	2.48571
	± 0.00608	± 0.00612	± 0.00611	± 0.00609	± 0.00613	± 0.00620	± 0.00612	± 0.00612
Average F	59.20528	59.39205	59.37692	58.85968	58.81288	58.14310	59.11690	
Average M	3.56400	4.26400	4.12600	2.95900	2.91200	1.66100	3.84000	
	± 0.00609	± 0.00609	± 0.00610	± 0.00612	± 0.00613	± 0.00615	± 0.00613	

overall best performing GBest score is 58.99 – the worst performer among the three structures once more. The overall average best performing LBest score is 59.78, closely followed by the best performing Von Neumann average score of 60.30.

A first glance at the specific performance values (both M and F) show that the players are not performing all too well against the benchmark random moving player. The Messerschmidt *et al.* performance measure shows only a slight increase over the minimum possible M score of 0.0 for all configurations (an M score larger than 0.0 assumes positive learning occurred). The Franken performance measure states a result of 50.0 for a random-moving player (in this case an F score larger than 50.0 assumes positive learning took place). An overall best value of 62.7935 by Von Neumann does not seem to be such a large improvement – considering the opponent. These disconcerting observations lead to a more in-depth analysis of PSO parameter influence, discussed in more detail in the following chapter.

Table 5.2: Architecture performance matrix - Local Best. Larger M and F-values preferred.

	Hidden Units							
Particles	3	5	7	10	15	25	45	Averages
5 (F)	60.17183	60.53133	59.40717	57.27417	61.30850	59.00967	58.83367	59.50519
(M)	7.12000	8.12000	3.76000	0.34000	8.28000	2.77000	2.39000	4.68286
	± 0.00607	± 0.00605	± 0.00612	± 0.00617	± 0.00606	± 0.00612	± 0.00614	± 0.00610
10 (F)	61.61900	61.76900	59.17900	59.11317	59.37133	58.92567	58.94250	59.84567
(M)	9.16000	9.71000	4.87000	4.42000	5.02000	4.14000	3.26000	5.79714
	± 0.00604	± 0.00603	± 0.00611	± 0.00612	± 0.00610	± 0.00612	± 0.00614	± 0.00610
15 (F)	58.16717	60.40867	60.31167	60.73383	58.96450	59.02917	59.62550	59.60579
(M)	2.96000	8.01000	7.07000	8.40000	4.65000	3.67000	4.89000	5.66429
	± 0.00604	± 0.00604	± 0.00608	± 0.00606	± 0.00610	± 0.00614	± 0.00611	± 0.00608
20 (F)	59.81083	58.79000	58.98167	59.31833	59.68300	59.75517	59.07900	59.34543
(M)	5.25000	3.47000	3.29000	4.08000	4.78000	5.76000	4.37000	4.42857
	± 0.00606	± 0.00611	± 0.00610	± 0.00609	± 0.00610	± 0.00610	± 0.00612	± 0.00610
25 (F)	57.48367	61.41567	60.69467	57.46650	59.13300	59.85467	59.97933	59.43250
(M)	0.72000	9.22000	6.51000	1.13000	4.46000	5.10000	5.76000	4.70000
	± 0.00606	± 0.00605	± 0.00608	± 0.00615	± 0.00611	± 0.00609	± 0.00611	± 0.00609
30 (F)	59.41883	61.29533	60.33167	60.56767	59.32100	59.70600	59.16700	59.97250
(M)	4.92000	10.58000	6.29000	7.83000	4.27000	6.02000	3.05000	6.13714
	± 0.00607	± 0.00603	± 0.00607	± 0.00607	± 0.00612	± 0.00609	± 0.00614	± 0.00609
35 (F)	58.02050	59.50083	58.15983	58.63717	60.60000	59.45383	58.84500	59.03102
(M)	0.86000	7.21000	1.96000	3.14000	7.62000	5.22000	2.41000	4.06000
	± 0.00614	± 0.00606	± 0.00614	± 0.00612	± 0.00608	± 0.00612	± 0.00615	± 0.00611
40 (F)	62.58433	61.11300	60.71917	60.05300	58.35017	59.02583	59.08950	60.13357
(M)	11.01000	9.20000	8.73000	5.85000	3.26000	3.88000	3.38000	6.47286
	± 0.00597	± 0.00600	± 0.00606	± 0.00608	± 0.00614	± 0.00613	± 0.00614	± 0.00607
45 (F)	61.27900	59.19617	57.98300	58.77150	59.62917	60.66350	61.09583	59.80260
(M)	8.55000	4.67000	2.52000	3.56000	4.21000	7.63000	7.86000	5.57143
	± 0.00603	± 0.00609	± 0.00613	± 0.00612	± 0.00611	± 0.00608	± 0.00608	± 0.00609
50 (F)	60.57100	60.38683	62.63817	60.87517	61.31550	60.37950	61.63100	61.11388
(M)	9.01000	6.75000	10.78000	9.03000	8.27000	7.06000	9.14000	8.57714
	± 0.00603	± 0.00607	± 0.00605	± 0.00600	± 0.00607	± 0.00609	± 0.00607	± 0.00605
Average F	59.91262	60.44068	59.84060	59.28105	59.76762	59.58030	59.62883	
Average M	5.95600	7.69400	5.57800	4.77800	5.48200	5.12500	4.65100	
	± 0.00605	± 0.00605	± 0.00609	± 0.00610	± 0.00610	± 0.00611	± 0.00612	

5.6.2 Surface plots

In order to ease an overall performance comparison of the various neighbourhood structures, each performance matrix is visualised as a heightmap, with each configuration's F-value representing a height on a three dimensional surface. Each structure's surface is plotted with a unique identifying colour, and the Von Neumann surface's opacity is decreased to give an insight into lower-lying surfaces. Higher lying areas on the surfaces indicate increased playing performance, with the overall superior structure for a specific configuration showing its dominance by visually overlapping the other two structures' surfaces.

From visual inspection, the Von Neumann architecture seems to completely dominate the various configurations, with LBest being able to convincingly outperform Von Neumann in a selected few mid-range configuration sets. GBest is once again the worst performer, only making slight attempts to improve on LBest's performance – and outperforming both Von

Table 5.3: Architecture performance matrix - Von Neumann. Larger M and F-values preferred.

	Hidden Units							
Particles	3	5	7	10	15	25	45	Averages
15 (F)	60.42533	60.92733	59.16450	61.97550	60.75067	58.46417	59.26983	60.13962
(M)	6.63000	9.43000	4.39000	9.56000	7.77000	3.53000	3.71000	6.43143
	± 0.00605	± 0.00602	± 0.00610	± 0.00604	± 0.00608	± 0.00614	± 0.00613	± 0.00608
20 (F)	60.47567	60.19417	59.14200	59.24383	60.84083	60.41567	58.68867	59.85726
(M)	5.89000	6.43000	5.22000	5.02000	7.15000	6.46000	3.43000	5.65714
	± 0.00608	± 0.00608	± 0.00611	± 0.00611	± 0.00609	± 0.00609	± 0.00614	± 0.00610
25 (F)	58.84867	61.85950	60.51217	59.28617	59.24067	60.19317	58.46250	59.77183
(M)	5.42000	9.84000	6.26000	5.10000	5.37000	6.00000	2.03000	5.71714
	± 0.00606	± 0.00602	± 0.00608	± 0.00610	± 0.00609	± 0.00610	± 0.00615	± 0.00609
30 (F)	58.77800	61.99633	61.50067	58.42017	58.04900	59.76217	61.44633	59.99324
(M)	3.28000	11.04000	8.21000	2.12000	2.25000	5.50000	9.01000	5.91571
	± 0.00608	± 0.00602	± 0.00606	± 0.00614	± 0.00616	± 0.00612	± 0.00607	± 0.00609
35 (F)	59.77617	61.78300	60.81017	61.19450	60.78717	59.28067	58.97683	60.37264
(M)	5.93000	9.02000	8.67000	10.66000	8.19000	4.68000	4.18000	7.33286
	± 0.00609	± 0.00604	± 0.00607	± 0.00604	± 0.00608	± 0.00612	± 0.00613	± 0.00608
40 (F)	61.66283	62.61367	60.72750	59.45333	61.01300	59.71400	61.82367	61.00114
(M)	9.52000	12.26000	8.39000	4.37000	9.29000	5.57000	10.30000	8.52857
	± 0.00600	± 0.00600	± 0.00601	± 0.00612	± 0.00606	± 0.00611	± 0.00605	± 0.00605
45 (F)	61.21800	60.26700	59.97000	60.01933	60.80200	59.88700	59.59433	60.25110
(M)	8.61000	7.47000	5.30000	5.95000	7.24000	6.42000	5.91000	6.70000
	± 0.00602	± 0.00607	± 0.00606	± 0.00611	± 0.00608	± 0.00609	± 0.00611	± 0.00608
50 (F)	60.06550	60.09033	62.79350	58.56183	61.82400	62.23167	61.63100	61.02826
(M)	6.39000	5.45000	11.34000	2.92000	10.48000	11.06000	9.14000	8.11143
	± 0.00610	± 0.00609	± 0.00602	± 0.00614	± 0.00605	± 0.00604	± 0.00607	± 0.00607
Average F	60.15627	61.21642	60.57756	59.76933	60.41342	59.99356	59.98665	
Average M	6.45875	8.86750	7.22250	5.71250	7.21750	6.15250	5.96375	
	± 0.00606	± 0.00604	± 0.00606	± 0.00610	± 0.00609	± 0.00610	± 0.00611	

Neumann and LBest in a small region of the map only.

The Von Neumann structure seems to take advantage of larger swarms and smaller hidden layer sizes, performing weaker in smaller swarms with mid-range hidden layer sizes. A visual bulge on the surfaces towards the edge of the map (swarm size of 50 particles, 7 hidden nodes) indicate the location of the best performers for Von Neumann and LBest, with GBest's best performer located close by (40 particles, 5 hidden nodes).

The relatively weak overall performance values is clearly indicated by the performance scale on the left not exceeding 65.0. The following sections will look more closely at the overall impact on performance caused by an increase in swarm size, and an increase in hidden nodes.

5.6.3 Increase in hidden nodes

Figure 5.4 depicts the plot of an average for each column in the respective performance matrices, indicating the change in performance over an increase in hidden nodes. The Von Neumann structure clearly outperforms the LBest and GBest structures – as expected – showing the already mentioned preference for smaller hidden layer sizes (the average peak performance occurring quite quickly). Performance seems to stabilise on larger hidden layer sizes, with the

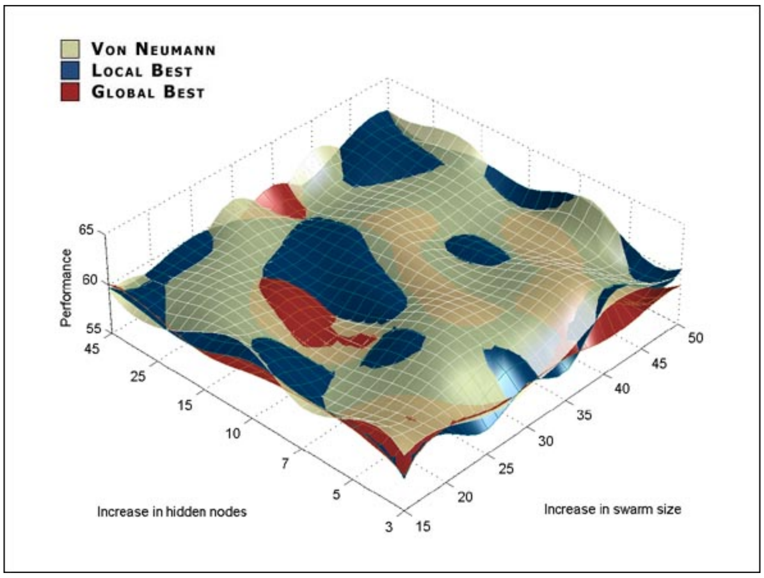
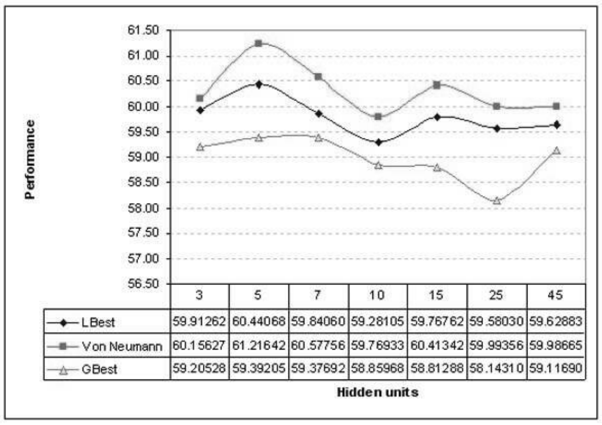


Figure 5.3: Surface plot for PSO structure comparison in Checkers.



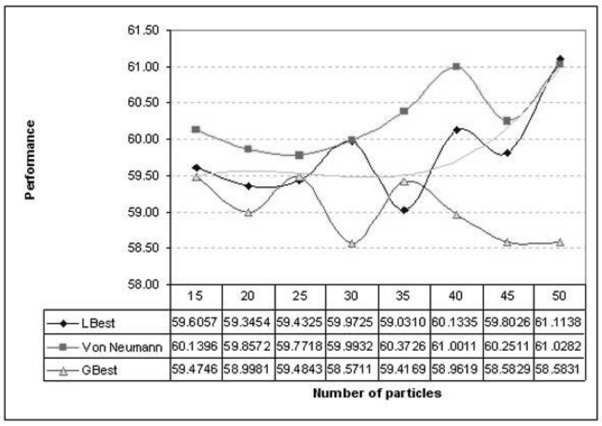
	3	5	7	10	15	25	45
LBest	59.91262	60.44068	59.84060	59.28105	59.76762	59.58030	59.62883
Von Neumann	60.15627	61.21642	60.57756	59.76933	60.41342	59.99356	59.98665
GBest	59.20528	59.39205	59.37692	58.85968	58.81288	58.14310	59.11690

Figure 5.4: Increase in hidden nodes during Checkers simulations.

graph showing little deviation after 15 hidden nodes.

The LBest structure almost mimics the Von Neumann performance behaviour, albeit on a slightly weaker level. This reinforces previous observations on the close relationship between LBest and Von Neumann. The same trend of an affinity for smaller hidden layer sizes is repeated, with performance once again stabilising on larger hidden layer sizes indicated by the little deviation present on the graph after 15 hidden nodes.

The overall inferiority of the GBest structure is clear from the graph, with a slight possibility of an increase in performance with significantly larger hidden layer sizes. The possibility of an increased performance beyond 45 hidden nodes was not investigated further, due to the already



	15	20	25	30	35	40	45	50
◆ LBest	59.6057	59.3454	59.4325	59.9725	59.0310	60.1335	59.8026	61.1138
■ Von Neumann	60.1396	59.8572	59.7718	59.9932	60.3726	61.0011	60.2511	61.0282
△ GBest	59.4746	58.9981	59.4843	58.5711	59.4169	58.9619	58.5829	58.5831

Figure 5.5: Increase in swarm size during Checkers simulations.

bad track-record of the GBest structure, as well as the significant computational time required by any larger experimental configurations.

It is worthwhile to note that at least positive learning occurred, and the players were able to attain F-values above 50.0 – indicative of intelligent playing behaviour.

5.6.4 Increase in swarm size

The impact of an increase in swarm size is depicted in figure 5.5, plotting the average Franken performance values for each row in the individual information sharing structure's performance matrix. The Von Neumann architecture dominates yet again, with the LBest architecture eventually equalling its rival structure's performance. The GBest structure remains inferior throughout.

The Von Neumann and LBest neighbourhood structures seem to show an overall gain in performance from an increase in swarm size (also indicated by the faint trend-line plotted over the LBest performance graph). There does however exist a definite sign of oscillatory behaviour, perhaps indicating the sensitivity by each neighbourhood structure to a slight change in neighbourhood size.

As promising as the sudden surge at the end of the plot seems, an increase beyond 50 particles was once again not investigated for computational reasons. The following chapter will try to isolate the specific PSO-related parameter influences on performance, without adding an additional computational overhead to the simulations.

5.7 Conclusion

This chapter continued the experimental analysis from the previous chapter on Tic-Tac-Toe, by exposing the coevolutionary PSO-based learning algorithm to a much more complex problem – the game of Checkers. An overview of the specific game rules that were implemented for the experimental work were followed by a specification of PSO parameter selection, and various ways of representing a board state before feeding it to the neural network.

Experimental work compared the performance of the various PSO information sharing neighbourhood structures to evolve an intelligent Checkers playing agent. The performance matrices, surface plots and graphs which illustrates the effect of an increase in hidden nodes and swarm sizes all confirm that the Von Neumann structure still dominates with superior performance.

The LBest neighbourhood structure has narrowed the performance gap between it and the Von Neumann structure considerably. GBest remains an inferior neighbourhood structure throughout.

The overall results remain disappointing, and more in-depth analysis of specific PSO parameters in the following chapter leads to a clearer understanding of the inner workings of the training algorithm in the game learning domain.

CHAPTER 6

Investigating performance factors

"Never discourage anyone... who continually makes progress, no matter how slow".
- Plato (427 BC - 347 BC)

This chapter aims to conduct a detailed investigation into the possible causes for the weak playing performance of the PSO-evolved Checkers players up to date. The influence of all known PSO parameters are experimentally examined, followed by a look at neural network related performance issues, and various coevolutionary algorithm extensions. In the process, a particle dispersement operator and a coevolutionary performance measure, based on a scoring system inspired by Formula One Grand Prix, are introduced. The chapter concludes with an analysis of stricter training criteria.

6.1 Introduction

The previous chapter's results for evolved Checkers players indicated a slight increase in intelligent playing behaviour as compared to a true random player. However, the results did not sufficiently show that the players were able to completely dominate the random player, as a highly developed Checkers player should be able to do. Instead of focusing on the expansion of the swarm sizes or neural network hidden layer sizes beyond the original test ranges, a study of the core PSO parameters, neural network settings and coevolutionary training techniques is instead conducted.

Unless stated otherwise, all the experimental work described in this chapter makes use of a base configuration of 15 particles and 3 hidden nodes, utilising the Von Neumann information sharing structure, and evolution over 500 epochs is assumed. The reason for these specific choices are predominantly due to the speed impact associated with a large number of simulations, which implies that small experimental configurations are preferred. The Von Neumann information sharing structure proved to outperform both the standard GBest and LBest structures in the previous two chapters, and is therefore the structure of choice. Based on the initial Checkers performance matrices given in section 5.3, the selection of 15 particles and 3 hidden nodes is justified due to the above average performance levels and overall space for improvement that exists. The overall best performing particle is benchmarked against a random-moving player for 150000 games as player one, and 150000 games as player two.

Section 6.2 covers the experimental analysis of the core PSO parameters, including the influence of the maximum velocity (VMax), the cognitive (c_1) and social (c_2) acceleration constants, the inertia weight, the LBest neighbourhood size, an increase in swarm size and an increase in hidden layer size.

A series of variations to presenting the board as input to the neural network is analysed in section 6.3, including different approaches to piece-value calculation and the use of different 'windowing' techniques to provide more spatial information to the network.

The standard coevolutionary approach to learning that has been followed up to now is extended to include a novel set of new methods, loosely based on Formula One Grand Prix [48] in section 6.4. The methods include a 'race' performance evaluator and particle dispersement operator. The new methods are compared to well-established coevolutionary constructs, including 'Hall-of-Fame' and random sampling.

The chapter concludes with a look at the behaviour of the PSO-training process under stricter conditions in section 6.5, including examining the influence of the drawing measure (100 moves), extending the training time and restricting the maximum move count.

Section 6.6 summarises some of the major experimental findings of this chapter.

6.2 Particle Swarm Parameters

The PSO algorithm and associated parameters were introduced in section 2.4 of the background chapter on computational intelligence techniques. A small example of the impact on playing performance due to 'tweaking' these parameter settings were given in section 4.6.6, as applied to Tic-Tac-Toe. The various parameter settings are thoroughly investigated in the following subsections to establish the impact each parameter may have on the playing performance of PSO-evolved Checkers players. The performance values listed throughout the chapter corre-

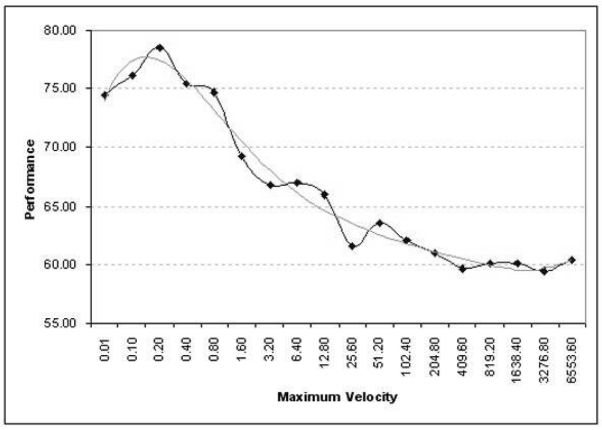


Figure 6.1: Influence of maximum velocity with trend-line.

spond to the computed Franken performance measure (F), as originally described in section 3.3.4.

6.2.1 Influence of maximum velocity (VMax)

During the initial experiments discussed in the previous chapter, no restriction was placed on the maximum allowed velocity. Closer analysis of the weight values contained by each particle showed an explosion in weight size roughly half-way through training. These very large weight values had an undesired effect on the neural network and saturated the sigmoid activation functions in use – ultimately resulting in weak playing performance.

A new set of experiments were conducted to test the impact of restricting the maximum velocity for all the particles in the swarm. The maximum velocity was non-linearly adjusted from 0.01 (strict velocity restriction) to 6553.6 (no effect). Figure 6.1 clearly shows a decline in performance as the restriction is relaxed (trend-line added to ease the observation of average performance). Small VMax values result in better performing agents, with scores increasing from the previous best of 61.842 to a much more respectable 78.44 using a VMax of 0.2.

The different methods of restricting the maximum velocity also have a significant impact on the performance, as illustrated in table 6.1. The constriction coefficient introduced by Clerc

Table 6.1: Performance of various methods to restrict the maximum velocity.

Method	No VMax	Strict Clamping	Scaling	Constriction
Performance (F-value)	61.84167	73.84900	70.70033	63.98267

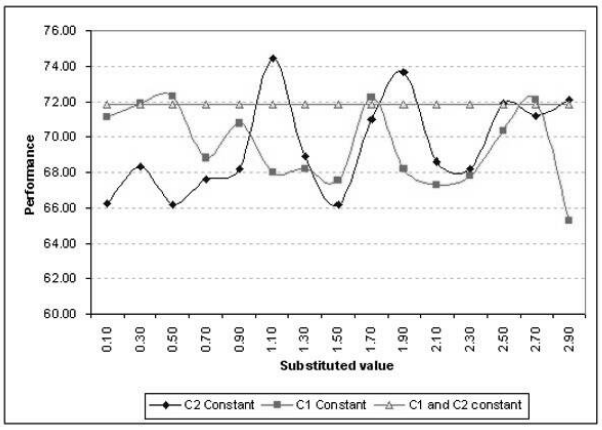


Figure 6.2: Influence of c_1 and c_2 parameters.

[31] [32] and mentioned in section 2.4.2 results in the weakest performance. Proportionally scaling the velocity vector to the new provided bounds – thereby still maintaining the velocity direction – achieves significantly better performance. However, strictly clamping the maximum velocity in all directions provides the best performance, even though conservation of the search direction is not guaranteed.

6.2.2 Influence of c1 and c2

The cognitive (c_1) and social (c_2) acceleration constants influence how much the particle's velocity update is affected by the personal best position found in the search space thus far, and the overall best position as defined by the PSO information sharing structure.

A set of experiments were conducted to test the effect of c_1 and c_2 on the performance of the game playing agent. Using the improvement of a small VMax as a new benchmark configuration, the default values for c_1 (1.0) and c_2 (1.0) resulted in a performance rating above 70.0.

The first set of experiments kept the c_1 value constant at 1.0, but adjusted the c_2 value between 0.1 and 2.9, at intervals of 0.2. The second set of experiments inverted this process, by keeping c_2 constant at 1.0, and adjusting c_1 within the same boundaries. As can be seen from figure 6.2, no significant improvement is made on the default score (represented by a straight line – for visual reference).

The c_1 and c_2 values were subsequently not adjusted.

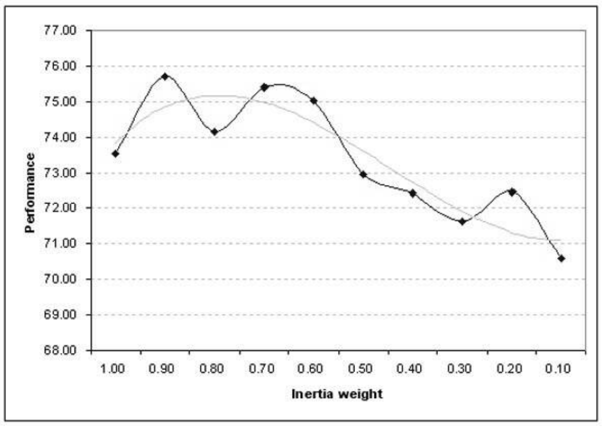


Figure 6.3: Influence of inertia.

6.2.3 Influence of inertia weight

The use of an inertia weight, as described in section 2.4.2, balances out the influence of local and global search capabilities of the swarm. Large inertia values facilitate global searches, while smaller inertia values facilitate local searches. For the experimental work in this study, a linearly decreasing inertia [123] was not used, but a static inertia experimentally determined. It is important to remember the underlying relationship between the inertia weight and the c_1 and c_2 acceleration constants to result in convergent behaviour, as defined by equation 2.7 in section 2.4.2. The following experimental test values were chosen correspondingly.

In order to locate the optimal inertia setting, experiments were conducted that statically reduce the inertia term from 1.0 to 0.1. The resultant performance graph is depicted in figure 6.3, clearly showing a decreasing performance as smaller inertia values restrict the swarm to only conduct a local search. An inertia value of 0.9 proved to work best for this particular swarm size and hidden node configuration.

6.2.4 Influence of LBest neighbourhood size

Up to now, an inclusive LBest neighbourhood size of 5 particles has been used to conduct the relevant experimental work for this study. The validity of this assumption was experimentally

Table 6.2: Performance of various LBest neighbourhood sizes.

Neighbourhood size	3	5	7	9	11	13
Performance (F-value)	76.833	76.842	73.643	70.330	71.505	65.632

Table 6.3: Architecture performance matrix - Updated standard GBest. Larger M and F-values are preferred.

Particles	Hidden Units							Averages
	3	5	7	10	15	25	45	
5 (F)	66.10633	69.56650	65.14367	65.94967	66.52750	63.12750	66.96350	66.19781
(M)	19.08800	28.47600	15.93000	19.57333	19.49800	12.10267	21.26333	19.41876
	± 0.00593	± 0.00575	± 0.00598	± 0.00592	± 0.00592	± 0.00603	± 0.00589	± 0.00592
10 (F)	72.46783	67.52800	67.89250	69.05750	68.15083	68.67917	68.62900	68.91498
(M)	32.96067	23.07800	21.11733	24.42267	24.48400	24.73200	25.98267	25.25390
	± 0.00566	± 0.00586	± 0.00589	± 0.00583	± 0.00583	± 0.00582	± 0.00580	± 0.00581
15 (F)	68.68933	68.75617	72.02100	72.22083	69.93017	69.85800	67.66433	69.87712
(M)	24.84667	26.49333	32.83200	33.82400	27.27933	27.30533	23.77067	28.05019
	± 0.00582	± 0.00579	± 0.00566	± 0.00564	± 0.00578	± 0.00578	± 0.00584	± 0.00576
20 (F)	73.51700	70.38650	71.85467	70.09017	68.35667	71.33300	71.51667	71.00781
(M)	36.89333	28.91400	33.15133	26.85533	23.73267	30.23533	31.59467	30.19667
	± 0.00557	± 0.00574	± 0.00565	± 0.00578	± 0.00584	± 0.00572	± 0.00569	± 0.00571
25 (F)	70.18050	77.03050	69.61833	68.36950	72.86900	67.01367	75.18517	71.46667
(M)	28.92400	46.32067	27.41000	23.56933	35.23467	20.66667	40.15400	31.75419
	± 0.00574	± 0.00533	± 0.00577	± 0.00585	± 0.00561	± 0.00590	± 0.00549	± 0.00567
30 (F)	74.27133	74.74700	72.04833	67.35350	71.90417	70.07183	78.00850	72.62924
(M)	38.68533	38.53867	32.55267	21.40533	32.78000	29.69533	49.38533	34.72038
	± 0.00552	± 0.00553	± 0.00567	± 0.00589	± 0.00566	± 0.00573	± 0.00524	± 0.00560
35 (F)	75.11533	74.84517	73.51400	74.27633	75.02833	71.03483	76.39067	74.31495
(M)	39.55733	41.83267	37.45667	39.36400	40.81200	29.29800	43.87600	38.88524
	± 0.00550	± 0.00545	± 0.00555	± 0.00551	± 0.00547	± 0.00573	± 0.00539	± 0.00552
40 (F)	71.89133	67.05517	73.46500	69.66483	78.68883	74.03117	70.69083	72.21245
(M)	30.82467	21.16733	35.24267	26.33733	47.66933	36.05133	30.91467	32.60105
	± 0.00570	± 0.00589	± 0.00561	± 0.00579	± 0.00529	± 0.00559	± 0.00570	± 0.00565
45 (F)	74.90067	75.88683	74.86533	72.02700	76.68700	71.58050	73.73800	74.24076
(M)	41.02400	42.61467	40.33733	32.34600	42.64600	32.90400	37.31400	38.45514
	± 0.00547	± 0.00543	± 0.00548	± 0.00567	± 0.00542	± 0.00566	± 0.00556	± 0.00553
50 (F)	72.54500	75.76850	73.53067	74.83467	70.60967	74.67733	71.99017	73.42229
(M)	34.56467	41.60400	35.65600	40.83667	29.49000	38.39933	32.87467	36.20362
	± 0.00562	± 0.00545	± 0.00560	± 0.00547	± 0.00573	± 0.00553	± 0.00566	± 0.00558
Average F	71.96847	72.15703	71.39535	70.38440	71.87522	70.14070	72.07768	
Average M	32.73687	33.90393	31.16860	28.85340	32.36260	28.13900	33.71300	
	± 0.005653	± 0.005621	± 0.005686	± 0.005735	± 0.005655	± 0.005749	± 0.005626	

tested, by increasing neighbourhood sizes from 3 particles to 13 particles in step sizes of 2. The results are depicted in table 6.2. A definite drop in performance is seen for neighbourhood sizes larger than 5 particles, immediately confirming the accepted theory that large LBest neighbourhoods approximate the performance of the GBest neighbourhood structure. The standard GBest structure has consistently shown to be the weakest of the three neighbourhood structures, and large neighbourhood sizes are subsequently avoided. The inclusive neighbourhood size remains set at 5 particles.

6.2.5 Updated neighbourhood structure comparison matrices

Taking the aforementioned parameter results into consideration, a new set of experiments were conducted to investigate the influence of an increased swarm size versus an increase in hidden layer size, for all three neighbourhood information sharing structures. VMax was restricted to

Table 6.4: Architecture performance matrix - Updated standard LBest. Larger M and F-values are preferred.

	Hidden Units							
Particles	3	5	7	10	15	25	45	Averages
5 (F)	65.91450	66.75150	63.48433	64.75767	62.61183	61.94600	66.91483	64.62581
(M)	18.04067	23.28867	12.41267	17.23667	10.74067	9.56667	17.07133	15.47962
	± 0.00594	± 0.00585	± 0.00603	± 0.00596	± 0.00605	± 0.00607	± 0.00596	± 0.00598
10 (F)	69.32417	69.56550	66.78167	70.60667	68.93500	66.36750	72.36100	69.13450
(M)	28.15000	26.69733	20.82200	30.35067	25.39267	20.90933	34.63533	26.70819
	± 0.00576	± 0.00579	± 0.00590	± 0.00571	± 0.00581	± 0.00589	± 0.00562	± 0.00578
15 (F)	73.97433	76.86050	75.00300	76.23917	76.38517	79.21417	77.93750	76.51626
(M)	36.17067	44.20133	39.03533	42.62733	40.39733	48.99200	47.38000	42.68629
	± 0.00558	± 0.00538	± 0.00552	± 0.00543	± 0.00548	± 0.00525	± 0.00530	± 0.00542
20 (F)	79.11733	76.35300	76.68300	78.98633	76.36883	78.81500	79.17867	77.92888
(M)	47.20400	42.97733	43.31800	48.45067	44.10600	47.05800	49.08600	46.02857
	± 0.00530	± 0.00542	± 0.00541	± 0.00526	± 0.00539	± 0.00530	± 0.00525	± 0.00533
25 (F)	78.68533	81.43967	77.00550	79.14917	75.93550	80.64050	77.14900	78.57210
(M)	47.52200	55.74867	44.25933	49.61067	41.16067	53.49667	43.40733	47.88648
	± 0.00529	± 0.00504	± 0.00538	± 0.00523	± 0.00546	± 0.00511	± 0.00540	± 0.00527
30 (F)	79.04300	82.66250	78.86217	81.07483	79.21200	81.13083	78.18250	80.02398
(M)	48.74200	57.77267	46.88600	54.05667	48.00333	53.33800	45.54733	50.62086
	± 0.00526	± 0.00497	± 0.00531	± 0.00509	± 0.00528	± 0.00512	± 0.00535	± 0.00520
35 (F)	80.89917	83.43150	81.61367	80.11333	81.08517	75.31450	82.44183	80.69988
(M)	53.77133	60.62067	54.97000	51.43000	55.26933	41.27867	57.43667	53.53952
	± 0.00510	± 0.00487	± 0.00507	± 0.00518	± 0.00506	± 0.00546	± 0.00498	± 0.00510
40 (F)	82.49650	80.15100	80.37150	79.43567	82.23800	81.85217	80.16600	80.95869
(M)	57.80933	51.46133	53.18133	48.99533	56.42467	55.64733	50.69133	53.45867
	± 0.00497	± 0.00518	± 0.00512	± 0.00525	± 0.00502	± 0.00504	± 0.00520	± 0.00511
45 (F)	79.64350	83.26300	79.74383	81.68783	80.61200	79.97117	78.60233	80.50338
(M)	50.81333	59.64200	50.64800	54.71400	51.31600	50.14200	47.07933	52.05067
	± 0.00520	± 0.00491	± 0.00520	± 0.00507	± 0.00518	± 0.00522	± 0.00530	± 0.00515
50 (F)	79.46400	81.33483	83.63583	81.07350	80.50183	81.21583	81.12033	81.19231
(M)	48.65667	53.79933	61.69933	53.85133	52.43467	53.12333	52.68667	53.75019
	± 0.00526	± 0.00510	± 0.00483	± 0.00510	± 0.00515	± 0.00512	± 0.00514	± 0.00510
Average F	76.85618	78.18130	76.31845	77.31242	76.38853	76.64677	77.40540	
Average M	43.68800	47.62093	42.72320	45.13233	42.52453	43.35520	44.50213	
	± 0.00537	± 0.00525	± 0.00538	± 0.00533	± 0.00539	± 0.00536	± 0.00535	

0.2, c_1 and c_2 set to 1.0 each, inertia equalled 0.9 and in the case of the LBest experiments an inclusive neighbourhood structure of 5 particles was used. The results are tabulated as Franken (F) and Messerschmidt *et al.* (M) performance values, with a corresponding confidence interval. Table 6.3 depicts the improved performance matrix for the GBest structure, followed by an improved matrix for LBest in table 6.4 as well as for Von Neumann in table 6.5.

Examining the best performing individual configurations only, it seems as if the LBest structure has surpassed the Von Neumann structure's dominance illustrated in previous chapters. The best LBest player scored 83.63583 with a swarm size of 50 particles and a neural network containing a hidden layer size of 7 nodes. This performance value constitutes a dramatic increase over the best performance achieved before optimisation, as mentioned in section 5.6.1. The previous overall best performance score was 62.7935, achieved by the Von Neumann structure using 50 particles and 7 hidden nodes.

Table 6.5: Architecture performance matrix - Updated Von Neumann. Larger M and F-values are preferred.

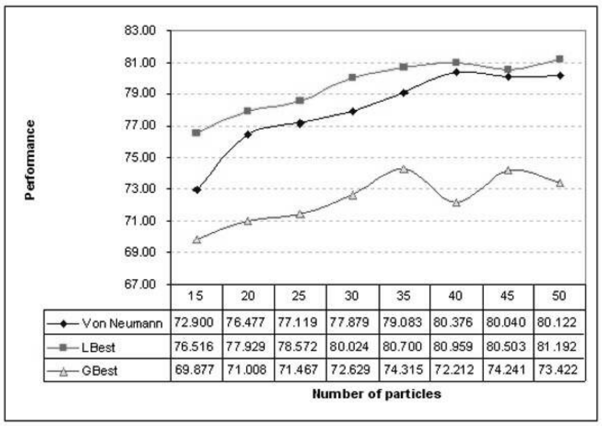
	Hidden Units							
Particles	3	5	7	10	15	25	45	Averages
15 (F)	75.95300	72.52350	73.59017	71.45667	72.86750	74.22300	69.68333	72.89960
(M)	40.91400	33.35933	35.13533	31.03000	32.80067	36.33533	28.30600	33.98295
	± 0.00547	± 0.00565	± 0.00561	± 0.00570	± 0.00566	± 0.00558	± 0.00576	± 0.00563
20 (F)	75.55967	77.35300	77.90983	75.52117	79.37950	78.03467	71.58267	76.47721
(M)	40.12333	45.29200	47.26800	40.71800	50.79867	47.10533	32.22267	43.36114
	± 0.00549	± 0.00535	± 0.00530	± 0.00547	± 0.00520	± 0.00530	± 0.00567	± 0.00540
25 (F)	78.49100	77.43033	76.62333	78.93833	76.52833	75.61333	76.21133	77.11943
(M)	48.29533	43.20333	41.83000	48.33467	44.52800	38.95400	41.19467	43.76286
	± 0.00527	± 0.00541	± 0.00545	± 0.00527	± 0.00537	± 0.00552	± 0.00546	± 0.00539
30 (F)	80.75833	78.20650	76.58350	76.15117	79.75383	78.01050	75.68633	77.87860
(M)	53.67467	46.41733	42.91067	42.68400	50.06600	47.18333	41.73200	46.38114
	± 0.00511	± 0.00532	± 0.00542	± 0.00542	± 0.00522	± 0.00530	± 0.00545	± 0.00532
35 (F)	79.40183	79.20450	78.70950	77.66483	75.65967	82.54383	80.39733	79.08307
(M)	48.39467	50.14800	47.75600	43.72400	41.51333	58.78333	52.95000	49.03848
	± 0.00527	± 0.00521	± 0.00528	± 0.00540	± 0.00545	± 0.00494	± 0.00513	± 0.00524
40 (F)	79.47700	81.67550	79.14467	79.32900	81.52617	82.88750	78.59083	80.37581
(M)	49.57733	55.73733	48.20867	48.15933	55.52667	58.53133	45.68867	51.63276
	± 0.00523	± 0.00504	± 0.00527	± 0.00527	± 0.00505	± 0.00495	± 0.00534	± 0.00516
45 (F)	77.14233	80.12483	80.60900	81.08250	79.23583	80.26700	81.81617	80.03967
(M)	42.25333	50.60400	52.86733	54.29133	47.64267	51.90333	55.19067	50.67895
	± 0.00543	± 0.00520	± 0.00513	± 0.00509	± 0.00529	± 0.00516	± 0.00506	± 0.00519
50 (F)	80.26183	80.83600	82.09100	77.57900	79.60783	79.03350	81.44800	80.12245
(M)	50.95600	51.91333	56.14667	46.69067	49.93733	47.71933	55.13667	51.21429
	± 0.00519	± 0.00516	± 0.00503	± 0.00531	± 0.00522	± 0.00529	± 0.00506	± 0.00518
Average F	78.38063	78.41927	78.15763	77.21533	78.06983	78.82667	76.92700	
Average M	46.77358	47.08433	46.51533	44.45400	46.60167	48.31442	44.05267	
	± 0.00531	± 0.00529	± 0.00531	± 0.00537	± 0.00531	± 0.00525	± 0.00537	

The optimised Von Neumann performance improves on its previous best score, but fails to maintain its dominance as overall best neighbourhood structure. The optimised best Von Neumann score of 82.88750, achieved using 40 particles and 25 hidden nodes, is however still within reach of the aforementioned overall optimised best score of LBest.

The GBest structure also manages to improve its previous performance levels, yet remains the worst performer among the three structures – only scoring 78.68883 with a swarm size of 40 particles and a neural network with 15 hidden nodes.

When computing overall averages for similar regions of the different performance matrices, the newly established performance trend continues. The LBest structure still outperforms both the Von Neumann and GBest structures, with an average performance of 79.5494. Von Neumann trails LBest with a performance of 77.9995, and GBest fails once more with 72.3964.

Before making use of a visual representation (such as a surface plot) of the performance matrices to analyse global trends, it is clear from quick inspection of the performance values that the use of the optimised PSO parameters have resulted in a significant performance increase for all three neighbourhood structures.



	15	20	25	30	35	40	45	50
Von Neumann	72.900	76.477	77.119	77.879	79.083	80.376	80.040	80.122
LBest	76.516	77.929	78.572	80.024	80.700	80.959	80.503	81.192
GBest	69.877	71.008	71.467	72.629	74.315	72.212	74.241	73.422

Figure 6.4: Increase in swarm size.

6.2.6 Increase in swarm size

In the previous chapter, an increase in swarm size was shown to benefit the Von Neumann and LBest information sharing structures. After applying the new PSO parameter values and calculating the performance matrices, the effect of an increase in swarm size can once again be studied. Figure 6.4 graphically depicts the average performance F-values, computed for each row of the relevant performance matrices and plotted in order of increasing swarm size.

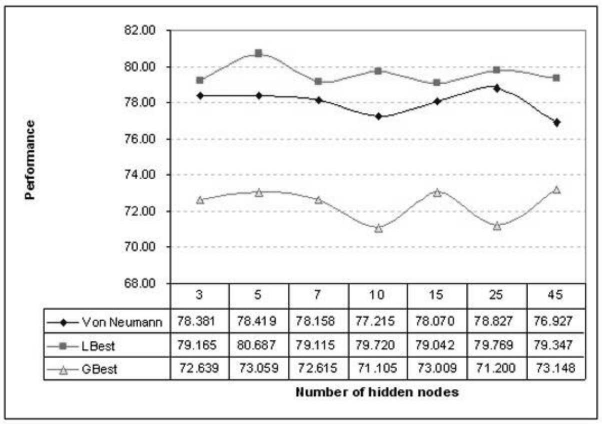
The first observation involves the clear performance ranking of the various neighbourhood structures. LBest, on average outperforms Von Neumann in all instances. GBest remains inferior throughout. An increase in swarm size also shows a definite increase in performance for all structures. Large swarm sizes cause slight oscillatory behaviour for the GBest structure, whereas both the LBest and Von Neumann structures show signs of converging on a stable solution.

On average, the performance values are significantly larger than their unoptimised predecessors, showing more controlled gains in performance that are less sensitive to the increase in swarm size – contrary to previous findings.

6.2.7 Increase in hidden layer size

Figure 6.5 graphically depicts the average performance F-values for each column in the different performance matrices, representing an increase in hidden layer size. The performance ranking of the three neighbourhood structures are once again clearly visible, with LBest outperforming Von Neumann, and GBest performing the worst.

Slight oscillatory behaviour is still present in all three structures' average performance plots, and the close relationship between LBest and Von Neumann as described in the previous



	3	5	7	10	15	25	45
Von Neumann	78.381	78.419	78.158	77.215	78.070	78.827	76.927
LBest	79.165	80.687	79.115	79.720	79.042	79.769	79.347
GBest	72.639	73.059	72.615	71.105	73.009	71.200	73.148

Figure 6.5: Increase in hidden layer size.

chapter is no longer visible. No definite performance trend due to the increase in hidden nodes can be established. Even though certain hidden layer sizes result in slightly improved performance values, the overall deviation between the F-values are not significant.

The optimised performance values outperform the previously computed average performance values yet again, resulting in a larger distinction between the worst (GBest) performer and the other two structures' performance.

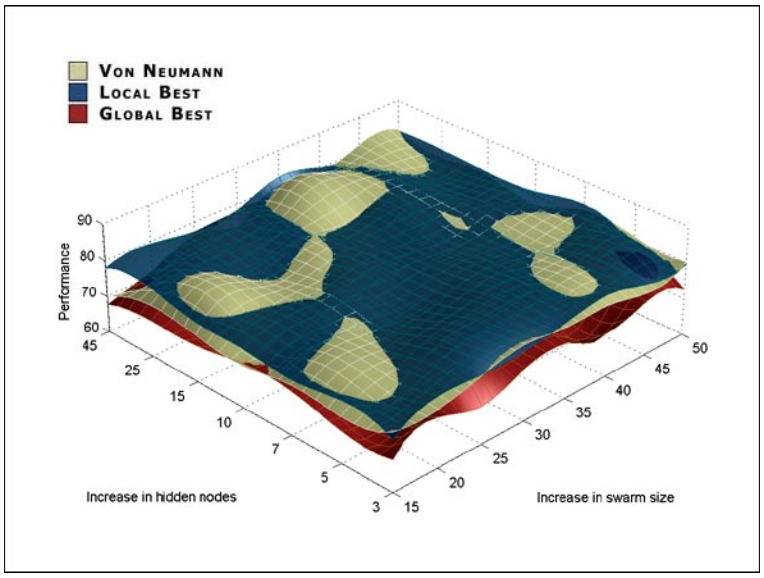


Figure 6.6: Improved performance surface plot.

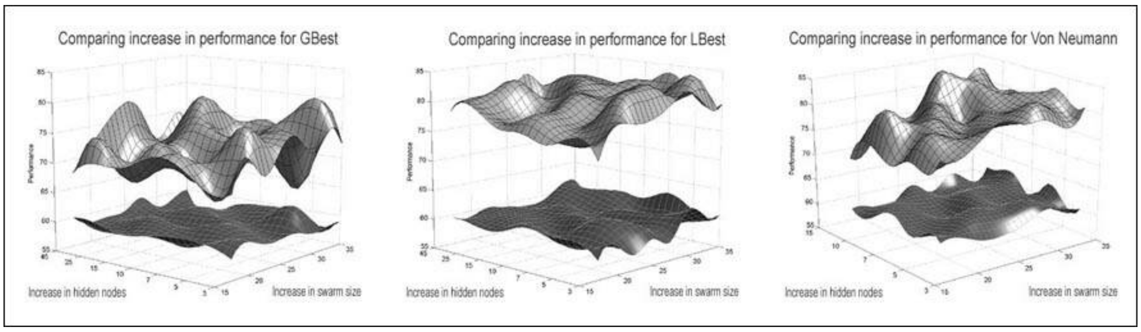


Figure 6.7: Individual gain by each neighbourhood structure.

6.2.8 Updated surface plots

Continuing with the visual analysis scheme applied to Tic-Tac-Toe and Checkers in the previous chapters, figure 6.6 depicts the various interleaved surface plots for the specific neighbourhood structure performance matrices. Each point on the surface corresponds to a height in a heightmap, represented by the performance (F-value) of a specific configuration for each neighbourhood structure.

A number of immediate observations can be made, starting with the absolute dominance of the LBest structure across large portions of the various configurations. The Von Neumann structure is able to outperform the LBest structure in a selection of very small and very large configurations, but the overall superior mid-range configuration performance is attributed to LBest. GBest does even worse than expected, without a single point of dominance across the complete configuration set.

It can also be valuable to examine the individual gains made by the various neighbourhood structures, when compared to the performance matrices provided in section 5.6.1 of the previous chapter. A series of comparison surface plots are illustrated in figure 6.7, starting with the performance increase of the GBest structure, followed by LBest and finally Von Neumann.

The GBest structure's improvement seems to be erratic at various configuration points of the performance matrix, with a slight indication of possible amplification of previously existing 'hot-spots' of good performance. The LBest structure shows a much more constant increase in performance across all configurations, whereas the Von Neumann structure shows more pronounced improvement in larger configurations.

Once again, all structures showed definite improvement over previously calculated surfaces, with a much more acceptable intelligent playing performance and F-values reaching into the mid 80's against a random-moving player.

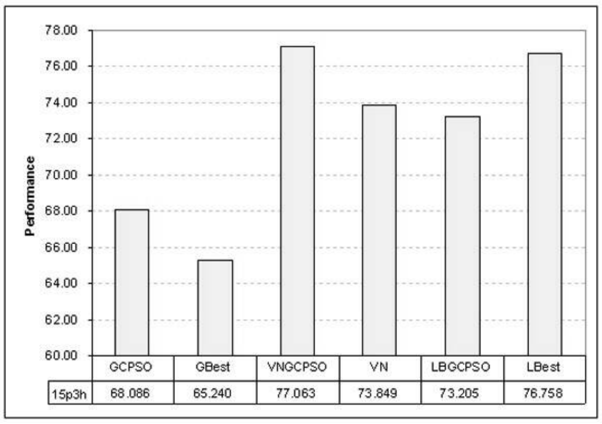


Figure 6.8: Overall PSO neighbourhood structure comparison.

6.2.9 Overall PSO comparison

Before departing from any PSO-specific investigation, it is perhaps necessary to examine the application of neighbourhood specific improvements developed in recent years – specifically the introduction of GCPSO introduced by Van den Bergh [137] and described in section 2.4.3. GCPSO was originally intended as an improvement on the standard GBest neighbourhood structure, allowing for an improved local search of the search space by the global best particle that reduced the probability of stagnating on suboptimal solutions. The GCPSO optimisation is extended to the LBest and Von Neumann neighbourhood structures, hereafter referred to as LBGCPSO and VNGCPSO respectively.

Some experimental work was conducted to determine the adequate GCPSO search term that should be used for the remainder of the simulations in this section. Search terms were tested from 0.1 to 2.1, at intervals of size 0.2 in each case. The experimental results are listed in table 6.6 for the GCPSO algorithm as applied to the standard GBest structure. Search terms with values larger than 0.3 showed serious oscillatory behaviour, resulting in definite decreasing performance as the search term was increased. A GCPSO search term value of 0.3 was selected to complete the remainder of the experiments, due to its superior performance.

Figure 6.8 graphically depicts the specific performance differences for the designated base experimental configuration of 15 particles and 3 hidden nodes. The expected improvement

Table 6.6: Performance of various GCPSO search term sizes.

Search term size	0.10	0.30	0.50	0.70	0.90	1.10	1.30	1.50	1.70	1.90	2.10
Performance (F-value)	71.84	74.01	68.58	70.50	66.90	70.00	66.31	70.43	67.75	67.85	68.02

for GBest is evident, and repeated for the Von Neumann (VN) structure's GCPSO variant as well. Somewhat surprisingly, the LBest structure does not seem to benefit from the GCPSO enhancements for this small configuration setting, but is expected to benefit on larger configurations.

It is encouraging to see that the VNGCPSO structure is able to outperform the LBest structure, with the VNGCPSO algorithm's F-value of 77.063 starting to approach the very good intelligent playing performance achieved by larger configurations. It should be noted that the GCPSO enhancement is selectively chosen for this small configuration size, and more in-depth analysis of more recent PSO algorithm enhancements, as well as the GCPSO performance on larger configurations are left as future work resulting from this research.

6.3 Neural network input representation

A previous discussion in section 5.5.3 on specific board state representation as input to the neural network highlighted the values assigned to different Checkers pieces, and the subsequent 'my piece versus opponent piece' value system. The following subsections investigate possible variations to the standard representation, with section 6.3.1 covering different piece value schemes, and section 6.3.2 investigating possible windowing methods that will allow for more spatial information about the board to be captured.

It should be noted that apart from the insights gained during the previous section on the effect of the maximum velocity and inertia terms to the PSO performance, the base experimental configuration has not changed to reflect any superior swarm size, hidden layer size or information sharing structure. The configuration used for the experimental work for the remainder of this chapter still relies on the Von Neumann structure, 15 particles and 3 hidden nodes. The c_1 and c_2 values remain fixed at 1.0, while the inertia and maximum velocity terms are set to 0.9 and 0.1 respectively.

The aim of the experimental work is not to evolve the most successful game playing agent, but instead analyse the impact that various other factors have on the game playing performance. The experimental results should be compared in this light.

6.3.1 Numeric input variations

Up to now, the different values assigned to the various Checkers pieces were evenly sampled in the range (0, 1). A player's personal kings were each represented by 1.0, and personal men represented as 0.75. Open spaces were represented by 0.5, while opponent men and kings were represented by 0.25 and 0.0 respectively. This can also be referred to as a 'centred' approach to piece valuation, with the open space representing the pivot to the scoring structure.

Upon examination of a third-party evaluation function (used in chapter 7 for benchmarking, and hereafter referred to as *SmartEval*), a different organisation to piece valuation became apparent. The author of SmartEval [51] assigned 64 points for a personal king, 16 points to an opponent king, 4 points for a personal man, 1 point for an opponent man, and 0 for an open space. The piece values were no longer pivoted around the open space, and a definite exponential decrease to the different piece values were shown.

A set of experimental tests were conducted to test the influence of the different piece value representations on the playing performance of the PSO-evolved players. A variation on the aforementioned non-centred (or linear organisation) approach to piece valuation were additionally examined. This variation scaled the large piece values originally assigned by the SmartEval author to smaller values within the range (0, 1).

The experimental results concluded that the original linear organisation of piece values as used by SmartEval performed the worst when applied to neural network input, with an F-value of 70.319. This can mainly be attributed to the saturation of the sigmoid activation function due to the use of the large numeric piece values. The scaled variant of the SmartEval piece value scheme performed better, with an F-value of 74.196, thereby confirming the possible cause of function saturation in the previous case. The original 'centred' approach to piece valuation proved to be the most successful piece representation scheme, with an F-value of 75.953.

Chellapilla and Fogel [53] [27] [26] evolved the piece values for personal and opponent kings. Their observations include a change of playing strategy that would be focused on crowning a piece above possibly capturing an opponent piece in the near future, or vice versa. Manual adjustment of the ratios between king and man piece value sizes did not yield a definite increase or decrease in performance, but a similar change in playing strategy is expected. Detailed analysis of playing behaviour is however listed as future work resulting from this study, and the evolution of king values are not further experimentally examined.

The original piece value representation scheme subsequently remains unchanged.

6.3.2 Windowed input

As defined by the rules of the game, only 32 squares of the board are valid playing areas. These 32 squares represented the inputs to the neural network, with each square's value corresponding to the current value of the piece residing on it, or the value assigned to an empty space.

Chellapilla and Fogel [53] [27] [26] made use of a 'sliding window' in an effort to provide more detailed information regarding the spatial aspects of the board to the neural network. The first window size has equal sides of 8 blocks, encompassing the whole board, and totalling

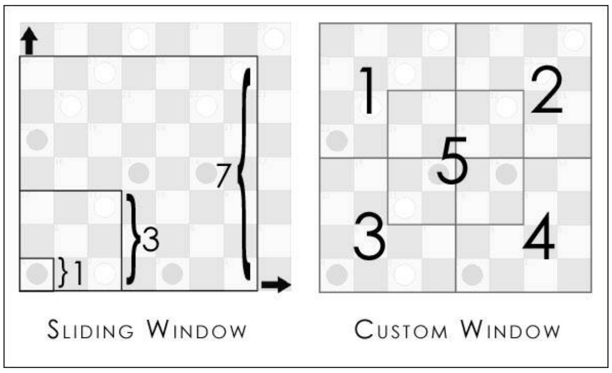


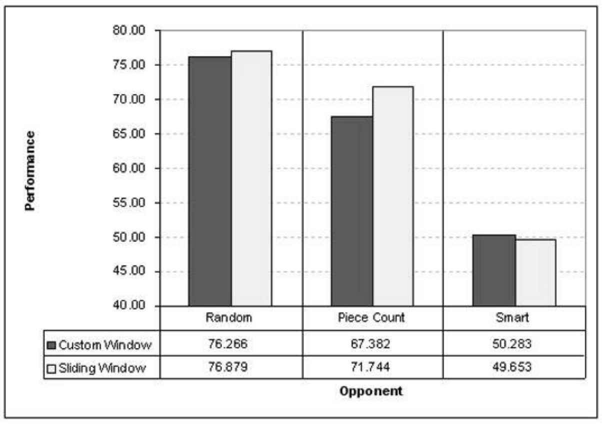
Figure 6.9: Various window formation schemes.

32 inputs. The window is then reduced by 1 square along the length and width to form a 7-by-7 input window, capable of 'shifting' three times to cover the complete board (one row down, or one column right, or a combination of the two). The smaller window results in an additional 98 inputs. The authors continue to minimise and shift the window until they reach a window size of 3, resulting in a grand total of 854 inputs. The neural network architecture in their work includes an additional hidden layer referred to as the 'spatial processing layer', consisting out of 91 neurons to 'process' the additional inputs, and hopefully extract some knowledge regarding the mobility of pieces.

A scaled down implementation of the aforementioned 'sliding window' approach is followed in this section's experimental work. The sliding window is computed for sizes 8-by-8 to 2-by-2, decreasing by a single block in both dimensions for each subsequent calculation. However, the windows are not concatenated to form a combination of various 'resolutions' of the information. Instead, the window sizes are examined individually for performance, and the best performing window size is compared to a custom created window. Both approaches are illustrated in figure 6.9. The custom window scheme consists of five 4-by-4 sub-windows, four of which are located in the corners of the board, and the fifth centrally placed to overlap the neighbouring windows and hopefully emphasise the importance of centre square control.

The first experiments aim to evaluate the effectiveness of the individual 'sliding-window' techniques. The experiments were conducted with swarm sizes of 15 particles and 5 hidden nodes. The results are depicted in table 6.7, with the 7-by-7 window size outperforming the other listed sizes by scoring an F-value of 76.879.

The 7-by-7 sliding window is compared to the custom created window by competing against three different opponents: a random-moving player, a piece-count based player and the aforementioned SmartEval player. More detailed information on the latter two opponents are presented in the following chapter, and the results are only listed to compare the relevant



	Random	Piece Count	Smart
■ Custom Window	76.266	67.382	50.283
□ Sliding Window	76.879	71.744	49.653

Figure 6.10: Influence of window formation.

windowing techniques.

Figure 6.10 graphically depicts the performance of the two approaches. The sliding window is able to outperform the custom window when competing against the random-moving and piece-count based players, but fails with a small margin against the SmartEval opponent. Overall, the best performance against the random player is set by the sliding-window technique, with an F-value of 76.879 – a significant improvement over the previous 'standard' input performance of 72.524.

An investigation into the use of the concatenated sliding window inputs as demonstrated by Chellapilla and Fogel [53] [27] [26] is left for future work resulting from this study.

Table 6.7: Performance of sliding-window formation techniques.

Window dimensions	Total inputs	Performance (F-value)
8-by-8	32	72.524
7-by-7	98	76.879
6-by-6	162	73.331
5-by-5	200	75.093
4-by-4	200	73.408
3-by-3	162	74.925
2-by-2	98	76.092

6.4 Coevolutionary techniques

After reviewing the influence of specific parameters relating to PSO along with variations of possible board state representations earlier in this chapter, it is now necessary to investigate the coevolutionary nature of the training algorithm to identify possible performance enhancements. The experimental work to date have only relied on a primitive tournament scheme, based on random sampling of five opponents, to quantify individual playing performance during each generation. This section will expand on this basic coevolutionary scheme by introducing a number of novel approaches to coevolution, based on Formula One Grand Prix [48].

6.4.1 Introducing Grand Prix methods

Formula One Grand Prix is a well-known international motor-sport, governed by the rules set forth by the Fédération Internationale de l'Automobile (FIA) [48]. The game of Checkers in itself has very little in common with Grand Prix (GPX), apart from the elements of competition and skill, as well as the emergence of a victorious competitor at the end. Due to the points scoring system associated with GPX racing, it also shares similarities with non-zero sum games. Drivers (players) finishing within a certain range of positions behind the victor is assigned a fraction of the winning points. In a number of cases the racing teams require their drivers to cooperate to achieve a greater goal, be it the increase of possible vehicle constructor championship points or increasing the aspiring world champion's lead on the overall points leader board. This may even sometimes require a driver to give up a winning position.

The major elements borrowed from current Grand Prix motor-sport are the concepts of 'racing seasons' in which a number of players (drivers) participate, as well as a variation of the corresponding scoring system for individual drivers. The first application of the borrowed elements is a coevolutionary racing system, more clearly described in section 6.4.2. A particle dispersement operator that makes use of the GPX scoring system is introduced in section 6.4.3, which resembles some characteristics from particle repelling – previously discussed in section 2.4.5.

6.4.2 Race performance

Blair and Pollock mention the occurrence of the so-called 'Buster Douglas effect' in their paper on coevolutionary environments and TD-Gammon [18]. Buster Douglas was world heavyweight boxing champion for nine months in 1990. The similarity between the short-lived boxing champion and co-evolving game players is simply the occurrence of a seemingly strong player created by chance through random mutation, that is able to perform very well initially, yet fail soon thereafter. The quick success and subsequent failure of the individual can simply be attributed to its exploitation of a common genetic weakness in the population, while still lacking robustness due to its 'lucky' genetic structure.

Grand Prix has similarly experienced so-called 'once-shot wonders', with certain drivers securing a win against all odds. The GPX points system does however favour consistency, and it is this consistency of playing performance in the coevolutionary environment that will be rewarded through the GPX racing scheme.

A predefined period of evolution bordering the maximum number of allowed epochs is selected to constitute the racing season between the different players. For example, the last 10% of generations may be allocated to the racing season. As soon as the season starts, each subsequent generation can be seen as a specific race that needs to be completed. After each generation, all the individuals (including personal best particles) are temporarily sorted according to their generation-specific coevolutionary scores, and in decreasing number of total moves if necessary. The reasoning behind this ordering is simply that a strong player will be able to defeat its opponent in as few moves as possible, whereas a weak player may struggle and need more moves. Due to the random sampling selection scheme in use, a strong and weak player may end up with identical coevolution scores depending on the level of randomly selected opponents they encountered.

The top five best performing players are assigned GPX points, after which the original population ordering is restored. Each player's GPX points total is maintained separately from its coevolution points. The GPX points accumulate over the racing season, in contrast to the coevolution points that are recalculated for each new generation. At the end of the training process (end of the racing season), the population is sorted according to their final GPX scores, and the top player is benchmarked accordingly.

It should be noted that the GPX scores do not influence the training process, and only serves as a method to identify more consistent performing players for benchmarking, hopefully eliminating any 'one-shot wonders' that outperformed their peers in the very last generation.

Change to algorithm

Figure 6.11 lists an adapted version of the training algorithm first introduced in section 3.5, which now includes a GPX racing season. For the sake of brevity, a condensed description is used where no change has been made to a specific section of the algorithm.

```
1. Instantiate population of agents.
2. Repeat for 500 epochs:
   (a) Add each agent's personal best NN configuration to the population.
   (b) For each individual in the population:
       i. Randomly select 5 opponents and play a game against each, always
          starting as 'player one'.
       ii. Assign +1 point for a win, -2 for a loss, and 0 for a draw after every game.
   (c) Compute best performing particle according to PSO algorithm in use.
   (d) For each agent (excluding personal best) update position and velocity according
       to standard PSO equations.
   (e) If GPX racing season has started:
       i. Sort population according to coevolutionary points score, and smallest number
          of moves if necessary.
       ii. Assign separate GPX scores to top 5 players, maintaining the sum of all
           previous GPX points.
3. Determine single best performing agent in whole population according to final GPX
   race ranking.
4. Perform benchmarking and compute performance values.
```

Figure 6.11: Adapted training algorithm to include GPX racing.

Influence of percentage of generations assigned to race

The first series of experimental tests aim to determine the influence of the percentage of generations assigned to the racing season, assuming a linearly decreasing scoring system. Since the search space in a coevolutionary environment represents a 'moving target' for possible optimisation, a balance should be maintained between players that were able to perform well during the initial stages of evolution, and those that perform well near the end of the evolutionary process. One possible drawback of assigning a large percentage of generations to the racing season, is the possibility of an early dominating player being selected for benchmarking based on previously assumed 'superior performance'. If a player is able to build up a 'reserve' of GPX points, it may be possible to secure its selection for benchmarking quite early on in the season.

Restricting the duration of the racing season may alleviate this possible problem.

Subsequent experiments were conducted with racing seasons represented by 5%, 10%, 20% and larger generation percentages until the maximum (100%) was allocated, with the remaining increased percentages occuring at intervals of 20%. The results do not show a consistent performance trend associated with an increased racing season duration. On average, a slight increase in performance is noticed for small (and decreasing) racing seasons, with a pronounced performance increase occuring at a duration of 40% (with an F-value of 77.301). Its isolated occurance however does not seem to make it statistically significant, and the results remain inconclusive for the specific baseline PSO and neural network configuration.

Scoring system

The second series of experimental tests aim to examine the impact of an exponential decreasing GPX scoring system, similar to the pre-2003 FIA [48] regulations, and compare it to the current and more linearly decreasing scoring system. Under the exponential decreasing scoring system implemented for the experimental work in this study, the top five best performing players are assigned points in the following order (starting with the winning player): 12, 8, 5, 3, 2. In contrast, the linear decreasing system assigns GPX points in the following order (also starting with the winning player): 12, 10, 8, 6, 4.

The distinction between linear and exponentially decreasing points systems stem from the imbalances experienced in the pre-2003 Formula One Grand Prix season, in which a winning driver was favoured too much and consistency not clearly rewarded. The new scoring regulations have addressed this balancing problem with the introduction of the linearly decreasing points system. It should be noted that the exact points allocated to the players in these experiments do not correspond 100% with the FIA regulations [48], and should instead be regarded as being based on the original systems. No attempts have been made to test larger (or smaller) individual point sizes.

Experimental tests once again involved adjusting the racing season duration in an identical manner to the aforementioned experiments. The experimental results are depicted in table 6.8. Apart from severe oscillatory behaviour, the exponential decreasing system seems to favour small durations, showing a definite increase in average performance for decreasing racing seasons. Apart from the observed trend on average performance, no definite conclusion can be made regarding the superiority of the exponential decreasing or linearly decreasing scoring systems. Overall, it is encouraging to notice some performance increases due to the GPX racing season scheme, and extending the baseline PSO and neural network configuration may yield more consistent results, but its examination is left for future work resulting from this research.

6.4.3 Particle dispersement

In the traditional evolutionary computing schemes the weak performing individuals are culled from the population through an elitism operator. After culling has been completed, the population size is restored by reproduction and subsequent addition of new individuals to the competition pool.

Culling individuals in the PSO context is not completely viable in this coevolutionary scheme, due to the inclusion of the personal best position as an additional player in the competition pool. An alternative to culling for coevolutionary PSO algorithms is presented here, utilising the aforementioned GPX scoring system and particle repelling techniques.

As the coevolutionary training process progresses, certain members of the swarm may get stuck in false local optima, which may negatively affect the overall training of the swarm. The particle dispersement operator aims to alleviate this problem by re-initialising a certain percentage of the worst performing particles to random positions in the search space. These newly dispersed particles' personal best positions are not re-initialised, but instead serve as a 'homing beacon' for the dispersed particles to guide them in the general direction of previously known suboptimal solutions. The premise is that in the worst case, a particle will return to its previously inferior position. The possibility does however exist that the particle may find a better solution en-route to the inferior solution instead. This may cause a shift in the global search of the swarm, potentially dragging the prematurely converged particles from their false optima towards the newly found superior solution.

Instead of assigning a contiguous range of generations to a racing season as was done in the previous coevolutionary extension, the particle dispersement operator is invoked at regular intervals throughout the duration of the training process. At each invocation, the individuals (excluding personal best particles) are sorted according to their current coevolutionary points

Table 6.8: Performance of various GPX scoring systems.

Season duration (%)	Linear decrease	Exponential decrease
100	74.977	73.107
80	73.475	71.264
60	73.889	76.591
40	77.301	70.334
20	72.887	78.412
10	74.816	72.855
5	75.169	78.025

total, and shortest number of moves made if applicable. The reasoning behind this sorting scheme has already been discussed in section 6.4.2.

After sorting the population, a section of the worst performing particles are re-dispersed into the search space. The size of the re-dispersed section is constant throughout the whole simulation, but the specific size selection is experimentally examined below. Re-dispersing a particle involves re-initialising its position (weight) vector according to the scheme developed by Wessels and Barnard [146], described in section 3.2.1. The velocity vector as well as the personal best position remain unchanged. The best performing particles are assigned GPX points according to the system described in section 6.4.2. GPX scores are however not maintained across multiple dispersement invocations. After evolution has completed, the population is sorted according to the final GPX scores, and the best performing particle selected for benchmarking against a random-moving player.

Change to algorithm

Figure 6.12 lists an adapted version of the training algorithm first introduced in section 3.5, which now includes a particle dispersement operator based on the GPX scoring system and particle repelling techniques. For the sake of brevity, a condensed description is used where no change has been made to a specific section of the algorithm.

Influence of dispersement size

The number of particles selected for dispersement should be experimentally examined. The baseline PSO and neural network configuration of 15 particles, 3 hidden nodes and utilising the Von Neumann structure is still applicable. A linear GPX scoring scheme is implemented across 500 generations, and the number of intervals at which the dispersement operator should be invoked is arbitrarily set to 50. Further experimental analysis on dispersement intervals are conducted in the following subsection.

Dispersement sizes were selected in the range [20%, 80%], at intervals of 10%. The results from the experiments are listed in table 6.9. A definite decline in performance is observed as a larger percentage of the swarm is re-dispersed into the search space. This would seem logical,

Table 6.9: Performance of various re-dispersement sizes over 50 intervals.

Dispersement size (%)	20	30	40	50	60	70	80
Performance (F-value)	77.812	76.034	73.106	75.696	75.848	69.571	70.114

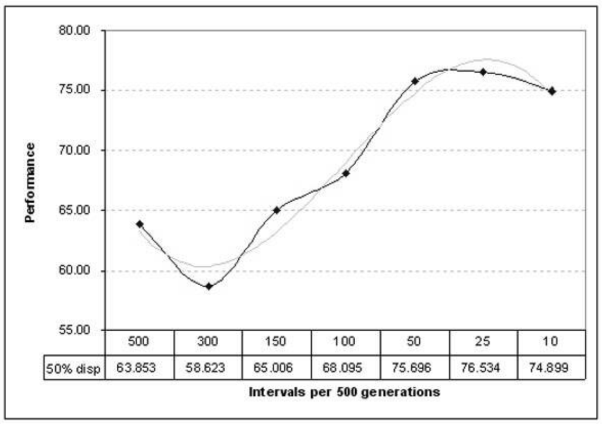
```
1. Instantiate population of agents.
2. Repeat for 500 epochs:
    (a) Add each agent's personal best NN configuration to the population.
    (b) For each individual in the population:
        i. Randomly select 5 opponents and play a game against each, always
           starting as 'player one'.
        ii. Assign +1 point for a win, -2 for a loss, and 0 for a draw after every game.
    (c) Compute best performing particle according to PSO algorithm in use.
    (d) For each agent (excluding personal best) update position and velocity according
        to standard PSO equations.
    (e) If current generation requires dispersement:
        i. Sort population according to coevolutionary points score, and smallest number
           of moves if necessary.
        ii. Select a percentage of the worst performing players, and re-initialise position
            (weight) vectors according to Wessels and Barnard.
        iii. Assign separate GPX scores to top 5 players, not maintaining the sum of all
             previous GPX points.
3. Determine single best performing agent in whole population according to final GPX
   points ranking.
4. Perform benchmarking and compute performance values.
```

Figure 6.12: Adapted training algorithm to include particle dispersement.

as a significant number of particles may actually have converged on good solutions, which are subsequently 'lost'. A dispersement size of 20% seems to provide the best results, attaining an F-value of 77.812.

Influence of dispersement interval

As was mentioned in the previous discussion on particle dispersement, the frequency at which dispersement takes place may also have a significant impact on performance. After particles have been re-dispersed, they should be allowed to explore the search space for a certain number



	500	300	150	100	50	25	10
50% disp	63.853	58.623	65.006	68.095	75.696	76.534	74.899

Figure 6.13: Influence of dispersement interval.

of generations before possibly being re-dispersed again. This will allow enough time to re-converge on possible optimal solutions.

The aforementioned experimental setup is repeated, with the addition of an arbitrarily chosen dispersement size of 50%. The experimental results are graphically depicted in figure 6.13, along with a faint trend-line aiding visual analysis. A clear increase in performance is observed when using smaller intervals. The best performance is achieved with 25 intervals per 500 generations (or 5%), scoring an F-value of 76.534.

It is once again encouraging to see that the implementation of the particle dispersement operator is able to increase the performance of the baseline PSO and neural network configuration of 73.849. Extending the investigation of dispersement performance into larger swarm and hidden layer sizes is left as future work resulting from this research.

6.4.4 Implementing 'Hall of Fame'

The concept of a 'Hall of Fame' (HOF) was covered in section 2.5.2. To quickly recap, the HOF maintains a list of previous best performing players, which gets updated at selected intervals to include the newly evolved players. Before the new players are inserted into the HOF, a mini-tournament is held among the existing HOF inductees. The results of this 'all versus all' tournament decide which of the previous best solutions should be permanently discarded from the HOF, in order to make room for the new players. The 'all versus all' tournament scheme relates to how the players compete against each other, which in this case means each player plays against each other player in a round-robin fashion – no random selection takes place. The HOF has a fixed size for the duration of the simulation.

Due to the constantly changing search landscape native to coevolutionary techniques, it may be possible for previously found 'good' solutions to be lost due to the current playing focus of the swarm. The HOF's main purpose is to make sure that newly evolved players are still able to compete against previously found solutions, increasing the robustness of the final solutions.

This section implements the HOF technique for co-evolving Checkers players, introducing the GPX scoring scheme as a simple ordering technique for the 'all versus all' tournaments. After each tournament is completed, the players are ranked according to their coevolution points and minimum number of moves where applicable. This ordering is used to select the number of players that must be replaced by the newly evolved players. It should be noted that the players inducted into the HOF do not continue training while they compete against each other, but instead serve as 'snapshots' of previous optimal players.

Three different aspects regarding any HOF implementation is experimentally studied below, including the influence of the HOF size, the number of inductees replaced at a time, and the number of intervals at which the HOF is invoked.

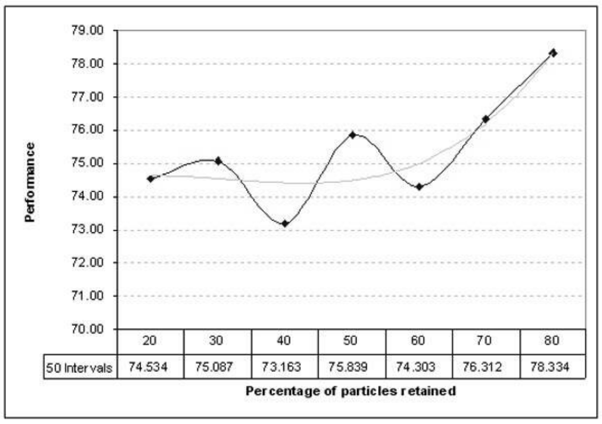
Influence of 'Hall of Fame' size

A set of experiments were conducted to determine the influence that the size of the HOF may have on playing performance. A larger HOF should be able to contain a more concise collection of previous performers, but may cause a strain on computing performance, due to the internal round-robin tournament scheme. Since a swarm size of only 15 particles is used, the HOF sizes ranged from 2 to 14 particles, at increasing intervals of 2 particles in each instance. The HOF was invoked at every 10 generations, at which point 50% of the original HOF players were replaced.

The experimental results are listed in table 6.10, and do not show a severe fluctuation of performance among the different HOF sizes. The results do show a slight advantage to using mid-range HOF sizes, with 8 particles scoring an F-value of 76.982, and 10 particles scoring an F-value of 76.768. Overall, no definite conclusion can be made regarding the HOF size for such a small PSO and neural network configuration. As mentioned previously, any extension to the scheme can be considered as future work resulting from this research.

Table 6.10: Performance values for various 'Hall of Fame' sizes.

Hall of Fame size	2	4	6	8	10	12	14
Performance (F-value)	76.116	76.245	73.940	76.982	76.768	76.614	74.956



	20	30	40	50	60	70	80
50 Intervals	74.534	75.087	73.163	75.839	74.303	76.312	78.334

Figure 6.14: Influence of ‘Hall-of-Fame’ survival rate.

Influence of survival rate

The following experiments aimed to investigate the influence of the number of HOF-inducted players that are replaced for each invocation of the HOF. Replacing a large number of historical players at a time may defeat the purpose of the HOF in its entirety, as the resultant HOF will degenerate into a smaller copy of the existing swarm. Replacing a too small number of particles may lead to insufficient exposure to the newly evolved players, thereby restricting the influence newly developed playing strategies may have on future HOF inductees.

The experiments were conducted with a HOF size of 10 players, consistently invoked at every 10 intervals across 500 generations. The results from the experiments are graphically depicted in figure 6.14, along with a faint trend-line to ease average performance comparison.

The graph does show some initial oscillatory behaviour, but the average performance indicates a clear advantage to retaining a large percentage of players. The best performing retention rate (80%) scored an F-value of 78.334, clearly showing an increase beyond the benchmark configuration performance of 73.849.

Influence of ‘Hall of Fame’ intervals

A final experimental analysis of the HOF implementation addresses the rate at which the HOF is invoked. Invoking the HOF too often may lead to a quicker replacement of previous historically best players, which may not always be desirable.

A HOF with space for 10 players were used for every specific interval test. At each interval, 50% of the particles were replaced. The results from this experiment is listed in tabel 6.11. Although no consistent performance trend is visible, there does seem to be an increase in performance for small interval sizes. The best performing test made use of 10 intervals across

Table 6.11: Performance of various invocation intervals for the HOF.

Intervals per 500 generations	500	300	150	100	50	25	10
Performance (F-value)	76.003	75.284	78.116	76.407	72.735	76.568	80.813

500 generations, and scored an F-value of 80.813.

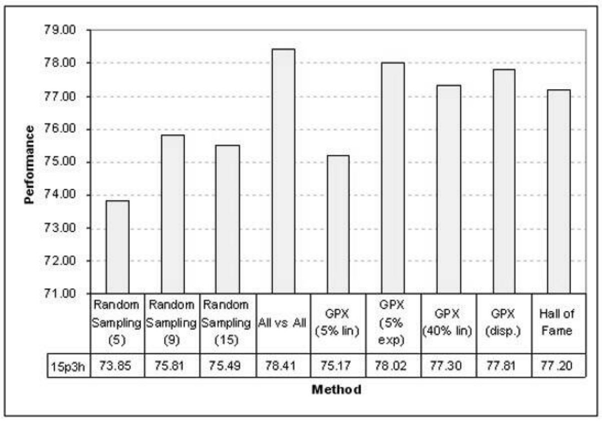
Overall, the HOF coevolutionary technique seem to improve on the benchmark performance for a configuration of 15 particles and 3 hidden nodes. The next section provides an overall performance comparison of the various coevolutionary techniques listed in this chapter.

6.4.5 Summary of coevolutionary methods performance

In order to ease comparison between the standard PSO-training algorithm's performance and the coevolutionary training optimisations introduced in this chapter, a graphical side-by-side comparison is made of a select set of test cases. Figure 6.15 depicts the results as a bar graph for an experimental configuration of 15 particles, 3 hidden nodes and utilising the Von Neumann neighbourhood information sharing structure.

The original PSO-based algorithm relied solely on random selection to quantify the specific individual's performance. Figure 6.15 shows that an increase in the number of opponents chosen during random selection may improve the overall performance of the specific individual.

The 'all versus all' tournament scheme employed by the HOF technique can be extended to include the complete swarm. This does however place a tremendous burden on computation time due to the sheer number of simulations that need to be completed in order to accurately quantify performance.



	Random Sampling (5)	Random Sampling (9)	Random Sampling (15)	All vs All	GPX (5% lin)	GPX (5% exp)	GPX (40% lin)	GPX (disp.)	Hall of Fame
15p3h	73.85	75.81	75.49	78.41	75.17	78.02	77.30	77.81	77.20

Figure 6.15: Performance of various scoring structures.

For each encounter, 6 games are played:

1. Player A (P_A) playing as player 1 (P_1), against Player B (P_B) playing as player 2 (P_2).
2. P_A as P_2, against P_B as P_1.
3. P_A's Personal Best (PBest) as P_1, against P_B as P_2.
4. P_A's PBest as P_2, against P_B as P_1.
5. P_A as P_1, against P_B's PBest as P_2.
6. P_A as P_2, against P_B's PBest as P_1.

Since such a thorough set of games are played to accurately quantify playing strength, the 'all versus all' technique does outperform the other coevolutionary methods – albeit with a significant computational overhead.

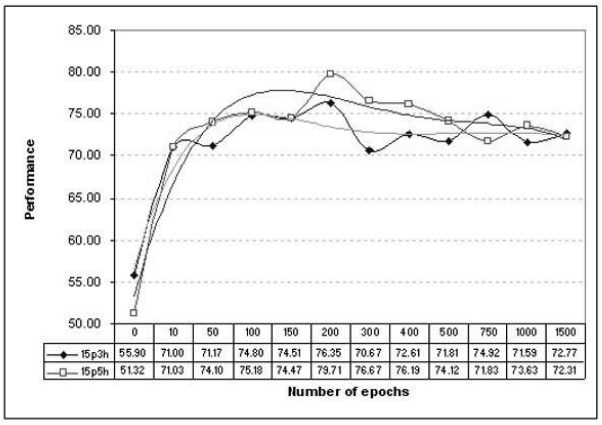
The Grand Prix racing season techniques perform reasonably well, with certain configurations on both the exponential and linearly decreasing scoring structures rivalling the superior 'all versus all' performance. The particle dispersement operator relying on the GPX scoring structure also performs adequately well. It should be noted that the overhead associated with the GPX racing and dispersement operators only incur a fraction of the computation penalties associated with the 'all versus all' technique, while still resulting in closely matched Franken performance values.

The HOF technique performs suitably well, but is also restricted by the overhead induced by the 'all versus all' tournament scheme associated with its inner workings. Limiting the HOF's size and/or the intervals at which the HOF is invoked may reduce the associated overhead.

6.5 Stricter training conditions

Up to now, the implemented rules of Checkers stated that a game may not last longer than 100 moves. This scheme has also been implemented by Chellapilla and Fogel [53] [27] [26] in their work on evolving intelligent Checkers game playing agents. Furthermore, the duration of evolution has been consistently set to 500 generations. This section investigates the impact on performance by adjusting these two parameters, and investigates whether or not it is possible to force a population to exhibit strong playing behaviour quicker than normal, or if the maximum allowed number of moves may actually favour certain playing styles.

Section 6.5.1 investigates the influence of the training duration on playing performance. Section 6.5.2 investigates the possibility of training a player to win within a certain restriction



	0	10	50	100	150	200	300	400	500	750	1000	1500
◆ 15p3h	55.90	71.00	71.17	74.80	74.51	76.35	70.67	72.61	71.81	74.92	71.59	72.77
□ 15p5h	51.32	71.03	74.10	75.18	74.47	79.71	76.67	76.19	74.12	71.83	73.63	72.31

Figure 6.16: Influence of training duration.

on the maximum allowed number of moves (smaller than 100). Finally, section 6.5.3 investigates the upper-bound of the move count, and how virtually removing it may impact playing performance.

6.5.1 Influence of training duration

The assumption of using 500 epochs to train a population of players to play Checkers have been made on past experience of PSO algorithms and coevolutionary training. That assumption is experimentally examined in this section, by performing benchmarks from players that did not participate in a single evolution step (*i.e.* the original randomly initialised particles), to players that have evolved for 1500 generations.

Two different player configurations are used. Both make use of swarm sizes of 15 particles, but the first relies on a neural network with 3 hidden nodes, while the second relies on a neural network with 5 hidden nodes. The reason for an increase in hidden nodes is to examine the possible effect of overfitting on the different configurations over time.

Figure 6.16 graphically depicts the playing performance (using F-values) for the different evolved players over time, along with faint trend-lines to aid in average performance analysis. Both graphs show a sharp rise in performance as training (evolution) progresses, reaching a peak at 200 generations. The performance of both configurations seem to steadily decline thereafter, with the player utilising the increased hidden layer outperforming the baseline configuration consistently. After 500 generations the graphs degenerate into a mutual pattern of oscillatory behaviour, almost converging at 1500 generations.

A number of interesting observations can be made from the two graphs. The most obvious observation is that learning does indeed occur quite rapidly! The second observation is the

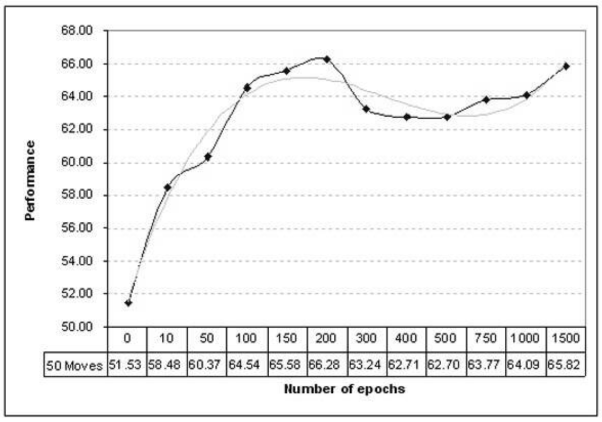


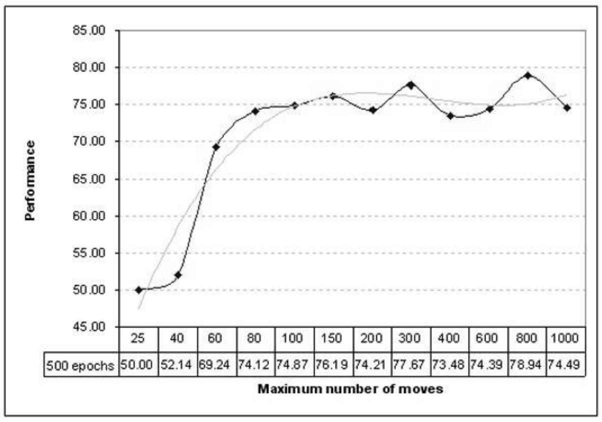
Figure 6.17: Winning in less than 50 moves.

performance difference caused by an increase in hidden layer size, with larger hidden layer sizes outperforming smaller hidden layer sizes. It would seem that neural network specialisation might be the cause for the decline in performance. As described in section 2.2.2, training a neural network for too long may cause it to over-specialise and thereby lose its ability to correctly predict the correct move to make from the options provided by the game tree. In this specific case over-specialisation is enforced due to the convergence of the swarm on a possible suboptimal solution, resulting in a decreased level of diversity and only allowing for a certain playing style. The best individuals are not exposed to a diverse range of players during training (due to the convergence), and thereby over-specialise their neural networks to only cope with the current population's playing style. When faced with the random playing style of the benchmark opponent, it is unable to generalise on the previously unseen board states, resulting in incorrect evaluation of key board states, and degraded performance.

Increasing the duration of evolution beyond 500 epochs does not seem to be a worthwhile option for the baseline configuration. Larger configurations may require more generations to converge, but a degradation in performance is expected thereafter.

6.5.2 Winning under pressure

The next set of experiments with the baseline configuration involves restricting the players to compete for only 50 moves, instead of the usual 100. Hopefully this will force agents to discover quicker methods of winning against their opponents, naturally increasing their playing strength due to the enforced pressure to win. The evolution will be extended to continue for up to 1500 generations, thereby providing enough time for the players to evolve their expert playing strategies.



	25	40	60	80	100	150	200	300	400	600	800	1000
500 epochs	50.00	52.14	69.24	74.12	74.87	76.19	74.21	77.67	73.48	74.39	78.94	74.49

Figure 6.18: Influence of a restricted move count.

The results from these experiments are depicted in figure 6.17. The peak performance first observed in the previous section is once again repeated at 200 epochs. The exact F-value however, shows a definite decreased performance (66.28 versus 76.35). This can be attributed to the increased difficulty involved in winning in less than 50 moves, with most games having a higher probability to result in a draw.

A drop in performance is experienced after 200 epochs, after which a steady gain in performance is noted until 1500 epochs. The highest performance value attained at 200 epochs is not regained within the 1500 epoch limit. From the results it is clear that players can be evolved to win at Checkers in less than 50 moves. An analysis of playing behaviour indicates a large number of draws and wins for the evolved player, with less than 15% of the games being won by the random-moving player. It would seem as if the reduced move count has handicapped the random-moving player's ability to adequately perform as a benchmark opponent, rather than the evolved players outperforming the random-player on merit. It is encouraging to see that it is possible to win in less than 50 moves, but exactly how that is accomplished is not clear. Increasing the configuration size to include more particles and/or hidden nodes may yield better performing agents.

The hypothesis of winning in less than 50 moves is however hereby proved with the baseline configuration, and the possible evolution of more optimal performing agents in the future are not disregarded.

6.5.3 Varying the maximum number of moves

Expanding on the previous experiments with a restricted maximum move count, the inverse approach may also prove interesting to examine. A series of experiments were conducted to

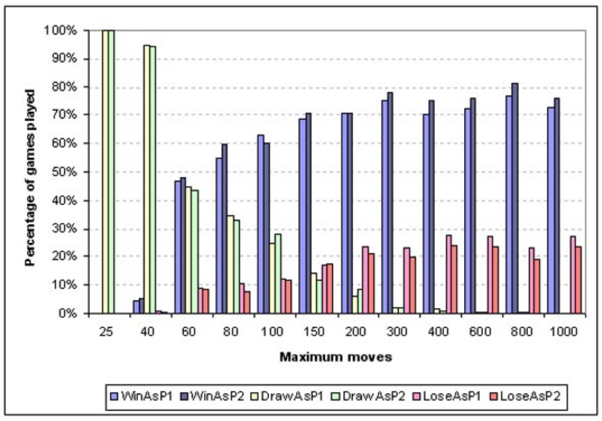


Figure 6.19: Win/Draw/Lose relationship.

determine the influence of an increased maximum move count on the ability of the evolved players to increase their performance, while maintaining evolution for 500 epochs. All the experiments made use of the baseline configuration of 15 particles and 3 hidden nodes, utilising the Von Neumann neighbourhood structure and restricted maximum velocity.

The F-value performance results for experiments ranging from a maximum of 25 moves to a maximum of 1000 moves are graphically depicted in figure 6.18. Games restricted to 25 moves consistently result in a draw between the evolved and random-moving players, showing the definite improbability to win in 25 moves or less. A sharp increase in performance is experienced after 40 moves, with performance levels converging for a short period after 80 moves. Extending the maximum move count beyond 150 moves results in visible oscillatory behaviour, with selective performance increases being achieved above the 150 move score of 76.19. It is disappointing to see that even with a maximum move count of 1000 moves, the evolved player is not able to deprive the random-moving player of any wins.

More insight into the percentage of games won, drawn or lost for each maximum move setting is provided in figure 6.19.

The high number of draws alluded to earlier is evident in the small maximum move settings, with draws making place for wins as the move restriction is relaxed. After 400 moves the number of draws are virtually zero, with the evolved player still able to beat the random-moving player consistently in all the cases, even managing wins in over 80% of the games played. As mentioned earlier, the increase in the number of wins does not guarantee that the evolved player will be victorious in all the games it competes in. Even though the percentage of games won does increase slightly, the number of draws diminish completely and instead make way for the random-moving player to win a portion of the games.

It is expected that larger configurations of swarm or hidden layers sizes may perform proportionally better, but the average trend in win/lose/draw behaviour is expected to remain the same for at least the front half of the graph.

6.6 Conclusion

This chapter aimed to investigate the various reasons for the poor playing performance of Checkers agents described in the previous chapter. The first set of experiments investigated PSO-specific parameter choices. A major increase in playing performance was achieved by restricting the maximum velocity to 0.2 – contrary to the previous experiments conducted with Tic-Tac-Toe and Checkers. Adjusting the c_1 and c_2 values did not yield an overall performance increase, but a static inertia value of 0.9 proved to have a beneficial effect. The inclusive LBest neighbourhood size of 5 particles were experimentally shown to be superior to larger neighbourhood sizes. The analysis of overall performance matrices indicated that the LBest structure has surpassed the dominance of the Von Neumann structure. An increase in swarm size benefited all the neighbourhood structures' performance, while an increase in hidden layer size was inconclusive. Finally, the selected use of an improved PSO structure, GCPSO, allowed for even larger gains in performance. The adapted Von Neumann structure (VNGCPSO) was able to regain the best performance during an overall neighbourhood performance comparison with the baseline configuration.

The second set of experiments aimed to investigate the effect that different approaches to board state representations for neural network evaluation may have on the overall playing performance. Different piece valuation schemes and 'sliding-window' approaches were experimentally examined. The standard 'centred' approach to piece valuation remained dominant, while there proved to be an advantage to using a windowing scheme. The sliding window of size 7-by-7 squares selectively outperformed the custom window organisation of five 4-by-4 squares.

The third set of experiments extended the basic coevolutionary training algorithm to include concepts borrowed from Formula One Grand Prix. The introduction of the Grand Prix racing season paradigm during training aimed to benefit players that play consistently well, while eliminating any potential 'one-shot wonders'. Both linearly and exponentially decreasing scoring systems were examined, with a marginal improvement gained by the linearly decreasing scheme due to its focus on rewarding consistency. A particle dispersement operator was introduced that relied on the GPX scoring scheme to overcome convergence on suboptimal solutions. Finally, the implementation of the 'Hall of fame' coevolutionary structure was examined, and experimentally compared to the aforementioned Grand Prix methods. The Grand Prix methods perform adequately well, without the extensive overhead required by any ‘all-versus-all’ methods.

Lastly, an analysis on the maximum move count and training duration showed some insight into possible neural network over-specialisation, stagnation on suboptimal solutions, and the overall win/lose/draw ratios for extended play. The majority of the experiments in this chapter were conducted with the baseline configuration of 15 particles and 3 hidden nodes, and results may vary for more complex configurations. Extending the baseline configuration is left as future work resulting from this study.

CHAPTER 7

Assessing intelligence

"The ability to focus attention on important things is a defining characteristic of intelligence".
- Robert J. Shiller

After resolving the reasons for the initial poor playing performance, this chapter takes the benchmarking of the evolved Checkers players one step further, by analysing playing performance against two 'intelligent' evaluation functions. The impact of training and playing on deeper tree depths are also investigated, ending with a discussion on the possibility of improving the coevolutionary training partner.

7.1 Introduction

Up to now the standard method of benchmarking an evolved player has relied on measuring its performance while playing 300 000 games against a random-moving player. The random-moving player was chosen for its consistency, varying playing style and lack of human intelligence. Even a novice observer to these experiments would ponder: "How intelligent are these agents really?" That question is answered in this chapter, by pitting the best evolved agents against more intelligent adversaries.

Section 7.2 introduces the two 'intelligent' evaluation functions used for this goal, namely a piece-count based evaluation, and SmartEval – a handcrafted evaluation function from the author of Cake++ [51]. Section 7.3 benchmarks the new evaluation functions, and compares their performance against a random-moving player, evolved players, and themselves.

The consequences for increasing the tree depth during training and playback are experimentally covered in section 7.4, followed by a short series of observations on typical playing behaviour of the intelligent agents, as made by the author.

The final discussion revolves around the possibility of improving the coevolutionary training partner, thereby causing an additional improvement in playing performance, in section 7.6. Section 7.7 summarises some of the important points relating to the chapter.

It should be clearly stated from the start of this chapter that the experimental work conducted hereafter was aimed at briefly exploring avenues for future research, by highlighting the shortcomings of the current neural network training approach and hypothesising about possible solutions to the problem. Unless otherwise stated, the experimental configuration remained 15 particles and 3 hidden nodes utilising the Von Neumann PSO structure, with c_1 and c_2 remaining at 1.0, and the inertia and maximum velocity terms set at 0.9 and 0.1 respectively. No definite conclusions are made based on the limited experimental configuration, and the results should at most be interpreted as interesting precursors to more in-depth study following from this research. As previously mentioned, the aim of the study is not to create the most intelligent game playing agent, but instead investigate the different factors that impact playing performance. The reader is urged to review the experimental results in this light.

7.2 Intelligent evaluation functions

In order to examine the intelligent playing behaviour on a broader scale, this study makes use of two different types of intelligent evaluation functions. The intuitive piece-count based evaluation function is discussed in more detail in section 7.2.1, followed by a closer look at a sample hand-crafted evaluation function, codenamed SmartEval and based on ‘simple checkers’ by Martin Fierz [51], in section 7.2.2.

7.2.1 Piece-count based evaluation

The piece-count evaluation function is not terribly complex, and can be considered to not include any human intelligence at all. It does however rely on the basic understanding of arithmetic, something most humans are intuitively capable of. As already mentioned in section 5.5.3 on the neural network configuration for Checkers game agents, previous work by Chellapilla and Fogel [53] [27] [26] added a piece-count evaluation result as additional input directly into the output layer, thereby aiding the neural network to make more ‘informed’ decisions about the board state.

Fogel and Chellapilla reasoned that even the most novice player would have an understanding of the number of pieces on the board, where having more pieces than your opponent would place you in a positive position. It was for this reason that they included the additional 'hard-coded' term into their neural network configuration. For the experimental work in this study, a pure neural network approach was followed, using no external information apart from the immediate board state.

The piece-count evaluation function used as the first intelligent benchmarking opponent for this chapter's experimental work, is defined as follows:

$$\epsilon = (\mu \times pieceCountAdvantage) + pieceCount + \omega \qquad (7.1)$$

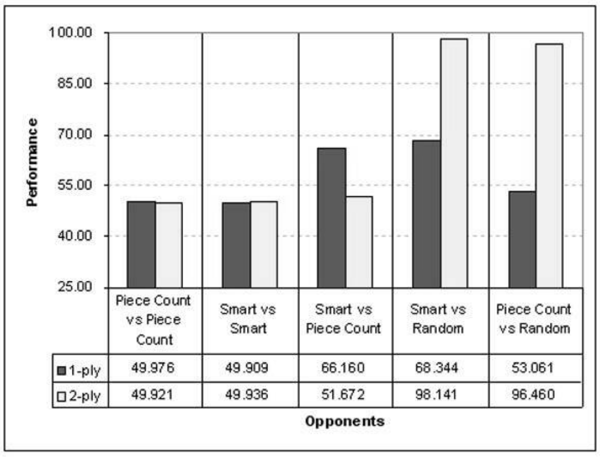
where ϵ is the resultant evaluation and μ represents a bias factor to the calculation of the piece-count advantage. The $pieceCountAdvantage$ term is calculated as the difference between the number of pieces held by each player, while $pieceCount$ represents the total number of available personal pieces on the board. Finally, ω acts as a random tie-breaker term to resolve decisions between equal board state evaluations.

7.2.2 SmartEval – hand-crafted evaluation

The next intelligent benchmarking opponent is provided by publicly available source code from the author of Cake++, Martin Fierz [51]. Originally called 'simple checkers', the code formed part of a basic Checkers engine that could be extended into a much larger game playing system. To avoid any possible engine-related performance mismatches during the benchmarking process, only the evaluation function was used and converted into the local code-base – resultantly known as *SmartEval* thereafter.

SmartEval consists of more than 100 lines of readable C code, and as far as can be determined considers the following well-known hand-picked features of a Checkers game to form a final evaluation:

- Determines whether a back rank guard is available.
- Determines if an intact double corner exists.
- Evaluates control of the central area of the board.
- Brings the number of pieces on the edges of the board into consideration.
- Calculates material advantage.



	Piece Count vs Piece Count	Smart vs Smart	Smart vs Piece Count	Smart vs Random	Piece Count vs Random
■ 1-ply	49.976	49.909	66.160	68.344	53.061
□ 2-ply	49.921	49.936	51.672	98.141	96.460

Figure 7.1: Benchmark results for the intelligent evaluation functions.

All of the above features are appropriately weighted and combined into a final evaluation function to quantify the value of the current board state. A tie-breaking term is also added to resolve decisions on identical board state evaluations.

7.3 Benchmarking Intelligence

Two approaches to benchmarking was followed in this chapter. The first continued to use self-play to evaluate the strength of the individual players, as was done with the random-moving player in section 3.3.1. The second form of benchmarking involves comparing the performance of the evolved players making use of the neural network-based evaluator against the intelligent evaluation functions. The performance in all of the cases are quantified through the use of the Franken performance measure.

7.3.1 Self-play

In order to accurately compare the performance of these intelligent evaluation functions, the same process of benchmarking through self-play – as employed for the random-moving player – is repeated here. Each evaluation function played 150000 games as player one, and 150000 games as player two, against an identical version of itself.

As an added interest, the ply-depth was increased by one level to observe the advantage gained from evaluating board states deeper into the game tree. Figure 7.1 graphically depicts the results from the initial benchmarking process using the Franken performance measure (F-values). The performance values are listed from the perspective of the first-named evaluation function, for example, ‘Smart vs Piece Count’ would list Smart’s performance value when competing against a piece-count based player.

The first interesting observation is once again the balanced nature of the players using the respective intelligent evaluation functions. In both the 'Smart vs Smart' and 'Piece Count vs Piece Count' cases, the play strength is equally matched in the region of 49.9.

Looking at the win/draw/lose ratio's for the different benchmarks, the piece-count based evaluation has a large number of draws, and balanced number of wins when playing as either the first or second player. The SmartEval, however, has a very large number of balanced wins and losses as either player, with less than 2% of the games resulting in a draw.

Extending the ply-depth on self-play benchmarking does not have any significant impact on the performance value computed for each evaluation function. The piece-count based player's playing behaviour remains the same, with a large number of draws and balanced number of wins as either player. The SmartEval based player shows a significant increase in the number of drawn games (almost half), and a sudden shift in advantage to the player playing first.

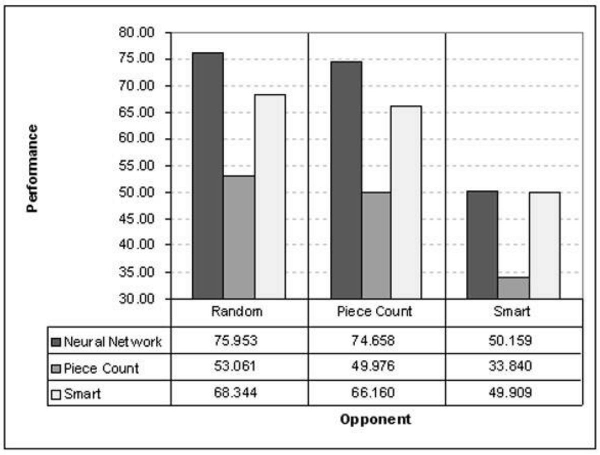
Allowing the intelligent evaluation functions to compete against other opponents, including the random-moving player, yields interesting results. The SmartEval based player is able to outperform the piece-count based player on single-ply, but is faced with stronger opposition as soon as the ply-depth is increased. Play analysis shows an increase in drawn games as the ply-depth is increased, with balanced wins on either side playing as player one or two.

SmartEval clearly outperforms the random-moving player on both ply-depths, but this was expected. The piece-count based player also outperforms the random-moving player on an extended ply-depth – trailing only slightly behind the performance achieved by SmartEval – yet curiously struggles on single-ply. Play analysis show equal levels of winning as player one and player two on single-ply, but a significant decrease in draws as compared to the piece-count self-play benchmark.

Judging by the extremely high performance values of the SmartEval and piece-count based evaluation functions against a random-moving player on two-ply, it would seem that any advantage that the random-moving player had when competing on a single-ply depth has completely disappeared.

7.3.2 Evolved players

The true potential of the evolved Checkers players are now tested by benchmarking them against the three custom evaluation functions, namely the random-moving player, the piece-count based player and SmartEval. The F-value results are graphically depicted in figure 7.2, once again with interesting results. All the benchmarks were completed on single-ply, with a configuration of 15 particles, 3 hidden nodes and utilising the Von Neumann information



	Random	Piece Count	Smart
■ Neural Network	75.953	74.658	50.159
▨ Piece Count	53.061	49.976	33.840
□ Smart	68.344	66.160	49.909

Figure 7.2: Performance of evolved players against intelligent evaluation functions.

sharing structure. These parameters were selected in order to remain comparable to results from the previous chapter, and for increased simulation speed. The performance values are computed using the Franken performance measure.

Starting with the random player as the first opponent, the performance of the neural network as computed in the previous chapter is listed again, alongside the performance of the piece-count and SmartEval evaluation functions. The neural network immediately shows its strength against the random player, scoring higher than both SmartEval and the piece-count evaluations. Referring back to table 3.2 in chapter 3 regarding the initial random-moving player self-play benchmark, and its subsequent conversion into the Franken performance measure, it is interesting to note how small an improvement the piece-count player is able to make in this regard (53.061 vs 50.0). The SmartEval player plays slightly better, yet barely reaches the lower fringes of performance observed by the weakest configurations in the improved performance matrices from chapter 6.

The piece-count player is now considered as benchmarking opponent, and the various performance values from the competing evaluation functions are listed once again. As was intended by its inclusion, the piece-count evaluation function provides a more difficult opponent for all the competing players. The neural network is once again the best performing player against the piece-count opponent, but does so with a slightly decreased performance rating of 74.658 (as compared to the random-moving player benchmark of 75.953). This enforces the presence of stiffer competition. The SmartEval function is outperformed by the neural network performance yet again, while also showing a slight drop in performance. The self-play piece-count value was already computed and discussed in section 7.3.1 of this chapter, and is included for completeness only.

Lastly, the performance against the SmartEval player is also listed in figure 7.2. The SmartEval evaluation function was chosen to provide the stiffest competition to the other benchmark problems, and this can clearly be seen in the low performance values. It is encouraging to see that the neural network player is able to outperform even the SmartEval player – albeit by the smallest of margins (0.25 F-value points). Looking at the win/draw/lose ratio's of the benchmark matches, it seems overall to be evenly spread between winning as player one, winning as player two, or drawing in either case. It seems that during some simulations the best evolved individual would 'specialise' in a certain playing style, by generally performing better as player one, or better as player two, or even drawing more games. There are also players that perform equally well irrespective of starting position, which could be considered to be 'well-rounded' players.

The piece-count based player fails to make an impression on the SmartEval performance (as already noted in the previous benchmark). The performance value listed is simply the compliment of the Franken performance measure first calculated for the SmartEval player, and now reported from the perspective of the piece-count player (originally 66.160, now reported as 33.840). The self-play SmartEval performance value has already been discussed, and is once again included for completeness.

Even though the results are not groundbreaking in terms of performance, they are encouraging for this minimal configuration – proving that the neural network player is indeed at the very least as 'intelligent' as the SmartEval player at single ply-depth. An increase in ply-depth is discussed below, and benchmarking against more advanced players are left as future work resulting from this research.

7.4 Increasing tree depth

The main reasons for keeping to a single ply-depth in the experimental testing for this research, was attributed to two factors. Firstly, the focus of the study was on PSO and its applicability to game learning, and not to create an efficient game engine capable of beating the world's best game players. Therefore a large number of simulations required the restriction of the tree depth to single-ply due to time-constraints and processor availability. Secondly, competing at single-ply in itself is a form of 'worst-case' reasoning about the evaluation function, as it cannot rely on accurate 'guessing' about any opponent responses, and instead is strictly limited to its representation of the board state. The random-moving player itself also has no use of an increased ply-depth. As the previous section illustrated, this training and playing scheme proved to work well in benchmarking different evaluation functions.

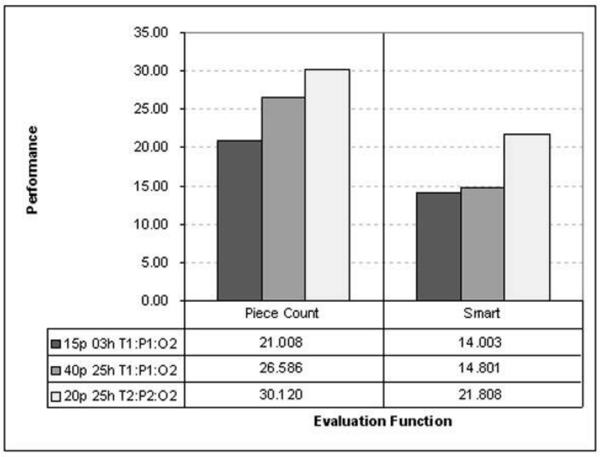
The benefits of an increased ply-depth have been discussed at various sections earlier in this study. While conducting the single-ply experiments, a number of questions were raised, namely:

1. How would a player trained on single-ply perform against an opponent playing on two-ply? (similar to HITECH vs LOTECH as discussed in section 3.4.1)
2. Would it be possible to construct a single-ply configuration that outperforms an intelligent player playing at two-ply?
3. Would an evaluation function evolved through training on single-ply be scalable for use in two-ply evaluation?
4. Would an increase in the training ply-depth require an increase in the complexity of the PSO simulation configuration in order to cope with the additional game information?
5. Would it be possible to train a player on a large ply-depth, and use the resultant evaluation function contained by the neural network to play on smaller ply-depths? This should require a fraction of the processing time while playing, and can be seen as 'compressing' the game tree knowledge into the inherent black-box structure of the neural network.

The majority of these questions are addressed below, by referring to figures 7.3 and 7.4 that illustrate the potential impact of increasing the PSO configuration size as a substitute to using an increased tree-depth, and training and playing on deeper tree depths respectively. The symbols used in these figures to represent experimental configurations have the following meaning:

- p [preceded by the swarm size]
- h [preceded by the hidden layer size]
- T [followed by the training ply-depth and a colon separator]
- P [followed by the playing ply-depth for the evolved player and a colon separator]
- O [followed by the playing ply-depth for the opponent]

The experiments were conducted according to the training algorithm described in chapter 3, and only serve as a single example that addresses, proves or disproves the appropriate hypothetical question. No attempt has been made to optimise the performance of the evolved player, in order to maintain comparability with previously listed configuration results. Optimisation of the player performance is regarded as future work resulting from this very brief experimental exploration.



	Piece Count	Smart
■ 15p 03h T1:P1:O2	21.008	14.003
▨ 40p 25h T1:P1:O2	26.586	14.801
□ 20p 25h T2:P2:O2	30.120	21.808

Figure 7.3: Increasing configuration as a substitute for tree depth.

7.4.1 Q1: Performance against a dominating ply-depth

Previous research on the impact of an increased ply-depth on performance were discussed in section 3.4.1. The discussion mentioned the dominance of LOTECH – a 'dumber' version of a more complex Checkers program codenamed HITECH – that was able to judge moves one ply-depth deeper than its predecessor. LOTECH consistently outperformed the more intelligent HITECH.

The experiments conducted for this section aim to duplicate the findings on a smaller scale, by pitting an evolved player (initially playing at singly ply-depth) against the two intelligent evaluation functions introduced earlier in this chapter (both playing at a ply-depth of two). The first entry in figure 7.3 lists the results for the benchmark configuration with 15 particles, a hidden layer size of 3 nodes and utilising the Von Neumann neighbourhood information sharing structure.

In each case the best evolved player occurring over 15 simulations played against the piece-count and SmartEval evaluation functions, competing in an overall total of 150 000 games as player one, and 150 000 games as player two. The Franken performance measure was used to quantify the exact playing performance.

Referring back to figure 7.2, the evolved neural network player managed to outperform both the piece-count and SmartEval players on single ply. Pitting the neural network against the same players, but allowing the opponents to have an advanced look into the future by means of an extended ply-depth, does not provide the same results.

The results show a significant drop in performance, from 74.658 to 21.008 against the piece-count player, and from 50.159 to 14.003 against SmartEval. The only consistency in the results can be seen from the performance ranking of the two opponents, with the evolved player still performing 'better' against the piece-count based player than against SmartEval.

The trends established by the original Hitech versus Lotech experiments were repeated once again, with the conclusion that a player trained and playing on a single ply-depth, when pitted against a two-ply depth opponent, stands a great chance of losing the match.

7.4.2 Q2: Does increased complexity outperform increased ply-depth?

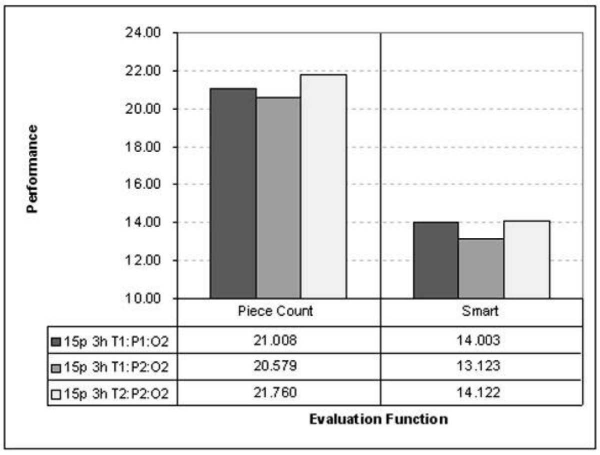
The poor preceding experimental results – although not completely unexpected when taking into account the small swarm and hidden layer size – lead to the following question: Would a more complex configuration be able to improve on the previous setback if the experiment was repeated? The second entry in figure 7.3 lists the results of a more complex configuration, consisting of 40 particles, 25 hidden nodes and still relying on the Von Neumann neighbourhood information sharing structure.

The new experimental results have improved over the performance values calculated for the initial configuration, but only by a small margin. The performance ranking of the two intelligent evaluation functions remain the same, with SmartEval still proving more difficult to beat than the piece-count based player. Analysis of the win/draw/lose ratios indicate a very sporadic winning ability by the evolved player, with the majority of the performance points being derived from drawn games. The fact that the reported F-values from figure 7.3 are still below 50.0 (the baseline for improved intelligent playing behaviour in the Franken performance measure), is an indication that the evolved player still does not outperform its opponents that have access to increased ply-depths.

The final experiment traded complexity with ply-depth, as illustrated by the third entry in figure 7.3. A configuration of 20 particles, 25 hidden nodes and utilising the Von Neumann neighbourhood information sharing structure was able to comfortably surpass the previous results – mainly due to the aid of the extended ply-depth. The results on their own are not very impressive, but the question has been answered: Increased complexity is not guaranteed to compensate in performance for a lack of training and/or playing ply-depth – its only guarantee being an increase in processing time.

7.4.3 Q3: Scalability of trained evaluation function

The time restriction usually associated with experimental work, specifically processing time during training, is something most researchers are familiar with. The inability to leave a computer to train for six months (as was done by Chellapilla and Fogel [53]) in addition to the aforementioned weak performance of the evolved players trained on single-ply and competing



	Piece Count	Smart
■ 15p 3h T1:P1:O2	21.008	14.003
▨ 15p 3h T1:P2:O2	20.579	13.123
□ 15p 3h T2:P2:O2	21.760	14.122

Figure 7.4: Training and playing on deeper ply depths.

against two-ply opponents, lead to the formation of the next question. Would it be possible to take advantage of the processing benefits of a single-ply training scheme (as has been followed for the majority of the computations in this study), evolve an intelligent game playing agent, and successfully apply its evaluation function in a larger game tree data structure? This ideally will provide the best of both worlds – fast training time on small ply-depths, and increased performance due to the extended ply-depth available during actual benchmarking/play.

The experimental results using the minimum configuration (15 particles, 3 hidden nodes, Von Neumann neighbourhood information sharing structure) is graphically depicted in figure 7.4. The first entry is a repeat from the previous results originally listed in figure 7.3, once again showing the weak performance of a player trained and playing on single-ply against a two-ply opponent. The second entry in this figure, however, answers the question quite easily. By extending the use of the evaluation function beyond its 'intended' application area (from single-ply to two-ply), the performance actually decreases against both the piece-count and SmartEval opponents.

It would seem that the evolved evaluation function is somehow tied to the original training depth, and that extended exposure to more board states provides an information overload to the originally focused single-ply version. This suspicion is confirmed with the final experiment conducted to address this question, with the results listed as the third entry in figure 7.4. Training a player on the correct ply-depth, ensures improved performance on that ply-depth, as is visible in the increased performance values for the small configuration trained on two-ply.

From these very restricted initial experiments, it is clear that the evaluation function evolved under the specific experimental conditions as set out in the beginning of the section, does not scale well for use in deeper ply-depths.

7.4.4 Q4: Do deeper training ply-depths require increased complexity?

The findings of the previous question sparked some hope that perhaps the poor performance of the evolved player was attributed to the wrongful handling of the evaluation function, and perhaps by training on deeper ply-depths a higher performance level can be achieved. Does an increase in ply-depth, and subsequent exposure to more board states, perhaps require a more complex configuration to properly assimilate the information while training?

This question is addressed by referring to the last entry in both figures 7.3 and 7.4. In each case the player was trained on a ply-depth of two, and subsequently benchmarked against the intelligent evaluation functions. The smaller configuration in figure 7.4 (15 particles, 4 hidden nodes, Von Neumann) scored an average of 21.760 against the piece-count based evaluation function, and 14.122 against SmartEval. In contrast to this – and simultaneously answering the question – the more complex configuration (20 particles, 25 hidden nodes, Von Neumann), also trained on two-ply depth and listed in figure 7.3, is able to improve on the performance quite substantially. The complex configuration scored 30.120 against the piece-count evaluation function, and 21.808 against SmartEval, illustrating the improvement.

Based on this and previous observations regarding ply-depth, an initial hypothesis would be to blame the specific combination of architecture complexity to ply-depth (both training and playing) for the poor performance results. Any definite conclusions in this regard will have to be more clearly investigated, and is left as future work resulting from this study.

7.4.5 Q5: Compressing game tree knowledge into a neural network

This last question can be seen as an inverse to question 3, and is not experimentally addressed in this study. The question revolves around the possibility of training a neural network as an evaluator on a much larger game tree structure and applying its 'expertise' using a smaller ply-depth – resulting in a compression of the knowledge into the black-box structure of a neural network. Assuming a best-case scenario (which for the moment seems highly unlikely), a neural network that approximates the principal variation of the game tree, should be able to provide the next move down this 'best pathway' without having to recompute and re-evaluate the other surrounding nodes.

This will allow the neural network to spend the minimum allocated time on node evaluation, improving performance under specific playing-time constraints. Even though the concept of 'compressing game tree knowledge' may seem far-fetched, it would be interesting to see if the observed unscalability of the evolved evaluation function repeats itself in this scenario, but only in reverse. In a way, this question ties up with the way humans play the game, only making use of shallow ply-depth calculations, but relying on an 'intuitive feel' of what a good move or strategy is composed of – based on vast exposure to previous games and training information, forming our own 'principal variation' of the game in our brain's biological network structures.

Naturally, the investigation into answering this daunting question is left for future work resulting from this study.

7.5 Observations

During play-testing of the custom implemented Checkers engine, and the subsequent 'bird's-eye view' of actual gameplay between evolved players at irregular intervals, a series of layman observations were made by the author.

The first observation comes from the final stages during most games – the so-called 'end-game'. It does happen at regular intervals that one player may achieve a significant piece-count advantage – outnumbering the opponent by for example five pieces to one. The dominant player will move on to crown all the regular 'men' in its possession, thereby indicating that it has learnt the importance of kings in the game.

The second observation continues in this scenario. The opponent player (vastly outnumbered) tries its best to avoid capture by the overwhelming force, making evasive moves and usually ending up in one of the opposite corners of the board.

After the dominant player has crowned most of its remaining pieces, it retreats to the 'safe' portion of the game board, and does not make any full-fledged commitments to find stray opponent pieces and capture them. One or two 'scout' pieces might be sent out from time to time, but a quasi 'cold war' scenario develops, ultimately resulting in a draw for both players as the allocated move-count expires.

During the middle game, it was interesting to note that the evolved players tried to keep the back rank in tact for as long as possible – corresponding to the feature identified in SmartEval. It is interesting to see that the players identified these playing strategies through coevolution alone, since playing against SmartEval does not allow the neural network-based agents to train any further.

The evolved players do however exhibit odd behaviour as well, sometimes unnecessarily sacrificing pieces for no visible reason (at least to the author). The general low performance scores exhibited towards the latter half of this chapter suggest that there is still some way to go before a robust and 'expert'-level player is evolved. As it does not form part of the main aims of this study, it is listed as future work resulting from this research.

7.6 Improving the training partner

Before closing the chapter on evolving Checkers playing agents, it is perhaps appropriate to discuss possible improvements that can be made in terms of the coevolutionary training approach. Up to now, this study has followed a purist's view on machine learning, whereby the experimental environment is completely void of any human intervention and any injected human intelligence apart from the game rules themselves.

The impact of the type and strength of the training partner has been studied before [47], with the search for the 'ideal trainer' instead pointing toward a perfect player, or highly advanced player – usually constructed through the application of human intelligence, and in direct conflict with the aforementioned purist's view of machine learning as applied in this study.

However, recent studies in coevolution has shown the tendency of populations to saturate on less-optimal solutions when only exposed to the peers they train against. Blair and Pollack [18] mentions one way of addressing this inherent drawback of the coevolutionary technique, is to provide a series of 'stepping options' for the agents to train against. As an agent attains a certain level of expertise, it is removed from the current training environment and exposed to more advanced playing partners – thereby allowing it to train and improve its performance once more. The construction of these 'stepping options', or improved intelligent opponents in these alternate training environments, more often than not closely involves human intervention. A series of hand-crafted evaluation functions can be used to construct these 'stepping options', by simply enabling and disabling certain advanced features of the evaluation functions in question. The specific features identified by the evaluation function do however remain the result of human intelligence, and the subsequent disabling of certain sections even more so.

Angeline and Pollack [6] made use of a variety of training partners, ranging from random players to expert players to evolve Tic-Tac-Toe playing agents. Training against a random player does not provide a too difficult challenge, and is quickly overcome – resulting on a stagnation of overall inferior players. Playing performance and game intelligence are increased when the evolving players are exposed to more capable opponents that require a more intelligent playing style in order to be beaten.

The experimental work conducted in this chapter involved training the evolving player against the piece-count based player when playing as player one, and training against the fellow members of the population when playing as player two. The results showed a definite biased increase in performance when playing as player one, but a decrease in performance when playing as player two. No in-depth research was conducted to find the cause of the lack of performance when playing as player two, but a 'too strong' opponent introduced in early training may be to blame. This shows the one advantage to pure population-based coevolution: a gradual increase in intelligent playing behaviour, allowing players to 'keep up' with the improved players, and not fail miserably by trying to beat an expert opponent from the start. Another reason for the imbalance in playing strength may lie in the scoring system, with a player relying on its strong 'player one' abilities to attain a good enough overall fitness, and never managing to improve its 'player two' playing strength due to the aforementioned lack of incentive.

This 'manual' approach to improving game playing behaviour has been studied in other games, such as Go, as well. For the moment, hand-made evaluation functions and game engines do not rival the human world champion Go players. Lubberts and Miikkulainen [91] evolved Go-playing agents by allowing the agents to compete against a respectable open-source Go engine. The agents were able to train and eventually outperform the hand-crafted engine, which resulted in no incentive to improve their playing performance thereafter. The drawback associated with coevolution is clearly displayed in this case, with stagnation occurring after the evolving population manages to reach a satisfactory playing level. The only problem in this case was the absence of any improved opponents (apart from human opponents) available to train against – a scenario that makes Go one of the most difficult game learning problems to solve, thereby representing the 'holy grail' for future research.

7.7 Conclusion

This chapter looked beyond the confines of the random-moving player benchmark, and presented two hand-crafted evaluation functions in order to properly quantify the playing strength of the evolved players. The neural network managed to outperform both the piece-count based and SmartEval evaluation functions on single-ply.

Extending the tree depth to two-ply resulted in poorer playing performance. A series of questions were posited in order to try and analyse the reasons behind the drop in performance, with very brief initial experimental results indicating the need for more in-depth research into the interaction of PSO configurations on deeper ply-depths.

A few layman observations on the playing style of the PSO-trained neural network players aimed to highlight some interesting playing behaviour, after which it was proposed that the coevolutionary training partner be changed.

This chapter concludes the experiments on two-player, perfect information games. The next chapter investigates the application of the PSO-based game learning approach to evolve intelligent players for use in the Iterated Prisoner's Dilemma – a two player, imperfect information, non-zero sum game.

CHAPTER 8

Iterated Prisoner's Dilemma

"Everyone complains of his memory; nobody of his judgement".
- La Rochefoucauld (1613 - 1680)

Building on the knowledge gained from experiments with zero sum games in the previous chapters, this chapter applies the PSO training technique to the interesting non-zero sum problem of the Iterated Prisoner's Dilemma (IPD). An overview of the problem and historic work is followed by an investigation of three different strategy generation approaches – all applying PSO in a different context. Experimental results indicate definite distinguishing characteristics of each method.

8.1 Introduction

Up to now this thesis focused on the applicability of PSO-based machine learning techniques on perfect information zero sum games. The notion of 'zero sum' games refers to their characteristically 'all-vs-nothing' scoring structure, where the winner has the highest payoff and the loser goes home empty handed. Non-zero sum games therefore represent the opposite standpoint[144], where both the winner and the loser can achieve some form of payoff. The prisoner's dilemma is an example of an imperfect information, non-zero sum game, in which the joint payoff is achieved by mutually cooperative behaviour.

In the previous experiments with Tic-Tac-Toe and Checkers it was crucial to evolve intelligent playing strategies that were able to beat the competition pool. For this specific application it is necessary to evolve competitive strategies that elicit mutually cooperative behaviour. Three different approaches to evolving strategies through the use of PSO are investigated. The first continues the method applied to Checkers and Tic-Tac-Toe – evolving neural network weights to enable an intelligent network to produce a strategy, given a set of historic decisions as input to the network. The second approach makes use of the lesser known Binary PSO (BinPSO) algorithm to directly evolve the strategy as discrete binary values (either cooperate or defect). The last method takes a novel approach to strategy generation by exploiting the symmetrical properties of well-known man-made strategies. In all of these cases, it is the first application of PSO to the IPD domain.

The rest of the chapter is organised as follows: An overview of the core IPD problem and relevant historic computer science-related work is presented in section 8.2. The well-known man-made strategies typically found in IPD competitions are described in section 8.3. The listed man-made strategies are subsequently compiled into a benchmark suite of strategies used to experimentally measure the performance of evolved strategies. The three different evolutionary processes to strategy generation are covered in section 8.5. Section 8.6 explains the experimental procedure followed for this study, and the exact results are analysed by various techniques in section 8.7. Section 8.8 concludes the chapter by summarising some of the major experimental findings.

8.2 Historic overview

8.2.1 The prisoner's dilemma

The mathematical foundations of perfect and imperfect information games were formalised in the definitive work on game theory by Von Neumann and Morgenstern [144] in 1944. The original example of a two-player perfect information game involves the fictional characters of Sherlock Holmes and his arch rival Moriarty - created by the author Sir Arthur Conan Doyle [33]. Holmes is trying to escape from Moriarty and boards a train to the coast. As his train pulls out of the station, he sees Moriarty boarding a second (and faster) train. On the railway track there is only a single halfway-house stop point, after which the coast (i.e. the final destination) is reached. Holmes has to decide whether to disembark half-way, or continue until the end of the line, keeping in mind that Moriarty also has these two options and will be able to make up for lost time with the faster train. The aim of the game for Holmes is to evade capture by Moriarty, and the objective for Moriarty is to close in on Holmes.

Von Neumann represents the possible payoffs for evasion and capture in a payoff matrix. The payoff matrix and its associated governing rules form the heart of the prisoner's dilemma. The prisoner's dilemma was originally invented around 1950 by Merril Flood and

Table 8.1: General form of the IPD payoff matrix.

	Cooperate	Defect
Cooperate	(ρ_1, ρ_1)	(ρ_2, ρ_3)
Defect	(ρ_3, ρ_2)	(ρ_4, ρ_4)

Melvin Drescher, and formalised by A. W. Tucker shortly thereafter [8]. As an example of an imperfect information game, the core concept behind the prisoner's dilemma is illustrated through the capture of two suspected felons by the police for committing the same crime. The felons are interrogated separately, and are each faced with one of two possible choices: either cooperate with the other felon (keep to their pre-planned story), or defect by reaching an agreement with the police. The felons have no prior knowledge of possible historic cooperative or defecting behaviour/trends, and are only informed of the other felon's decision after making their own. They are not allowed to change their decision, neither are they allowed to not make a decision at all.

Mutual cooperation by the felons is the most beneficial, and is 'rewarded' by assigning payoff ρ_1 to both felons. Defecting while the other felon expects cooperation, assigns the 'temptation' payoff ρ_3 to the defecting felon, and the 'sucker' payoff ρ_2 to the cooperating felon. Mutual defection is 'punished' by assigning a low payoff ρ_4 to both players. Table 8.1 depicts this general form of the payoff matrix.

A payoff matrix for the IPD is subject to two restrictions [8]:

i) $\rho_3 > \rho_1 > \rho_4 > \rho_2$

The first restriction lists a preference ranking for the payoffs, ensuring that mutual cooperation is more beneficial than mutual defection.

ii) $2\rho_1 > \rho_2 + \rho_3$

The second restriction dictates that mutual cooperation should be more beneficial than mutual exploitation (defecting when the other player cooperates and vice versa).

Table 8.2: Payoff matrix for the IPD as used by Axelrod.

	Cooperate	Defect
Cooperate	(3, 3)	(0, 5)
Defect	(5, 0)	(1, 1)

These two restrictions define the IPD. The final payoff used for either choice in the experimental work for this study is depicted in table 8.2, which is identical to the payoff matrix used by Axelrod [8].

The above description complies with the so-called *one-shot prisoner's dilemma*, in which each felon only makes a single decision before the matter is resolved by the police. In such a scenario, the best strategy would be to always defect, since the felon will either be awarded the biggest (temptation) payoff, or receive the equal punishment for mutual defection. The felon will never receive the lowest (sucker) payoff for foolishly expecting cooperation, and does not have to take into consideration any negative repercussions (retaliating behaviour from the other felon) for defecting.

It does however become a lot more interesting if the dilemma is extended beyond a single encounter (resulting in an *iterated prisoner's dilemma*), thereby causing the felons to not rely on the current encounter to be their last. If it is assumed that the felons know that they will meet a fixed number of times (for example 100 successive encounters), then the best strategy would be to defect on the very last move – as the last move can once again be regarded as a *one-shot prisoner's dilemma*. However, the players will anticipate defection on the last move for this exact reason, and thereby respond by defecting on the second-last move. Ultimately this reasoning will cause a chain-reaction of defection all the way back to the first move. This strategy is also referred to as the 'always defect' strategy in literature [8] and is the most stable strategy under the aforementioned conditions, resulting in a Nash equilibrium (i.e. it is the only outcome from which each player could only do worse by unilaterally changing its move [86]).

By looking at the restrictions defining the IPD payoff matrix, it is clear that mutual cooperation is the sought-after solution. An 'always cooperate' strategy however is always dominated by an 'always defect' strategy. In order to be successful in the IPD, the strategy should maximise the player's *average personal payoff* (through cooperation or exploitation) as well as maximising the *average combined payoff* (the sum of both players' payoff should approach the value for mutual cooperation). Section 8.5 deals with different ways to intelligently construct a strategy, and the aforementioned performance measures are discussed in more detail in section 8.6.1. It is perhaps now fitting to look at previous work in the field of the iterated prisoner's dilemma from a computer science perspective.

8.2.2 Related work

The IPD has been used to model cooperative behaviour (or lack thereof) in a variety of fields, including economics, politics, sociology, game theory and computer science. Due to its multidisciplinary application, an enormous volume of work has been produced [11] by researchers in these different fields. From a computer science perspective the definitive work is widely recognised to be the experiments conducted by Axelrod [8] in 1979 and the early 1980's.

Axelrod invited researchers from the wide range of application areas of the IPD to submit their strategies for a computer-based tournament. Various strategies were submitted, ranging in complexity from simple single history-based heuristics, to complex Bayesian network solutions. Much to the surprise of Axelrod and all the participants, the simplest strategy in the competition pool – tit-for-tat [10] (submitted by Anatol Rapoport) – managed to outplay the remainder of the submitted strategies. The same result was achieved when the experiment was repeated a couple of years later, even with a much larger variety of submitted strategies. Tit-for-tat (TFT) is based on reciprocity. It starts by cooperating on the first encounter, after which it repeats the decision the opponent made on the previous encounter. TFT's strength lies in its ability to facilitate mutual cooperative behaviour, in addition to its quick retaliatory response to defection. This strategy resembles the same behaviour followed by international superpowers during conflict and negotiations – once again illustrating the IPD's wide application base. In its traditional form however, TFT is not very successful in noisy environments. Other man-made strategies are discussed in section 8.3.

Recent computer science research has built on Axelrod's initial experiments to intelligently evolve effective IPD strategies [8] [9]. Darwen and Yao have focused on numerous aspects of the IPD problem, using coevolution to drive the evolution of robust strategies through genetic algorithms [35] (similar to Axelrod) and neural networks [37] [38]. Fogel worked with finite state machines (FSM) to represent IPD strategies [52], after which Harrald and Fogel replaced the FSMs with neural networks [65]. Fogel and Chellapilla continued to evolve neural networks to compete in the IPD [25], in addition to their aforementioned work on Checkers and Tic-

Tac-Toe. Some of Darwen and Yao's initial studies show that coevolutionary methods do mostly result in evolved populations that exhibit cooperative behaviour, but that by adapting to the 'moving target' introduced by this particular evolutionary scheme the robustness of the dominant strategy is not guaranteed [35]. Robustness is improved by forcing the population to play against a superior (but static) external strategy, while still competing against each other in the traditional round-robin fashion. Strategies evolved by this technique do not only perform well against the individuals in the population, but also perform much better against external (previously unseen) strategies.

Axelrod lists a variety of possible avenues of further research in his article [9] on the evolution of IPD strategies. One area involves the use of speciation in the evolutionary process. A thorough treatment of speciation (also referred to as 'niching' in popular literature) is once again beyond the scope of this thesis, but its applicability to the IPD as implemented by Darwen and Yao [36] is now briefly mentioned. GAs have been known to exhibit 'drifting' behaviour after prolonged periods of evolution, resulting in convergence on sub-optimal solutions. Speciation causes individuals in the population to form different 'clusters' in search space, converging on more than one optimum. In the context of the IPD, speciation results in several specialised strategies. Although each strategy is not immune to invasion, the application of speciation does increase the population's ability to successfully compete against a wider range of external strategies. Darwen and Yao applied the 'implicit fitness sharing' technique to construct niches for their GA-based population. By evaluating the external strategy's structure against the evolved individuals' structures, it is possible to categorise evolved strategies as either 'imitators' or 'imitator-answers' – the former closely resembling the external strategy, and the latter referring to strategies that can effectively compete against the external strategy. A gating algorithm is employed to categorise the strategies, after which a voting scheme is used among the 'imitator-answers' strategies to decide whether or not to cooperate or defect, based on the current interaction with the external strategy.

The basic two-player IPD can be extended in various ways. The first adaptation involves the introduction of noise to the interactions. Noise can be applied in two ways: Misinterpretation of the correct response (semantic error or noise), or correct interpretation of a distorted response (communication channel noise). For the experimental work conducted in this study, noise was added to the communication channel with varying probabilities, and its effects are examined in closer detail in section 8.6.2. The application of noise does not affect the size of the standard payoff for the actions taken by the players, as was done in earlier research by Bendor [86].

The second adaptation to the IPD involves a change to the payoff structure for each player. Firstly, the payoff values can be adjusted, while still adhering to the rules for the IPD as stated in section 8.2.1. This may result in a proportionally larger reward for mutual cooperation, and/or a proportionally smaller punishment for mutual defection, among other things. In order to ease comparison of work conducted in this study to work conducted by other researchers, the commonly referenced payoff matrix used by Axelrod [8] (depicted in figure 8.2) is used throughout this study.

The second adjustment to the IPD payoff structure allows for an increase in the size of the payoff matrix, thereby providing the players with more fine-grained levels of cooperation. Darwen and Yao have once again done extensive research in this area with, among other things, its specific application to missile defence [38]. They have also investigated the popular claims that genetic diversity (or the lack thereof) is responsible for convergence on cooperative (or non-cooperative) areas of the search space [37]. Their findings in this regard indicate that genetic diversity can in fact have an adverse effect on multi-choice IPD games. Highly diverse populations revert to participating in the traditional two-choice IPD (still represented by the outside corners of the multi-choice IPD), and avoid making more fine-grained choices – the very characteristic that makes multi-choice IPD so attractive, yet also difficult to 'solve'.

The last variation of the standard IPD discussed in this section involves the increase of the number of players in the immediate competition pool, commonly referred to as the n-player IPD [86]. It can be illustrated through the 'tragedy of the commons' as popularised by Hardin [64]. Neighbouring farmers all have access to the same public grazing land – referred to as a 'commons'. The limited resources of the commons allow each farmer to have a single cow graze on the land. In order to allow more of their own cows onto the commons, the farmers each have to pay a fee. Cooperation comes down to paying the fee and getting the benefit, while defecting implies sneaking an extra cow onto the commons without paying, but still getting the benefit. The temptation payoff is awarded if the player is not 'caught' sneaking an extra cow onto the commons, and the sucker payoff is rewarded to all the paying farmers for letting this happen. The reward for mutual cooperation is awarded when all the farmers pay their fee, and complete defection is punished when all the farmers sneak an extra cow onto the commons – thereby depleting the limited resource. The n-player IPD is still a very active research area, with a lot of potential application areas.

The standard two-player IPD is used in the experimental work for this study, with the single adaptation being the addition of noise in later experiments. Coevolutionary training similar to that of Chellapilla and Fogel [25], and Darwen and Yao [37][38][36][35] is employed to drive the evaluation of different strategies. The investigation of the application of PSO to some of the other adapted IPD paradigms is left for future work. An initial examination on the first two strategy generation approaches culminated in a conference paper, available in [57]. A more detailed treatment of the work presented in the paper is provided below.

8.3 Choosing an opponent

Experiments with human opponents participating in an IPD against each other have yielded interesting results in both psychology and political science [8] [11]. A detailed discussion on this interesting topic is beyond the scope (or even domain) of this thesis, but a very short description may prove to be useful. Humans are inherently selfish creatures, yet altruistic behaviour is frequently exhibited – especially among kin. Researchers have conducted studies among more than 500 pairs of identical and fraternal twins [150], measuring traits such as altruism, empathy and nurturance. Interestingly enough their findings show that on all three traits women score higher than men, and older individuals score higher than younger individuals. Furthermore, identical twins tend to be similar on the three traits, but fraternal twins tend to be different. The researchers attribute the findings on altruism to be approximately 50% due to genetic influence, 50% due to individual environmental effects, and only close to 2% due to the twins' home environment [150].

Humans are also emotional creatures, more often than not relying on moral judgement rather than rational thought. Rapoport makes mention of a couple of key concepts in a book review on this topic [107]. In a prisoner's dilemma, the difference in payoff for exploitation and mutual cooperation may lead a human to either make a decision of moral importance (mutual cooperation for the advantage of the group) or for personal benefit (exploitation of other players to increase personal payoff). Researchers have noted that individuals may often cooperate even in a one-shot prisoner's dilemma, contradicting the obvious rational choice of defection. Rapoport continues by stating that by using game theory to evaluate typical rational decisions in two-player non-cooperative games (such as the IPD) is similar to making assumptions about 'perfect conditions' in physics. The irony lies in the fact the physicists may be able to closely duplicate a perfect environment for a given experiment or state, but it is highly unlikely that the majority of IPD players will act 'perfectly' – i.e. completely rational. Therein rests the essence of the research problem for the various application areas of the IPD: to understand human behaviour and why certain actions are taken in contradiction to established scientific knowledge.

In order to ease experimental work in this study, humans are not used as primary opponents during training. Instead, some of the man-made strategies that participated in Axelrod's experiments [8] – most of which form part of the common IPD research base available to scientists today – were chosen to form part of a 'benchmark suite' of strategies. Continuing with the reasoning of section 3.3 on consistent players, a random player is also included – incidentally in accordance with Axelrod's own experimental procedure. The benchmark suite is used to evaluate the performance of the best strategy from the competition pool after competitive coevolutionary training has completed. The man-made strategies that make up the benchmark suite are described below:

- **Always Cooperate (ALLC)**: This is the most naïve strategy in the collection, and cooperates on every move.
- **Random (RAND)**: This player makes use of a randomly generated strategy, which generally is quite competitive.
- **Pavlov (PVLV)**: Designed to improve performance in noisy environments, Pavlov repeats its previous decision if it was profitable (ρ_1 or ρ_3), but changes its decision if it wasn't (ρ_2 or ρ_4).
- **Tit-for-tat (TFT)**: The very famous strategy submitted by Anatol Rapoport, starts with a move to cooperate, but thereafter repeats the last move made by its opponent.
- **Suspicious Tit-for-tat (STFT)**: The same as TFT, but starts with a defection.
- **Tit-for-two-tats (TFTT)**: Also invented to improve performance under noisy conditions, it has the same strategy as TFT, but only defects after two successive opposition defections.

8.4 Training algorithm

In order to train the population of individuals to participate in the IPD, the coevolutionary training scheme developed in chapter 3 for use with Tic-Tac-Toe and Checkers is marginally adapted. The complete algorithm is listed in figure 8.1. The most significant changes are due to the non-zero sum game environment and the intricacies of the IPD itself. This includes the absence of a 'board state', which is replaced by the historical position, the inclusion of a random initial state (historic position), and a fixed encounter time of 151 moves. The performance measures (discussed in more detail in section 8.6.1) differ quite substantially as well.

```
1) Instantiate population of agents.
2) Repeat for 500 epochs:
   A) Add each agent's personal best particle structure to the population.
   B) For each individual in the population:
      i) Play against every other player in the population.
         a) Determine a random historical starting position.
         b) Exchange cooperate/defect decisions for 151 iterations.
         c) Determine payoff using payoff matrix.
      ii) Keep track of total personal payoff.
   C) Compute best performing particle according to PSO algorithm in use.
   D) For each agent (excluding personal best) in the population do:
      i) Compare performance against personal best configuration.
      ii) Compare performance against neighbourhood's best particle.
      iii) Update velocity for each particle according to PSO algorithm.
      iv) Update position according to PSO algorithm.
3) Determine single best performing agent in whole population.
4) Best agent plays 10000 games against each of the 6 benchmark strategies.
6) Return to step 1 until 20 simulations have been completed.
7) Compute performance value over the 20 completed simulations.
```

Figure 8.1: Complete IPD training algorithm.

8.5 Strategy generation

As was mentioned in section 8.2.2, various techniques have been used to generate strategies for the IPD, ranging from man-made strategies [8] to genetic algorithms [35] and evolved neural networks [37] [38] [25] [65]. Axelrod identified four different key features to successful IPD strategies in his book [8], namely:

i) Do not be envious.

ii) Do not be the first to defect.

iii) Reciprocate both cooperation and defection.

iv) Do not be too clever.

In order to address these features, a player must have access to some historical information about previous encounters against a particular opponent. Axelrod and other prominent researchers have allowed a player to keep track of its last three personal decisions, as well as the last three decisions made by its opponent. This accounts for 2^6 possible historical states, resulting in a standard strategy containing 64 possible cooperate/defect actions. It is the aim of the evolutionary process to generate a 64-bit string that will represent a competitive strategy in the constantly changing coevolutionary training environment.

The following sections each approach the process of strategy generation using PSO from a different perspective. Evolving neural networks to evaluate historical positions and thereby generate a successful playing strategy is discussed in section 8.5.1. The direct evolution of the 64-bit string is achieved by the lesser known Binary PSO algorithm, and is discussed in more detail in section 8.5.2. The last approach makes use of the PSO as function optimiser to generate one half of a strategy, after which it is mirrored using one of four techniques to form a complete strategy. This exploitation of symmetry is covered in section 8.5.3.

8.5.1 Evolving neural networks

Section 1.4.3 provided some background information on the use of neural networks in zero-sum games, with specific reference to its applicability as a game state evaluator. Earlier experimental work in this thesis also showed its applicability to Tic-Tac-Toe and Checkers. The same neural network technique is now applied to the IPD.

Fogel and Chellapilla's [25] neural network consisted of six input units, a hidden layer with between two and twenty hidden units and a single output unit. The first three inputs to the network represented the last three moves made by the opponent, and the final three inputs represented the last three outputs by the network itself. The hidden and output nodes made use of the hyperbolic tangent activation function. The single output represented a decision to either cooperate or defect. The network was trained through the evolution of the network weights, and performance measured through a coevolutionary tournament scheme.

Darwen and Yao [37] [38] constructed a slightly different neural network, consisting of four input nodes, ten hidden nodes and once again a single output node. The first two inputs correlated with the player's last move as well as its opponent's last move. The last two inputs indicated whether the player exploited the opponent in the last move, or if the player was exploited by the opponent instead. Ten hidden units proved to work well for the authors, and the hidden and output layers also made use of the hyperbolic tangent as activation function. Coevolutionary training was once again successfully employed.

This thesis' experimental approach is similar to the method employed by Fogel and Chellapilla. A standard three-layer artificial neural network is constructed, with six input units, a selection of between five and thirty hidden units, and closing with a single output unit. The first three input units receive the last three decisions output by the network itself, while the last three input units accept the last three decisions made by the opponent. Sigmoid activation functions are used in all the hidden and output units. A final output value less than 0.5 represents cooperation, while a value larger than or equal to 0.5 represents defection.

Each particle represents the complete weight vector (input-to-hidden augmented by hidden-to-output weights) of the neural network, and training is accomplished by 'flying' the particles through the problem space towards the neighbourhood's best position. The LBest neighbourhood structure is used, due to its good performance record for training neural networks – as established in the previous chapters. A coevolutionary training approach is followed once again, and described in more detail in section 8.4.

After training has been completed, the best individual in the competition pool is selected and 'queried' for its response to all 64 possible historic interactions. The output is recorded as a 64-bit string and benchmarked against the suite of existing man-made strategies.

Parameter selection

Taking into account the valuable insight gained from the performance analysis of using ANNs to play Checkers, the following decisions were made regarding parameter selection. The relationship between an increase in swarm size and an increase in hidden-layer size will once again be investigated. The most prominent swarm sizes from the earlier work on Checkers – 20 and 40 particles respectively – are similarly applied to this problem. Due to the success of the LBest PSO neighbourhood structure in chapter 6, it is applied to train ANNs here using an inclusive neighbourhood size of 5 particles.

It is also worth re-stating that the PSO algorithm requires a personal best score comparison. The only accurate way of measuring any previous best playing behaviour is by including each particle's best position as an extra player in the competition pool. This results in the doubling of the swarm size in order to accommodate the new players. The swarm sizes listed for the experimental work *exclude* the personal best particles.

The influence of the maximum velocity on particle behaviour is expected to be even greater in the IPD domain, due to its limited search space. Experimental testing has shown a maximum velocity of 4.0 proved to elicit continued runs of cooperative behaviour. A proper inertia value was also experimentally determined to be 0.7. The cognitive (c_1) and social (c_2) acceleration constants are set to 1.4 each, while still adhering to the convergence equation by Van den Bergh [137] as given in section 2.4.2. The particles are initialised according to the ranges established by Wessels and Barnard [146] as described in section 2.2.1 on neural network weight initialisation, and defined by equation 2.1.

The results of the experimental tests for the remainder of the parameter selections are discussed alongside other experimental results in section 8.7.

8.5.2 Evolving strategies through Binary PSO

The second approach is more closely tied to the original work conducted by Axelrod [8], and subsequent implementations by Darwen and Yao [35] [36]. Axelrod made use of a history-bound of three decisions, allowing for 64 possible cooperate/defect interactions – represented as a binary strategy. The 64-bit string was augmented with 6 additional bits to indicate the three first decisions, bringing the final bit length for the chromosome representation to 70. Axelrod applied standard evolutionary computation constructs, such as selection, reproduction and mutation to a population of strategies in order to evolve more competent individuals. Each strategy's fitness was computed by playing an instance of the IPD for 151 interactions against each of the fellow strategies in the population.

As was mentioned in the background chapter on computational intelligence techniques (in section 2.4.4), Kennedy and Eberhart [77] developed a variant of the PSO algorithm to work in a discretized search space – called Binary PSO (BinPSO). This method allows for a particle to consist out of a string of binary values that are evolved by 'flying' the particle through search space. Each particle represents a complete strategy of 64 cooperate/defect values (represented in binary as 0 and 1 respectively). No additional bits are augmented to the strategy, as training requires the players to start from random historic positions – thereby forcing the strategies to cope with *any* historic position. This makes the IPD a slightly more difficult problem to train on initially, but results in more robust strategies being evolved over 500 epochs. Coevolutionary training is once again employed, with a more detailed description presented in section 8.4.

Parameter selection

In comparison to the PSO approach to train neural networks, the BinPSO approach requires a little bit more investigation to adequately determine its parameters – since no previous work has been done to this effect. The three different information sharing neighbourhood structures introduced in chapter 2 are once again compared to determine their individual efficiency in eliciting cooperative behaviour. The influence of applying an inertia weight to the standard PSO velocity update equation is experimentally determined between the ranges (0.5, 1.0). Similar testing correctly identifies the proper maximum velocity to be used for the BinPSO simulations, with values compared between the ranges (0.001, 10.0).

Further experimental analysis investigates the influence of an increase in swarm size for all the different neighbourhood structures. The cognitive (c_1) and social (c_2) acceleration values are each set to 1.0, while still adhering to the equations for convergence as specified by Van den Bergh [137]. In order to continue to make proper use of the coevolutionary training scheme, the competition pool is doubled to facilitate the use of the personal best player.

The results of the experimental tests for the remainder of the parameter selections are discussed alongside other experimental results in section 8.7.

8.5.3 Evolving strategies by exploiting symmetry

While experimenting with the previous two applications of PSO to the IPD domain, an interesting observation regarding the physical structure of the IPD strategies triggered the creation of yet another approach to evolve new IPD strategies through PSO. Table 8.3 lists the various man-made strategies in tabular form, given the 64 possible historic states. To ease the explanation, each strategy is depicted in two adjacent tables with its 'top half' (first 32 bits) on the left and 'bottom half' (last 32 bits) on the right. By looking at the TFT and TFTT strategies, it is possible to see that the bottom halves are exact copies of the top halves. The PVLV strategy presents a mirror image, pivoted around the 32^{nd} bit. This 'symmetrical property' of the two successful man-made strategies seemed worthwhile investigating.

By only considering the first 32 bits, it is possible to represent the TFT strategy as a sinusoidal wave (depicted in figure 8.2), with the positive areas of the graph representing defection (1), and the negative areas of the graph representing cooperation (0). By slightly shifting the sinusoidal graph in the horizontal and vertical directions, it is possible to represent the TFTT strategy as well (depicted in figure 8.3).

Up to now, the most prominent application of PSO techniques has been in the area of function optimisation. Figures 8.2 and 8.3 already illustrate how easy it is to create a strategy by adapting a simple sinusoidal function. Would it be possible to construct a 'higher resolution' mathematical function that will allow for more intricate arrangements of 0's and 1's, and adapting this function through the use of PSO?

The need for simplicity steered the creation of a combined sin/cos function. The function can be mathematically described as:

Table 8.3: Binary representation of strategies, illustrating symmetrical properties. ('Op' refers to the opponent, and 'My' to the local player)

Top half of strategy										Bottom half of strategy									
Bit	My history			Op history			TFT	TFTT	PVLV	Bit	My history			Op history			TFT	TFTT	PVLV
1	C	C	C	C	C	C	0	0	0	33	D	C	C	C	C	C	0	0	1
2	C	C	C	C	C	D	0	0	0	34	D	C	C	C	C	D	0	0	1
3	C	C	C	C	D	C	0	0	0	35	D	C	C	C	D	C	0	0	1
4	C	C	C	C	D	D	0	0	0	36	D	C	C	C	D	D	0	0	1
5	C	C	C	D	C	C	1	0	1	37	D	C	C	D	C	C	1	0	0
6	C	C	C	D	C	D	1	0	1	38	D	C	C	D	C	D	1	0	0
7	C	C	C	D	D	C	1	1	1	39	D	C	C	D	D	C	1	1	0
8	C	C	C	D	D	D	1	1	1	40	D	C	C	D	D	D	1	1	0
9	C	C	D	C	C	C	0	0	0	41	D	C	D	C	C	C	0	0	1
10	C	C	D	C	C	D	0	0	0	42	D	C	D	C	C	D	0	0	1
11	C	C	D	C	D	C	0	0	0	43	D	C	D	C	D	C	0	0	1
12	C	C	D	C	D	D	0	0	0	44	D	C	D	C	D	D	0	0	1
13	C	C	D	D	C	C	1	0	1	45	D	C	D	D	C	C	1	0	0
14	C	C	D	D	C	D	1	0	1	46	D	C	D	D	C	D	1	0	0
15	C	C	D	D	D	C	1	1	1	47	D	C	D	D	D	C	1	1	0
16	C	C	D	D	D	D	1	1	1	48	D	C	D	D	D	D	1	1	0
17	C	D	C	C	C	C	0	0	0	49	D	D	C	C	C	C	0	0	1
18	C	D	C	C	C	D	0	0	0	50	D	D	C	C	C	D	0	0	1
19	C	D	C	C	D	C	0	0	0	51	D	D	C	C	D	C	0	0	1
20	C	D	C	C	D	D	0	0	0	52	D	D	C	C	D	D	0	0	1
21	C	D	C	D	C	C	1	0	1	53	D	D	C	D	C	C	1	0	0
22	C	D	C	D	C	D	1	0	1	54	D	D	C	D	C	D	1	0	0
23	C	D	C	D	D	C	1	1	1	55	D	D	C	D	D	C	1	1	0
24	C	D	C	D	D	D	1	1	1	56	D	D	C	D	D	D	1	1	0
25	C	D	D	C	C	C	0	0	0	57	D	D	D	C	C	C	0	0	1
26	C	D	D	C	C	D	0	0	0	58	D	D	D	C	C	D	0	0	1
27	C	D	D	C	D	C	0	0	0	59	D	D	D	C	D	C	0	0	1
28	C	D	D	C	D	D	0	0	0	60	D	D	D	C	D	D	0	0	1
29	C	D	D	D	C	C	1	0	1	61	D	D	D	D	C	C	1	0	0
30	C	D	D	D	C	D	1	0	1	62	D	D	D	D	C	D	1	0	0
31	C	D	D	D	D	C	1	1	1	63	D	D	D	D	D	C	1	1	0
32	C	D	D	D	D	D	1	1	1	64	D	D	D	D	D	D	1	1	0

$$f(x) = sin(2\pi(x-a) \times b \times cos(2\pi \times c(x-a))) + d \tag{8.1}$$

where the variable a represents the horizontal shift of the function, b represents the maximum frequency of the sin function, c represents the frequency of the cos function (in effect the rate of change for the sin frequency), and lastly d represents the vertical shift of the function. The resultant function with default coefficients is depicted in figure 8.4.

The power of the PSO as function optimiser can be put to the test by allowing it to adjust the aforementioned function parameters (a,b,c,d). The resultant function is evenly sampled at 32 positions and evaluated to either represent a cooperate or defect decision. In order to fully exploit the symmetrical properties of the new IPD strategies, a 5^{th} parameter is added to the set of PSO evolvable terms. The new parameter, ψ, dictates the exact method by which symmetry should be created, and its use is described in more detail below.

The 32 sampled bits represent the 'top half' of the IPD strategy. These bits can be ma-

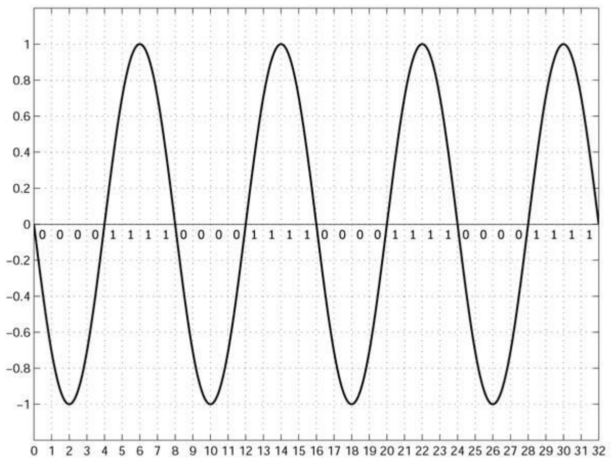


Figure 8.2: First 32 bits of the TFT strategy represented as a sinusoidal wave.

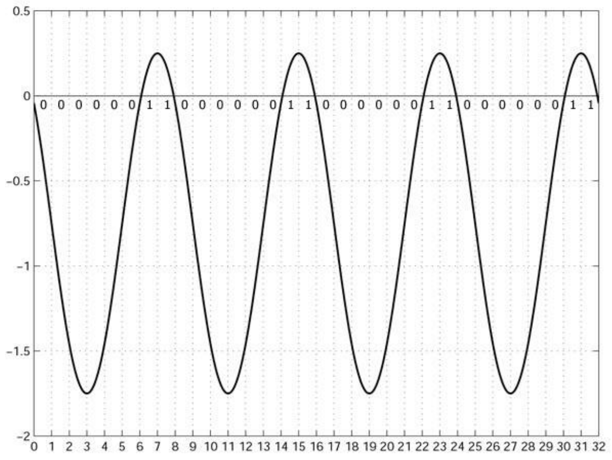


Figure 8.3: First 32 bits of the TFTT strategy represented as an adjusted sinusoidal wave.

nipulated in a variety of ways to construct a full 64-bit strategy, as depicted in figure 8.5. The original 32 bits can be copied onto the previously empty 'bottom half' of the strategy, thereby creating symmetry by repetition as applied by the TFT and TFTT strategies. Secondly, the original bits can be mirrored around the 32nd bit, as illustrated by the PVLV strategy. The last two symmetry operations repeat these methods, but instead make use of the compliment of the original 32-bit string.

The evolved symmetry parameter, ψ, is first fed through a sigmoid function to scale it into the (0, 1) range. The resultant value is compared to fall within one of four evenly spaced intervals between 0 and 1, each representing a specific symmetry construction method. The final 64-bit strategy is compiled thereafter, and used in a coevolutionary training scheme to measure the individual's performance.

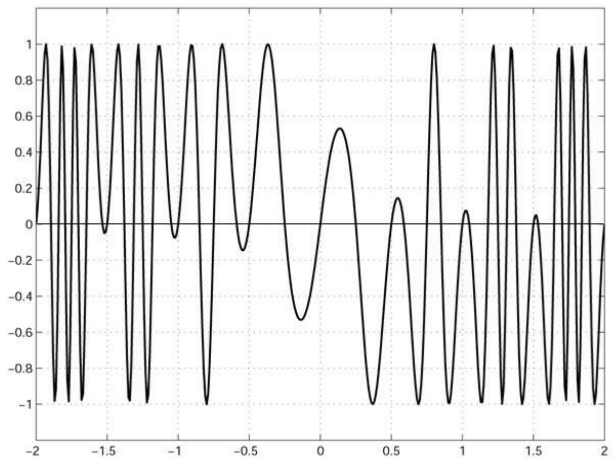


Figure 8.4: Combination of sin and cos function to be optimised.

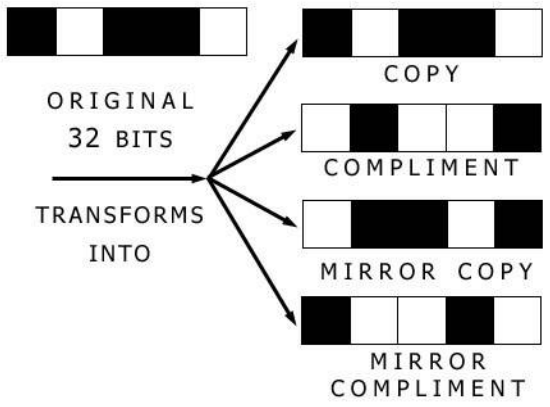


Figure 8.5: Constructing symmetry in four different ways.

Parameter selection

Since this approach to IPD strategy generation is quite unique, an array of experimental tests similar to those required for the BinPSO approach are necessary to determine the influence of various PSO parameter settings. The three different information sharing neighbourhood structures are once again compared to determine their individual efficiency in eliciting cooperative behaviour. The value of the inertia weight is experimentally examined within the range (0.5, 1.0). The influence of the maximum velocity value is tested within the range (0.001, 10.0). The effect of different swarm sizes is once again be examined for all the different neighbourhood structures. The cognitive (c_1) and social (c_2) acceleration values are set to 1.0 each. In order to continue to make proper use of the coevolutionary training scheme, the competition pool is doubled to facilitate the use of the personal best player.

The results of the experimental tests for the remainder of the parameter selections are discussed alongside other experimental results in section 8.7.

8.6 Experimental procedure

Before critically examining the experimental results, it is first necessary to discuss the various methods of determining a strategy's performance in relation to other individuals in the population. Section 8.6.1 describes the performance measures introduced for experimental work on the IPD for this thesis. The population training scheme is clearly outlined in the training algorithm presented in section 8.4. Finally, the introduction of noise into the experimental environment is covered in section 8.6.2.

8.6.1 Measuring performance

As was mentioned in the historic overview of the IPD (section 8.2.1), each player participating in the IPD aims to maximise two separate strategy performance measures, namely the player's *average personal payoff* and *average total payoff*. It is also necessary to evaluate the population behaviour in order to measure the population's ability to cooperate for a finite length of time. The following subsections discuss these separate performance measures in more detail.

Population behaviour

The IPD payoff matrix restrictions clearly indicate that mutual cooperation is more beneficial than mutual defection or exploitation. The logical next step in terms of performance measurement would be to quantify the population's cooperative behaviour. From experimental analysis, the following relation seems to be a good indication that a cooperative state is approaching:

$$(\beta \wedge \tau) > (10 \times populationSize) \tag{8.2}$$

where β represents the total number of cooperations for the individual with the largest total payoff, and τ represents the average total number of cooperations for the population – both for a single epoch.

Thus, the first measure of performance totals the number of simulations that satisfied the above relation, indicating the total number of epochs during which population cooperation was achieved. Secondly, it is important to determine the longest uninterrupted session of cooperation during which the above relation held (longest stretch of cooperative behaviour across all simulations).

Table 8.4: 'Man-made' strategy benchmark – no noise.

Strategy	**Personal Payoff**	**Opponent Payoff**	**Total Payoff**
RAND	2.68756	2.06250	4.75007
PVLV	2.76425	2.21621	4.98046
ALLC	2.49580	3.33613	5.83193
TFT	2.45963	2.45480	4.91443
STFT	2.46007	2.45524	4.91531
TFTT	2.54840	2.89081	5.43921

Strategy performance

It is also important to measure the performance of an individual strategy in relation to the whole population. This is accomplished by competing an evolved strategy against the benchmark suite of strategies listed in section 8.3, and allows each of the strategies in the population to be tested independently.

The first performance measure inferred from the benchmarking process is the need to maximise the individual's *average personal payoff*. A high average personal payoff indicates that the player is cooperating and maybe even briefly exploiting some of its opponents. The second performance measure links up with the population performance measure, whereby the *average total payoff* is calculated. A high average total payoff indicates that not only the current strategy, but also its opponents are achieving high payoffs – hopefully due to mutual cooperation.

Table 8.4 lists the various performance measures as calculated for each of the benchmark strategies, by playing 1 million games against each of the various benchmark opponents in a round-robin fashion. Each encounter consisted of 151 mutual interactions (corresponding to the average number of interactions used by Axelrod [8]), and started by assuming a random historical position. The Rand3 pseudo-random number generator developed by Knuth [105] was used for consistency.

Experimental analysis will examine how the numbers in table 8.4 change as more competitively evolved strategies are introduced, as well as how the competitive strategies' payoffs compare to the benchmark results. The benchmark strategies work well in mutually cooperative environments, so the aim of evolved strategies would be to improve on the benchmark *average personal payoff*, while still maintaining a reasonable level of cooperation – measured through the *average total payoff*.

Table 8.5: 'Man-made' strategy benchmark – 5% noise.

Strategy	Personal Payoff	Opponent Payoff	Total Payoff
RAND	2.64382	2.08117	4.72499
PVLV	2.82916	2.10013	4.92928
ALLC	2.43957	3.27225	5.71183
TFT	2.53880	2.46150	5.00030
STFT	2.53900	2.46180	5.00080
TFTT	2.49307	3.10632	5.59939

8.6.2 Noisy environments

Real-world situations all have a certain degree of noise, be it from faulty sensory equipment or just a simple misunderstanding. Noise can also be added to the IPD to determine whether or not a population of individuals can still arrive at a mutual cooperative agreement. Certain strategies such as *Pavlov*, *Generous Tit-for-tat* and *Tit-for-two-tats* have been hailed to be more effective in noisy environments. Wu and Axelrod [148] summarise three options to adequately deal with noise:

i) Add generosity to a reciprocating strategy.

ii) Add contrition to a reciprocating strategy.

iii) Make use of a new strategy (Pavlov).

Work by Foster and Young [149] has shown that the behaviour of a population consisting only out of ALLC, TFT and Always Defect strategies tend to change as soon as noise is added to the system. Randomness is applied according to the central limit theorem, which states that an increase in population size diminishes the effects of individual random behaviour – noise is therefore applied inversely proportional to the population size. Foster and Young observe that in large population sizes (and small noise), individuals tend to play Always Defect, and conclude that cooperative strategies should be constantly tested by non-cooperative strategies in order to maintain cooperation in the long run.

The performance of the various evolutionary processes are experimentally evaluated under conditions with 5% noise added to the communication channel, irrespective of the population size. The benchmark strategy performance under 5% noise is listed in table 8.5, computed by playing each benchmark strategy against all its neighbours in a round-robin fashion for 1 million games, 151 interactions per encounter.

8.7 Experimental results

This section systematically presents a wide range of experimental results as alluded to in earlier sections of this chapter. In summary, the following experiments were conducted:

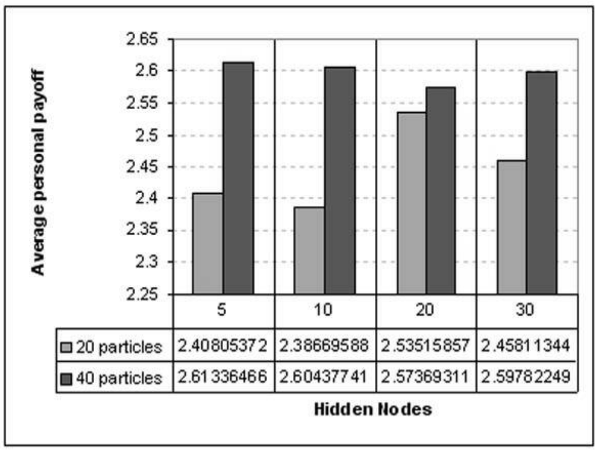
1. Influence of parameter choices
   - Neural Network approach
     - Increase in hidden nodes
     - Increase in swarm size
   - Binary PSO and Symmetry Approaches
     - Influence of VMax
     - Influence of Inertia
     - Influence of Neighbourhood structure
     - Increase in swarm size
2. Population performance (All approaches)
   - Benchmark results
   - Benchmark results with noisy environment
3. Strategy performance (All approaches)
   - Benchmark results
   - Benchmark results with noisy environment

8.7.1 Parameter choices

The following subsections list results for the experiments conducted to determine the appropriate parameter settings, before any population performance or strategy benchmarking are done. The Neural Network approach is examined first, after which the BinPSO and Symmetry approaches are examined in unison.

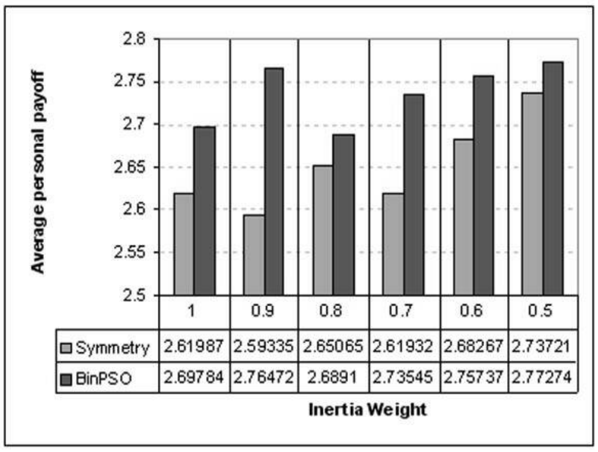
Neural Network approach

The first task is to determine the influence of an increase in hidden nodes, versus an increase in swarm size on the performance of the ANN to generate successful strategies. To recap, the following parameter settings were previously experimentally determined to result in optimal performance: VMax $= 4.0$, c_1 and $c_2 = 1.4$, and inertia $= 0.7$. Figure 8.6 graphically depicts the



	5	10	20	30
▨ 20 particles	2.40805372	2.38669588	2.53515857	2.45811344
■ 40 particles	2.61336466	2.60437741	2.57369311	2.59782249

Figure 8.6: Increase in hidden nodes versus swarm size for Neural Network approach.



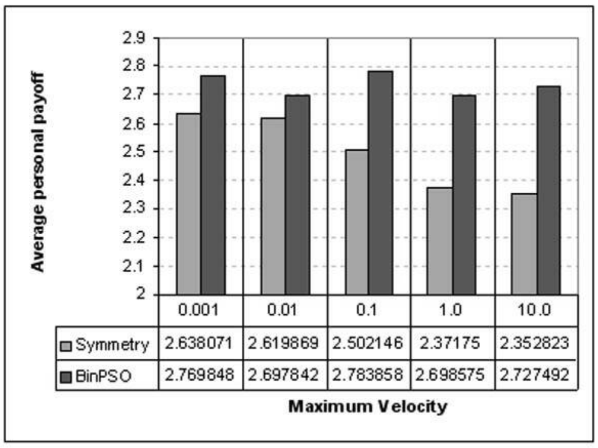
	1	0.9	0.8	0.7	0.6	0.5
▨ Symmetry	2.61987	2.59335	2.65065	2.61932	2.68267	2.73721
■ BinPSO	2.69784	2.76472	2.6891	2.73545	2.75737	2.77274

Figure 8.7: Influence of inertia term for BinPSO and symmetry approaches.

results using the aforementioned parameters, clearly indicating that an increase in swarm size has a significant improvement on the *average personal payoff*. Smaller swarms' performance generally improves with an increase in hidden nodes, whereas the potential gain to be made by larger swarm sizes in increasing their hidden layer size is marginally smaller. These numbers represent the strategies' *average personal payoff* against the benchmark strategies listed in section 8.3.

Binary PSO and Symmetry Approaches

The Binary PSO and Symmetry approaches require slightly more experimental analysis in order to determine the exact impact of specific parameter choices on playing performance. Previous experimental testing showed that setting c_1 and c_2 to 1.0 each resulted in adequate converging



	0.001	0.01	0.1	1.0	10.0
▨ Symmetry	2.638071	2.619869	2.502146	2.37175	2.352823
■ BinPSO	2.769848	2.697842	2.783858	2.698575	2.727492

Figure 8.8: Influence of the maximum velocity for BinPSO and symmetry approaches.

behaviour. Specific neighbourhood structure performance will be analysed in section 8.7.3, along with an increase in swarm size. For the experiments conducted in this section, the LBest neighbourhood structure was used. Experimental work was conducted to examine the influence of a static inertia term and maximum velocity for both approaches. No noise was applied in any of the parameter-specific experiments.

Figure 8.7 graphically depicts the results for experiments that tested inertia terms ranging from 1.0 to 0.5, in decrements of 0.1. For this experiment, an arbitrarily chosen maximum velocity of 0.01 was used. The results for both the Symmetry and BinPSO approaches show an increasing trend in average personal payoff as the inertia decreases, with the best performance being achieved with an inertia value of 0.5 in both cases. The best performing Symmetry approach managed to score 2.73721, while the overall best BinPSO approach scored 2.77274.

The influence of the use of a strictly capped maximum velocity term was also examined. For these experiments, an inertia term of 1.0 was chosen in order to balance the local and global searching capabilities of the swarm. A varying degree of maximum velocities were experimentally tested, including 0.001, 0.01, 0.1, 1.0, and 10.0. The results for these experiments are graphically depicted in figure 8.8. In this case, the two strategy generation approaches do not follow the same trend. The Symmetry approach experiences a decline in performance as the maximum velocity restriction is relaxed – a trend previously noticed with neural network training of Checkers agents. The BinPSO approach however doesn't seem to clearly benefit from a certain range of maximum velocities, showing some signs of oscillatory behaviour. Overall, the best Symmetry approach scored an average personal payoff of 2.638071 with a maximum velocity of 0.001, while the best BinPSO approach scored 2.783858 with a maximum velocity of 0.1.

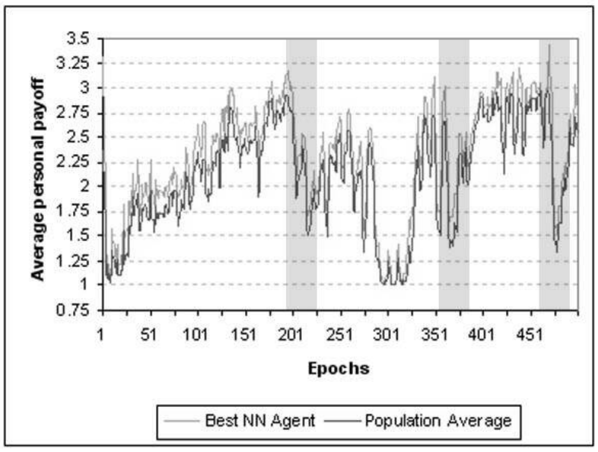


Figure 8.9: Catastrophic collapses occurring in neural network-based population.

8.7.2 Population performance

As was mentioned in section 8.6.1, the first primary measure of performance involved investigating the performance and behaviour of the population of IPD players. Ideally, a population should be able to enter into continuous mutual cooperative exchanges, hereafter referred to as 'cooperative runs'. The following subsections measure a certain strategy generation approach's ability to initiate a cooperative for each of its 20 simulations, as well as the longest cooperative run that emerged across all the simulations. The average personal payoff for the best individual in the population is plotted for the duration of the 500 epochs to indicate possible switches in strategy and subsequent shifts in performance.

The first subsection investigates a selected observation using the Neural Network approach – the so-called 'catastrophic collapses' occurring in noiseless environments. Thereafter, normal analysis continues in sections dedicated to noiseless and noisy environments respectively.

Neural Network approach - catastrophic collapses

In a noiseless environment using the Neural Network approach, it frequently occurred that the swarm did not maintain cooperation when the 500th epoch approached, but instead showed signs of 'burst cooperative runs'. A population may be improving its overall score by continually cooperating with fellow particles, until the end result is a genetically weak population that is easily invaded by an 'always defect' strategy. This leads to so-called *catastrophic collapses*, first formally observed by Lindgren [87], it also appeared in similar neural network-based work by Chellapilla and Fogel [25]. The reasons behind these collapses were critically examined by Darwen and Yao [35], and are briefly mentioned below.

Figure 8.9 illustrates this exact phenomenon from a population with a configuration of 40 particles and 10 hidden nodes. As soon as the *average personal payoff* approaches a value of 3.0, followed by a similar average population payoff (indicating mutual cooperation among the majority of individuals), the average payoff significantly drops as it is invaded by 'always defect' strategies. This is due to its genetically weak structure that closely resembles the 'always cooperate' strategy, which is easily invaded by the 'always defect' strategy. The relevant areas on the graph in figure 8.9 that indicate these collapses have been appropriately highlighted.

The population continually tries to get rid of the invading strategy (indicated by the rise of the average payoff), but is unable to do so permanently. Other example simulations never survive such a catastrophic collapse, and the mutual cooperation run returns to a mutual negative state of 'always defect' until the end of training. This results in potential 'always defect' strategies being benchmarked, subsequently resulting in poorer average performance. As it is a characteristic of the type of environment containing 'moving objectives', it is also reflected in the results and previous potentially 'better' strategies were not retained.

Noiseless environment

Continuing with the analysis of the Neural Network approach, table 8.6 lists the performance for evolved networks to play the IPD. As mentioned earlier, the LBest neighbourhood structure is applied, and swarm sizes of 20 and 40 particles are compared along with hidden layer sizes ranging from 5 to 30 nodes.

From the table, it is clear that an increase in swarm size results in an increased ability of the swarm to continue with cooperative behaviour. The configurations making use of 40 particles are able to maintain cooperative runs for almost four times their smaller predecessor's durations. The total number of simulations that were able to elicit cooperative runs varies inconsistently between the two swarm size configurations.

An inspection of hidden layer sizes indicates an increased average personal performance for mid-range sized hidden layers, with a decline in performance experienced for both smaller and larger selections. Overall, the best performer were able to achieve cooperative runs for 262 consecutive generations, with most of the configurations able to elicit cooperative behaviour in more 75% of the simulations.

Table 8.7 depicts the performance for the BinPSO and Symmetry approaches. A more diverse set of neighbourhood structures were used, and swarm sizes of 20, 30 and 40 particles were experimentally examined. The first observation can be made across both techniques. Both the Symmetry and BinPSO approaches are able to elicit cooperative behaviour in all 20 simulations, for each PSO neighbourhood scheme in use.

Table 8.6: Neural network cooperation results for 20 simulations – noiseless environment.

Size	Hidden nodes	Nr of cooperative runs	Longest run
20	5	11	31
20	10	20	76
20	20	18	62
20	30	17	37
40	5	17	257
40	10	15	262
40	20	19	143
40	30	17	135

Table 8.7: Binary PSO and Symmetry Approaches' cooperation results for 20 simulations – noiseless environment.

Structure	Size	Nr of cooperative runs	Longest run
BinPSO GBest	20	20	264
BinPSO Von Neumann	20	20	286
BinPSO LBest	20	20	445
BinPSO GBest	30	20	314
BinPSO Von Neumann	30	20	370
BinPSO LBest	30	20	500
BinPSO GBest	40	20	224
BinPSO Von Neumann	40	20	223
BinPSO LBest	40	20	414
Symmetry GBest	20	20	162
Symmetry Von Neumann	20	20	249
Symmetry LBest	20	20	178
Symmetry GBest	30	20	152
Symmetry Von Neumann	30	20	243
Symmetry LBest	30	20	191
Symmetry GBest	40	20	154
Symmetry Von Neumann	40	20	108
Symmetry LBest	40	20	254

The BinPSO approach favours a mid-range swarm size of 30 particles, with the LBest neighbourhood structure outperforming the other two competitors. The Symmetry approach also favours the LBest neighbourhood structure, but does not yield a consistent trend due to the increase in swarm size. Overall the BinPSO approach is able to elicit longer consecutive runs of cooperation when compared to the Symmetry approach, with the best performing configuration able to cooperate for all 500 epochs of evolution.

The collapsing behaviour for the Neural Network approach was previously discussed. It

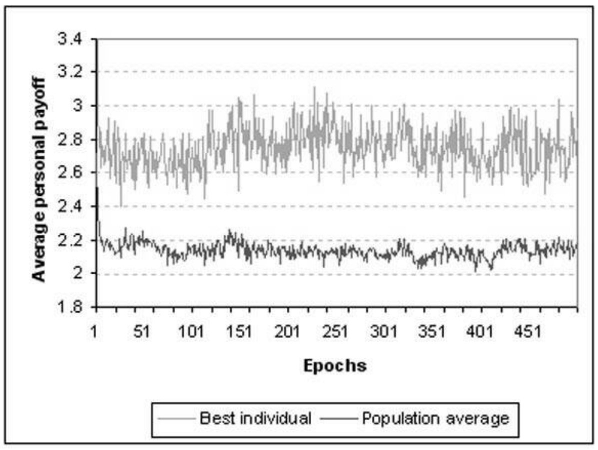


Figure 8.10: Average personal performance of best agent using the BinPSO approach over 500 epochs.

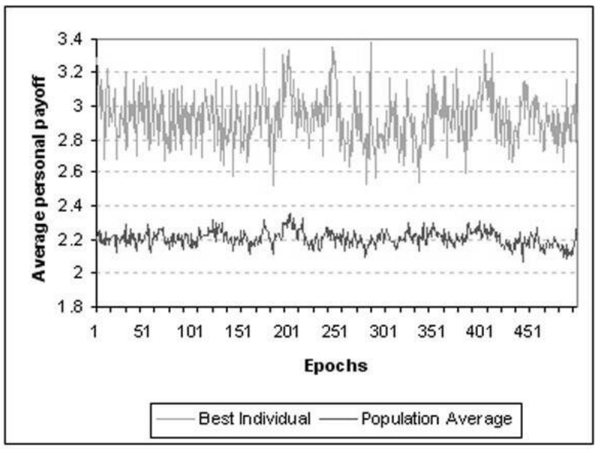


Figure 8.11: Average personal performance of best agent using the Symmetry approach over 500 epochs.

is possible to plot the individual behaviour for the best particle using the remaining two approaches as well. Figure 8.10 and figure 8.11 depict the average personal payoff across 500 epochs for the BinPSO and Symmetry approaches respectively. In comparison with the Neural Network graph in figure 8.9, it is clear that neither the BinPSO or Symmetry approaches suffer from catastrophic collapses, as both the graphs retain a positive average personal payoff without the evidence of mutual defection. Each graph separately depicts the average personal payoff for the best individual and the population as a whole.

The Symmetry approach's graph shows a larger deviation in average performance, but in the process manages to reach higher payoffs with peaks extending close to 3.4. The best individual's graph is roughly centred around a payoff of 3.0, indicating a high probability of mutual cooperation among the members of the population. In comparison, the BinPSO graph shows less deviation in average performance, with the best individual roughly achieving an average payoff of 2.75. Both approaches' populations' average personal payoffs are centred around 2.2, contrasting to the Neural Network population average that closely matched the best individual's behaviour. This may be the key to the success of the BinPSO and Symmetry approaches' performance, since it shows that the populations are less susceptible to high fluctuations in best performing behaviour. The best performing agents' strategies may contain a large number of defecting exchanges, but the exploitation by the individual over the group is accepted since the population as whole is in an equilibrium of mutually beneficial behaviour. The defecting strategies do not dominate the cooperating strategies in the Symmetry and BinPSO approaches. The same reasoning does not hold for the Neural Network approach, which is extremely vulnerable to exploitation.

Noisy environment

The application of noise to the IPD was previously discussed in section 8.6.2. Analysis of population behaviour shows a very interesting change in the various approaches' abilities to continue eliciting cooperative behaviour. Starting with the Neural Network approach, the results for an increase in swarm size and hidden nodes is depicted in table 8.8. It is clear that the Neural Network approach is completely unable to cope with even a slight noise margin of only 5%. An increase in swarm size results in even poorer performance – in contrast with the noiseless environment's experimental results. Small swarm sizes are able to elicit cooperation in only less than 25% of the simulations, with the longest uninterrupted run only consisting of 126 generations. Larger swarm sizes fail miserably, with only a single cooperative simulation

Table 8.8: Neural Network approach's cooperation results for 20 simulations – 5% noise applied.

Size	Hidden nodes	Nr of cooperative runs	Longest run
20	5	4	32
20	10	2	34
20	20	5	29
20	30	3	126
40	5	1	1
40	10	1	1
40	20	1	3
40	30	0	0

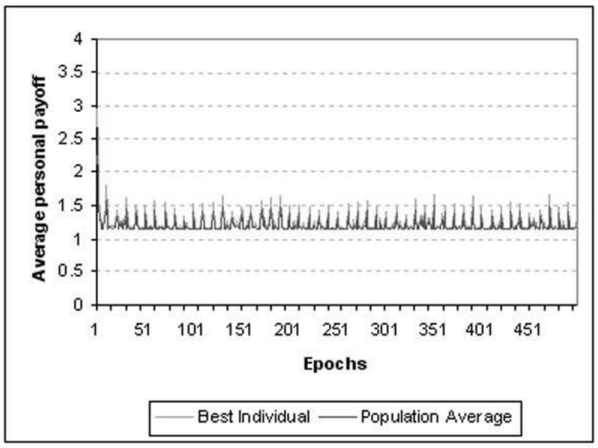


Figure 8.12: Average personal performance of best agent using the Neural Network approach over 500 epochs in a noisy environment.

and generation per configuration.

Plotting the average personal performance graph in figure 8.12, reveals that the Neural Network approach stagnates on an 'always defect' strategy. The consistent jumps in average personal performance is due to the use of the particle dispersement operator (as described in section 6.4.3) that is able to briefly influence the convergence of the swarm on the inferior local solution. Unfortunately, the swarm is not able to permanently recover from its poor performance, and the 'always defect' strategy prevails, along with the associated poor payoff. It is interesting to note that an increase of the noise level beyond 50% results in a sudden shift of strategy from a complete 'allways defect' position to an 'allways cooperate' position. The logical reasoning behind this is that such a high error rate would convert the 'cooperate' signal into a 'defect' signal, which is what the neural networks were aiming for all along.

The impact of the application of noise to the communication channel for the Symmetry and BinPSO approaches differ quite substantially from the Neural Network approach. Repeating the experiments conducted in the previous section, each approach was tested with swarm sizes of 20, 30 and 40 particles respectively. All three PSO neighbourhood information sharing structures were also applied for each approach in turn. In almost every single experiment, both the Symmetry and BinPSO approaches were able to elicit cooperative runs spanning the maximum of 500 epochs, for all 20 simulations. Table 8.9 lists the few exceptions to this case, but still illustrates a very good performance as compared to the Neural Network approach.

Plotting the average personal performance graphs for the BinPSO and Symmetry approaches also indicate slightly different playing behaviour when compared to the noiseless equivalents. The BinPSO approach performance in an environment with 5% noise is illus-

Table 8.9: Exceptions to superb performing Binary PSO and Symmetry Approaches' cooperation results for 20 simulations – 5% noise applied.

Structure	Size	Nr of cooperative runs	Longest run
Symmetry GBest	20	20	497
Symmetry LBest	20	20	473
Symmetry GBest	30	20	324

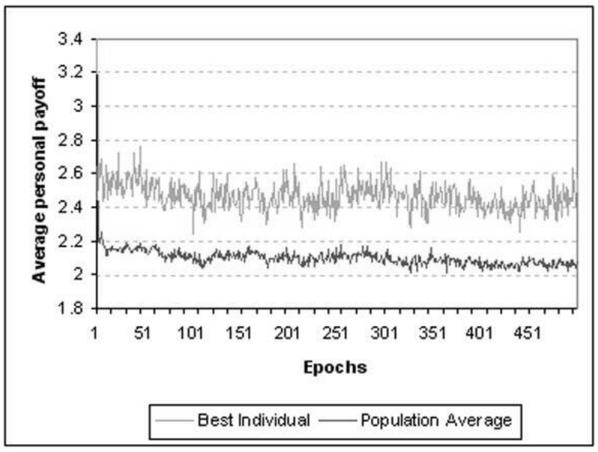


Figure 8.13: Average personal performance of best agent using the BinPSO approach over 500 epochs in a noisy environment.

trated in figure 8.13. As already alluded to in the aforementioned performance tables, the BinPSO approach is able to elicit cooperative behaviour from the other particles in the swarm. This fact is clearly indicated by the best individual's average personal payoff consistently centered around 2.5, and the average population performance plotted in the region of 2.1. When compared to the equivalent graph for a noiseless environment (figure 8.10), it seems as if the population performance is unaffected by the noise. The best individual's personal performance dropped slightly, but the deviation across 500 epochs has been minimised.

Figure 8.14 represents the performance plot for the Symmetry approach in a noisy environment. When compared to the noiseless equivalent in figure 8.11, it is clear that the deviation has been drastically reduced for both the best individual and the population average. It is also interesting to note that the Symmetry approach is not able to maintain a consistent average payoff, but instead shows definite signs of declining performance over 500 epochs. Even though the performance for the Symmetry approach is relatively good, it does not outperform the BinPSO approach.

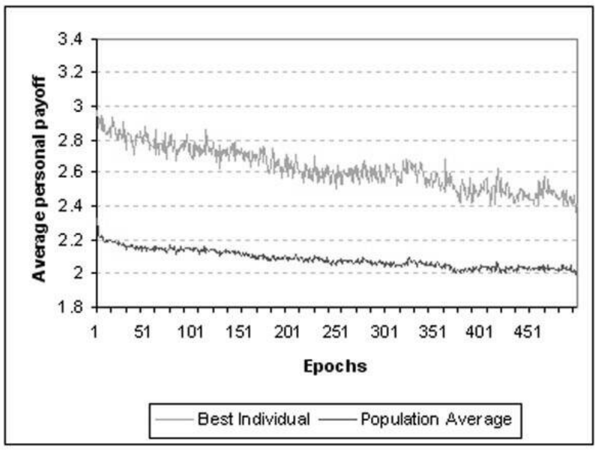


Figure 8.14: Average personal performance of best agent using the Symmetry approach over 500 epochs in a noisy environment.

8.7.3 Strategy performance

Noiseless environment

Table 8.10 depicts the performance of the various evolutionary approaches to IPD strategy generation, as computed against the benchmark suite of man-made strategies in a noiseless environment. The two best-performing Neural Network configurations from the experimental work on Checkers are now again applied to the IPD problem domain. The first constitutes a swarm size of 20 particles and 20 hidden nodes, and the second 5 hidden nodes and 40 particles. In both cases the LBest neighbourhood structure was applied, using an inclusive neighbourhood size of 5 particles.

In order to ease the performance comparison between the BinPSO, Symmetry and Neural Network approaches, the former two methods also only made use of swarms sizes of 20 and 40 particles. The analysis of the various neighbourhood structures were however extended in both the Symmetry and BinPSO approaches, and the standard GBest, LBest and Von Neumann architectures were examined.

The average total payoff over 151 interactions are listed beneath each man-made strategy's column, while the edge of the table represents the average personal payoff (APP) and average total payoff (ATP) per move for each strategy generation approach.

Firstly considering the Neural Network approach, the configuration of 5 hidden nodes and 40 particles outperformed 20 hidden nodes and 20 particles on both the APP and ATP. Even though the Neural Network approach only managed to score the 3^{rd} and 4^{th} lowest APP scores, it achieved the overall highest ATP scores due to its good level of cooperation with

Table 8.10: Best performing individuals, averaged over 20 runs and 10000 games per run – no noise.

Approach	Size	Arch	RAND	PVLV	ALLC	TFT	STFT	TFTT	Avg PP	Avg TP
Neural Network (20HN)	20	LB	355.37	355.92	597.52	313.50	313.47	361.07	2.5352	4.4861
Benchmark vs NN (20HN)	20	LB	303.39	296.69	236.22	313.44	313.41	304.38	1.9509	4.4861
Neural Network (5HN)	40	LB	362.43	366.82	587.32	324.88	324.88	401.38	2.6134	4.6009
Benchmark vs NN (5HN)	40	LB	286.91	313.28	251.53	324.81	324.82	299.35	1.9875	4.6009
BinPSO	20	LB	394.76	403.24	691.06	317.32	317.29	432.53	2.8214	4.3596
Benchmark vs BinPSO	20	LB	211.51	185.83	95.91	316.97	316.93	266.45	1.5382	4.3596
BinPSO	40	LB	416.30	421.45	684.79	267.94	267.79	407.98	2.7221	4.0315
Benchmark vs BinPSO	40	LB	161.19	141.17	105.31	266.92	266.77	244.88	1.3093	4.0315
BinPSO	20	GB	405.82	397.52	687.85	282.62	282.93	399.89	2.7115	4.1249
Benchmark vs BinPSO	20	GB	185.47	203.03	100.73	281.88	282.20	227.17	1.4133	4.1249
BinPSO	40	GB	407.52	393.59	670.72	301.58	301.46	423.63	2.7577	4.2696
Benchmark vs BinPSO	40	GB	181.51	208.85	126.43	301.13	301.01	250.81	1.5118	4.2696
BinPSO	20	VN	402.51	405.77	657.10	310.81	310.75	400.52	2.7455	4.3243
Benchmark vs BinPSO	20	VN	193.34	184.39	146.85	310.46	310.42	284.90	1.5787	4.3243
BinPSO	40	VN	419.82	429.00	688.11	283.77	283.67	385.67	2.7484	4.0900
Benchmark vs BinPSO	40	VN	153.06	133.56	100.34	283.07	282.96	262.54	1.3417	4.0900
Symmetry	20	LB	433.12	439.58	731.34	207.14	207.24	254.07	2.5083	3.4446
Benchmark vs Symmetry	20	LB	121.76	101.98	35.48	205.11	205.21	178.81	0.9364	3.4446
Symmetry	40	LB	441.80	449.35	741.97	199.86	199.96	251.07	2.5210	3.3787
Benchmark vs Symmetry	40	LB	101.84	83.96	19.54	197.68	197.79	176.27	0.8577	3.3787
Symmetry	20	GB	422.18	425.69	723.43	242.24	242.23	333.58	2.6373	3.7671
Benchmark vs Symmetry	20	GB	147.41	129.91	47.36	240.66	240.65	217.63	1.1298	3.7671
Symmetry	40	GB	421.78	423.84	731.64	257.88	257.85	339.86	2.6853	3.8449
Benchmark vs Symmetry	40	GB	148.35	145.35	35.04	256.43	256.38	209.06	1.1596	3.8449
Symmetry	20	VN	417.86	421.08	718.39	258.75	258.75	315.51	2.6383	3.8438
Benchmark vs Symmetry	20	VN	157.65	150.64	54.92	257.57	257.56	213.81	1.2055	3.8438
Symmetry	40	VN	421.60	435.48	746.39	264.22	264.23	349.83	2.7392	3.8550
Benchmark vs Symmetry	40	VN	148.87	118.51	12.92	262.92	262.94	204.69	1.1157	3.8550

its opponents. As already mentioned, the openly cooperative nature of the Neural Network approach also leads to its downfall as evident by the occurrence of catastrophic collapses. The Neural Network approach managed to outperform all the benchmark strategies, with the largest margin of more than 360 points against ALLC, and the smallest margin of only 0.06 against TFT and STFT.

An examination of the various neighbourhood information sharing structures as applied to the BinPSO approach indicate that the Von Neumann structure outperformed both the standard GBest and LBest structures. The BinPSO approach was able to outperform most of the Symmetry and Neural Network APP scores, but still trailed the Neural Network approach on ATP. The various BinPSO approaches managed to achieve the top four APP scores. BinPSO outperformed all the benchmark strategies with the largest margin of 596 against ALLC and the smallest margin of 0.33 against STFT. It managed to outperform the Neural Network approach's APP, but could not duplicate the deep level of cooperation achieved by the Neural Network approach. The benchmark strategies struggled to cooperate with BinPSO as evident by the low ATP scores.

Finally considering ATP scores of the various Symmetry approaches reveal that the Von Neumann structure yet again outperformed the standard GBest and LBest structures. The Symmetry approach unfortunately recorded the overall worst ATP scores, but managed to mostly surpass the Neural Network approach on APP. It is interesting to note that the Symmetry approach almost completely deprives the benchmark strategies of recording any APP scores. The Symmetry approach also outperformed all the man-made benchmark strategies, which struggled to cooperate with it when considering the ATP scores.

Table 8.11 provides a detailed comparison of the individual performance of the various man-made strategies against the evolved approaches. The APP scores reflect those of the benchmark strategies, and the average opponent payoff (AOP) scores correspond to the different evolved approaches.

The RAND strategy struggled most against the Symmetry approach, only scoring 0.674 against a LBest configuration of 40 particles. Not considering its performance against the other benchmark strategies, RAND managed to cooperate the best with a Neural Network approach of 20 hidden nodes – scoring an ATP value of 4.362. RAND was unable to exceed the individual payoff scores of any of its opponents.

PVLV managed to score the 3^{rd} highest ATP score against the man-made benchmark strategies, but struggled most to cooperate with the Symmetry approach – only managing 0.556 against a configuration of 40 particles utilising the standard LBest neighbourhood structure. PVLV did manage to score a respectable ATP of 4.502 against a Neural Network opponent with 5 hidden nodes, but could not exceed any of the evolved strategies' individual payoffs.

The ALLC man-made strategy managed to score the highest ATP of all the investigated strategies. It cooperated very well with the other benchmark strategies, recording an APP close to 3.0. It was able to partially cooperate with the Neural Network approach, scoring slightly less than 3.0 due to the occurrence of catastrophic collapses. The ALLC strategy was totally exploited by the BinPSO and Symmetry approaches, only managing a low score of 0.086 against the Symmetry approach with a swarm size of 40 particles and utilising the Von Neumann structure.

The TFT and STFT strategies had a much more balanced performance, due to the reciprocating nature of these strategies. As most of the other man-made strategies, the TFT and STFT strategies performed best against the benchmark strategies with an ATP above 4.91. Even though neither strategy was able to outperform the evolved strategies' AOP, both TFT and STFT were able to equal the AOP or come within 0.02 in numerous occasions. TFT and STFT struggled against the Symmetry approach, but were able to cooperate with the Neural

Network and BinPSO approaches. The overall best ATP against an evolved strategy was 4.303, achieved in competition with the Neural Network approach utilising 5 hidden nodes.

TFTT performed slightly worse than the TFT and STFT strategies when considering the APP scores, but was able to elicit deeper cooperation with all the opponent strategies – indicated by the superior ATP scores.

Noisy environment

Table 8.12 depicts the performance of the various evolutionary approaches to IPD strategy generation, as computed against the benchmark suite of man-made strategies in an environment with 5% noise applied to the communication channel. The parameter settings that applied to the experiments in the noiseless environment are once again used in this set of experiments.

The average total payoff over 151 interactions are listed beneath each man-made strategy's column, while the edge of the table represents the average personal payoff (APP) and average total payoff (ATP) per move for each strategy generation approach.

Starting with the Neural Network approach, the experimental configuration of 20 hidden nodes and swarm size of 20 particles outperformed the configuration of 5 hidden nodes and a swarm size of 40 particles, on both APP and ATP scores. The 'superior' performance is negligible though, with marginal differences of 0.0001 and 0.0002 in both cases. When compared to the noiseless equivalent the Neural Network's APP declined slightly. The benchmark opponents were completely unable to cooperate with either Neural Network, as indicated by very low AOP scores (0.799). The poor cooperation can be attributed to the 'always defect' strategy that emerges due to the presence of noise, as depicted in figure 8.12. The Neural Network approach achieved the overall worst ATP scores of any of the evolved strategies, only managing 3.25, but still outperformed all the man-made benchmark strategies.

Analysis of the BinPSO approach's results once again shows that the Von Neumann structure is able to outperform both the standard GBest and LBest structures. The BinPSO approach is able to mostly outperform the Symmetry approach on APP scores, but completely outperforms the Neural Network approach's APP score. Overall, the BinPSO approach has the best ATP score, almost achieving scores reported in the noiseless environment. This indicates the strategy's good performance in the noisy environment. None of the benchmark strategies could compete with the BinPSO approach, with a large number of the BinPSO scores improving on the equivalent noiseless environment scores. The improvement in scores can be attributed to the benchmark strategies' inability to cope with noise as well as the BinPSO approach, resulting in a shift in performance from the AOP to APP scores.

Lastly, the results for the Symmetry approach indicate that both the Von Neumann and standard GBest structures are able to improve over the APP and ATP scores reported for the noiseless environment – also outperforming the Neural Network APP scores. The Von Neumann structure results in the largest APP and ATP scores for the Symmetry approach, clearly illustrating the structure's dominance yet again. All of the benchmark strategies' APP scores were restricted close to 1.0, a limit somewhat higher than the reported noiseless environment equivalent. Overall the ATP scores serve as an indication of the poor cooperative state between the benchmark strategies and the Symmetry approach. The Symmetry approach managed to outperform all the benchmark strategies, with the largest margin of 734.53 against ALLC, and the smallest margin of 26.49 against STFT.

Table 8.13 provides a detailed comparison of the individual performance of the various man-made strategies against the evolved approaches in the noisy environment. The APP scores reflect those of the benchmark strategies, and the average opponent payoff (AOP) scores correspond to the different evolved approaches.

The first major observation involves noticing the overall drop in APP scores for most of the man-made benchmark strategies, when compared to the reported noiseless environment scores. Once again none of the man-made strategies could manage to outperform any of the evolved strategies. TFTT manages to outperform both the TFT and STFT strategies' ATP scores in the noisy environment, supporting the aforementioned claim that TFTT is more suited to elicit cooperative behaviour in noisy environments. PVLV performs equally well under noisy conditions, with ATP scores comparatively similar to TFTT. The most interesting observation however is the performance of the RAND strategy, with ATP scores comparing well with TFTT and PVLV.

Overall, the man-made benchmark strategies continued to struggle the most against the Symmetry approach, and remained unable to bring about any form of sustained cooperative behaviour that benefited both the APP and AOP scores.

8.8 Conclusion

This chapter applied the PSO-based coevolutionary training technique to the non-zero sum game of the Iterated Prisoner's Dilemma. It was the first application of particle swarm optimisation in the history of the IPD. Three novel strategy generation methods were presented. The first continued the original zero-sum approach of evolving neural networks as evaluators, given three historic personal and opposition states. The second made use of the lesser known Binary PSO algorithm to directly evolve a 64-bit strategy. The last approach exploited the symmetrical nature of man-made strategies, and explored the PSO's function optimisation abilities to evolve the first 32 bits of a 64-bit strategy. Two performance measures were established, with the first analysing the general ability of a population to bring about mutually cooperative behaviour. The second investigated the specific strategy performance when benchmarked against a selection of six well-known man-made IPD strategies.

The experimental results indicated that the Neural Network approach was able to elicit deep levels of cooperation, which could lead to so-called catastrophic collapses due to the development of genetic weaknesses. The Neural Network approach was unable to elicit any cooperative behaviour in an environment with 5% noise applied to the communication channel.

The results for the BinPSO approach seem to suggest a state of equilibrium in which cooperative behaviour takes place, but exploitation also exists. The defecting strategies were neither 'allowed' to flourish, nor completely die out. This increased the robustness of the population of strategies, and the BinPSO approach was able to reach a state of equilibrium in a noisy environment as well.

Finally, the Symmetry approach showed similar performance traits as the BinPSO approach in a noiseless environment, but was unable to maintain a consistent payoff in a noisy environment. The Symmetry approach managed to deprive the benchmark strategies of attaining any payoffs beyond 1.0 – a characteristic not suited for mutual cooperation against a diverse collection of opposing strategies.

Table 8.11: Performance of individual man-made strategies - no noise.

RAND					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.688	2.063	4.750
NN (20HN)	20	LB	2.009	2.353	4.362
NN (5HN)	40	LB	1.900	2.400	4.300
BinPSO	20	LB	1.401	2.614	4.015
BinPSO	40	LB	1.067	2.757	3.824
BinPSO	20	GB	1.228	2.688	3.916
BinPSO	40	GB	1.202	2.699	3.901
BinPSO	20	VN	1.280	2.666	3.946
BinPSO	40	VN	1.014	2.780	3.794
Symmetry	20	LB	0.806	2.868	3.674
Symmetry	40	LB	0.674	2.926	3.600
Symmetry	20	GB	0.976	2.796	3.772
Symmetry	40	GB	0.982	2.793	3.775
Symmetry	20	VN	1.044	2.767	3.811
Symmetry	40	VN	0.986	2.792	3.778

PVLV					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.764	2.216	4.980
NN (20HN)	20	LB	1.965	2.357	4.322
NN (5HN)	40	LB	2.075	2.429	4.504
BinPSO	20	LB	1.231	2.670	3.901
BinPSO	40	LB	0.935	2.791	3.726
BinPSO	20	GB	1.345	2.633	3.978
BinPSO	40	GB	1.383	2.607	3.990
BinPSO	20	VN	1.221	2.687	3.908
BinPSO	40	VN	0.885	2.841	3.726
Symmetry	20	LB	0.675	2.911	3.586
Symmetry	40	LB	0.556	2.976	3.532
Symmetry	20	GB	0.860	2.819	3.679
Symmetry	40	GB	0.963	2.807	3.770
Symmetry	20	VN	0.998	2.789	3.787
Symmetry	40	VN	0.785	2.884	3.669

ALLC					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.496	3.336	5.832
NN (20HN)	20	LB	1.564	3.957	5.521
NN (5HN)	40	LB	1.666	3.890	5.556
BinPSO	20	LB	0.635	4.577	5.212
BinPSO	40	LB	0.697	4.535	5.232
BinPSO	20	GB	0.667	4.555	5.222
BinPSO	40	GB	0.837	4.442	5.279
BinPSO	20	VN	0.973	4.352	5.325
BinPSO	40	VN	0.665	4.557	5.222
Symmetry	20	LB	0.235	4.843	5.078
Symmetry	40	LB	0.129	4.914	5.043
Symmetry	20	GB	0.314	4.791	5.105
Symmetry	40	GB	0.232	4.845	5.077
Symmetry	20	VN	0.364	4.758	5.122
Symmetry	40	VN	0.086	4.943	5.029

TFT					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.460	2.455	4.914
NN (20HN)	20	LB	2.076	2.076	4.152
NN (5HN)	40	LB	2.151	2.152	4.303
BinPSO	20	LB	2.099	2.101	4.200
BinPSO	40	LB	1.768	1.774	3.542
BinPSO	20	GB	1.867	1.872	3.739
BinPSO	40	GB	1.994	1.997	3.991
BinPSO	20	VN	2.056	2.058	4.114
BinPSO	40	VN	1.875	1.879	3.754
Symmetry	20	LB	1.358	1.372	2.730
Symmetry	40	LB	1.309	1.324	2.633
Symmetry	20	GB	1.594	1.604	3.198
Symmetry	40	GB	1.698	1.708	3.406
Symmetry	20	VN	1.706	1.714	3.420
Symmetry	40	VN	1.741	1.750	3.491

STFT					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.460	2.455	4.915
NN (20HN)	20	LB	2.076	2.076	4.152
NN (5HN)	40	LB	2.151	2.152	4.303
BinPSO	20	LB	2.099	2.101	4.200
BinPSO	40	LB	1.767	1.773	3.540
BinPSO	20	GB	1.869	1.874	3.743
BinPSO	40	GB	1.993	1.996	3.989
BinPSO	20	VN	2.056	2.058	4.114
BinPSO	40	VN	1.874	1.879	3.753
Symmetry	20	LB	1.359	1.372	2.731
Symmetry	40	LB	1.310	1.324	2.634
Symmetry	20	GB	1.594	1.604	3.198
Symmetry	40	GB	1.698	1.708	3.406
Symmetry	20	VN	1.706	1.714	3.420
Symmetry	40	VN	1.741	1.750	3.491

TFTT					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.548	2.891	5.439
NN (20HN)	20	LB	2.016	2.391	4.407
NN (5HN)	40	LB	1.982	2.658	4.640
BinPSO	20	LB	1.765	2.864	4.629
BinPSO	40	LB	1.622	2.702	4.324
BinPSO	20	GB	1.504	2.648	4.152
BinPSO	40	GB	1.661	2.805	4.466
BinPSO	20	VN	1.887	2.652	4.539
BinPSO	40	VN	1.739	2.554	4.293
Symmetry	20	LB	1.184	1.683	2.867
Symmetry	40	LB	1.167	1.663	2.830
Symmetry	20	GB	1.441	2.209	3.650
Symmetry	40	GB	1.385	2.251	3.636
Symmetry	20	VN	1.416	2.089	3.505
Symmetry	40	VN	1.356	2.317	3.673

Table 8.12: Best performing individuals, averaged over 20 runs and 10000 games per run – 5% noise applied.

Approach	**Size**	**Arch**	**RAND**	**PVLV**	**ALLC**	**TFT**	**STFT**	**TFTT**	**Avg PP**	**Avg TP**
Neural Network (20HN)	20	LB	441.71	441.66	710.02	201.72	201.69	229.49	2.4573	3.2569
Benchmark vs NN (20HN)	20	LB	101.93	101.99	30.67	165.75	165.71	158.38	0.7996	3.2569
Neural Network (5HN)	40	LB	441.68	441.66	710.04	201.71	201.74	229.44	2.4572	3.2567
Benchmark vs NN (5HN)	40	LB	101.96	101.96	30.63	165.70	165.72	158.34	0.7995	3.2567
BinPSO	20	LB	411.84	397.18	649.56	268.27	268.19	331.94	2.5684	3.9268
Benchmark vs BinPSO	20	LB	171.56	206.87	125.24	250.40	250.25	226.39	1.3584	3.9268
BinPSO	40	LB	408.65	393.27	654.71	281.78	281.93	356.84	2.6238	4.0165
Benchmark vs BinPSO	40	LB	179.13	210.52	117.13	266.16	266.39	222.46	1.3927	4.0165
BinPSO	20	GB	405.28	402.49	655.04	275.90	275.82	367.63	2.6293	3.9982
Benchmark vs BinPSO	20	GB	186.99	188.00	116.63	258.90	258.90	230.80	1.3689	3.9982
BinPSO	40	GB	415.02	407.32	644.18	264.66	264.76	327.12	2.5641	3.8771
Benchmark vs BinPSO	40	GB	164.34	183.97	133.63	245.26	245.29	217.09	1.3130	3.8771
BinPSO	20	VN	405.49	406.85	649.54	269.93	270.03	357.25	2.6038	3.9609
Benchmark vs BinPSO	20	VN	186.38	187.27	125.33	250.48	250.52	229.48	1.3570	3.9609
BinPSO	40	VN	415.54	401.02	656.24	259.35	259.32	335.26	2.5681	3.8525
Benchmark vs BinPSO	40	VN	163.00	193.36	114.74	237.65	237.71	217.16	1.2843	3.8525
Symmetry	20	LB	428.79	428.75	700.70	237.23	237.16	295.74	2.5700	3.5737
Benchmark vs Symmetry	20	LB	131.91	131.18	45.29	209.19	209.08	182.70	1.0037	3.5737
Symmetry	40	LB	431.14	438.42	699.68	233.34	233.31	290.95	2.5682	3.5353
Benchmark vs Symmetry	40	LB	126.42	108.55	46.87	204.85	204.82	184.65	0.9671	3.5353
Symmetry	20	GB	421.41	424.82	688.65	243.27	243.36	331.76	2.5974	3.6863
Benchmark vs Symmetry	20	GB	149.23	141.80	64.10	216.78	216.79	197.80	1.0888	3.6863
Symmetry	40	GB	428.51	428.79	697.17	242.75	242.75	305.24	2.5885	3.6161
Benchmark vs Symmetry	40	GB	132.71	128.26	50.76	215.80	215.82	187.59	1.0275	3.6161
Symmetry	20	VN	423.22	425.34	692.30	235.40	235.40	302.75	2.5545	3.5985
Benchmark vs Symmetry	20	VN	145.02	137.41	58.40	207.22	207.17	190.59	1.0439	3.5985
Symmetry	40	VN	436.18	433.17	705.04	216.21	216.24	245.61	2.4861	3.3765
Benchmark vs Symmetry	40	VN	114.80	121.19	38.53	183.48	183.56	165.10	0.8903	3.3765

Table 8.13: Performance of individual man-made strategies - 5% noise.

RAND					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.644	2.081	4.725
NN (20HN)	20	LB	0.675	2.925	3.600
NN (5HN)	40	LB	0.675	2.925	3.600
BinPSO	20	LB	1.136	2.727	3.863
BinPSO	40	LB	1.186	2.706	3.892
BinPSO	20	GB	1.238	2.684	3.922
BinPSO	40	GB	1.088	2.748	3.836
BinPSO	20	VN	1.234	2.685	3.919
BinPSO	40	VN	1.079	2.752	3.831
Symmetry	20	LB	0.874	2.840	3.714
Symmetry	40	LB	0.837	2.855	3.692
Symmetry	20	GB	0.988	2.791	3.779
Symmetry	40	GB	0.879	2.838	3.717
Symmetry	20	VN	0.960	2.803	3.763
Symmetry	40	VN	0.760	2.889	3.649

PVLV					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.829	2.100	4.929
NN (20HN)	20	LB	0.675	2.925	3.600
NN (5HN)	40	LB	0.675	2.925	3.600
BinPSO	20	LB	1.370	2.630	4.000
BinPSO	40	LB	1.394	2.604	3.998
BinPSO	20	GB	1.245	2.665	3.910
BinPSO	40	GB	1.218	2.697	3.915
BinPSO	20	VN	1.240	2.694	3.934
BinPSO	40	VN	1.281	2.656	3.937
Symmetry	20	LB	0.869	2.839	3.708
Symmetry	40	LB	0.719	2.903	3.622
Symmetry	20	GB	0.939	2.813	3.752
Symmetry	40	GB	0.849	2.840	3.689
Symmetry	20	VN	0.910	2.817	3.727
Symmetry	40	VN	0.803	2.869	3.672

ALLC					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.440	3.272	5.712
NN (20HN)	20	LB	0.203	4.702	4.905
NN (5HN)	40	LB	0.203	4.702	4.905
BinPSO	20	LB	0.829	4.302	5.131
BinPSO	40	LB	0.776	4.336	5.112
BinPSO	20	GB	0.772	4.338	5.110
BinPSO	40	GB	0.885	4.266	5.151
BinPSO	20	VN	0.830	4.302	5.132
BinPSO	40	VN	0.760	4.346	5.106
Symmetry	20	LB	0.300	4.640	4.940
Symmetry	40	LB	0.310	4.634	4.944
Symmetry	20	GB	0.425	4.561	4.986
Symmetry	40	GB	0.336	4.617	4.953
Symmetry	20	VN	0.387	4.585	4.972
Symmetry	40	VN	0.255	4.669	4.924

TFT					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.539	2.462	5.000
NN (20HN)	20	LB	1.098	1.336	2.434
NN (5HN)	40	LB	1.097	1.336	2.433
BinPSO	20	LB	1.658	1.777	3.435
BinPSO	40	LB	1.763	1.866	3.629
BinPSO	20	GB	1.715	1.827	3.542
BinPSO	40	GB	1.624	1.753	3.377
BinPSO	20	VN	1.659	1.788	3.447
BinPSO	40	VN	1.574	1.718	3.292
Symmetry	20	LB	1.385	1.571	2.956
Symmetry	40	LB	1.357	1.545	2.902
Symmetry	20	GB	1.436	1.611	3.047
Symmetry	40	GB	1.429	1.608	3.037
Symmetry	20	VN	1.372	1.559	2.931
Symmetry	40	VN	1.215	1.432	2.647

STFT					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.539	2.462	5.001
NN (20HN)	20	LB	1.097	1.336	2.433
NN (5HN)	40	LB	1.097	1.336	2.433
BinPSO	20	LB	1.657	1.776	3.433
BinPSO	40	LB	1.764	1.867	3.631
BinPSO	20	GB	1.715	1.827	3.542
BinPSO	40	GB	1.624	1.753	3.377
BinPSO	20	VN	1.659	1.788	3.447
BinPSO	40	VN	1.574	1.717	3.291
Symmetry	20	LB	1.385	1.571	2.956
Symmetry	40	LB	1.356	1.545	2.901
Symmetry	20	GB	1.436	1.612	3.048
Symmetry	40	GB	1.429	1.608	3.037
Symmetry	20	VN	1.372	1.559	2.931
Symmetry	40	VN	1.216	1.432	2.648

TFTT					
Opponent	*Size*	*Arch*	*Avg PP*	*Avg OP*	*Avg TP*
Bench Suite	N/A	N/A	2.493	3.106	5.599
NN (20HN)	20	LB	1.049	1.520	2.569
NN (5HN)	40	LB	1.049	1.519	2.568
BinPSO	20	LB	1.499	2.198	3.697
BinPSO	40	LB	1.473	2.363	3.836
BinPSO	20	GB	1.528	2.435	3.963
BinPSO	40	GB	1.438	2.166	3.604
BinPSO	20	VN	1.520	2.366	3.886
BinPSO	40	VN	1.438	2.220	3.658
Symmetry	20	LB	1.210	1.959	3.169
Symmetry	40	LB	1.223	1.927	3.150
Symmetry	20	GB	1.310	2.197	3.507
Symmetry	40	GB	1.242	2.021	3.263
Symmetry	20	VN	1.262	2.005	3.267
Symmetry	40	VN	1.093	1.627	2.720

CHAPTER 9

Conclusion and future work

"History will be kind to me for I intend to write it".
- Sir Winston Churchill (1874 - 1965)

This chapter summarises the major findings of the examined work, and provides a set of suggested future work that can be investigated as a result.

9.1 Conclusion

This study aimed to investigate the use of particle swarm optimisation to train neural networks as game state evaluators in a coevolutionary environment. Background investigation into existing game learning techniques and computational intelligence paradigms led to the introduction of a training algorithm for this specific purpose.

The algorithm was initially applied to the computational modest problem of Tic-Tac-Toe. An analysis of PSO neighbourhood structures indicated that the Von Neumann structure was superior in evolving intelligent game playing agents when no maximum velocity was present. It was also evident that an increase in performance was attainable by adjusting a selection of the core PSO parameters.

The problem space was increased significantly with the application of the algorithm to the game of Checkers. Once again, the Von Neumann architecture showed its supremacy as neighbourhood structure when no maximum velocity was present. Benchmarking the evolved players against a random-moving player showed that some degree of learning took place during training, but the overall performance was disconcerting.

Due to the poor playing performance in Checkers, a more detailed analysis of the core PSO parameters, neural network input representation and coevolutionary training techniques was conducted. Results show a definite increase in performance with selected PSO parameters, with the most influential change attributed to the use of a small maximum velocity. The LBest neighbourhood structure exceeded the Von Neumann structure's performance, and the application of more recent structure enhancements such as the GCPSO algorithm proved even more promising. The use of windowing techniques to represent more spatial information about the board state to the neural network resulted in a slight increase in performance, albeit with a computational complexity penalty.

Two new coevolutionary techniques based on Formula One Grand Prix racing were introduced. The first involved a racing scheme based on a scoring system inspired by the FIA regulations [48]. The racing scheme aimed to benefit consistency and eliminate any 'one-shot wonders' among the playing population. A particle dispersement operator, based on particle repelling techniques, but also making use of the GPX scoring scheme, was additionally introduced. Depending on its configuration, the dispersement operator allowed the swarm to avoid convergence on suboptimal solutions in the majority of cases. Both new coevolutionary techniques resulted in improved performance with small swarm sizes and neural network configurations.

The superior evolved players were benchmarked against players relying on human intelligence – a piece-count based evaluation function and a slimmed-down commercial strength evaluation function (SmartEval). An increase in training and playing tree depth were also investigated. The evolved players were able to outperform the piece-count based player and evenly match the SmartEval player. Tests were conducted using the smallest configuration size of 15 particles and 3 hidden nodes. An increase in tree depth resulted in a severe drop in performance, and a set of experimental questions were posited that could address the drawbacks associated with the PSO approach and large tree depths. It is expected that an increase in ply-depth will require a much larger experimental PSO and neural network configuration to be successful.

Finally, the PSO-based coevolutionary training algorithm's versatility was tested by applying it to the non-zero sum problem of the Iterated Prisoner's Dilemma. The PSO was applied in three distinct ways to generate IPD strategies – the first of its kind in the IPD problem domain. The first continued the existing approach of evolving neural network weights, which in this case implied making predictions on future cooperate/defect decisions given a history bound of 3 moves. The second approach utilised the Binary PSO algorithm to directly evolve a 64-bit strategy. The last approach relied on the PSO's function optimisation ability to generate the first half of a 64-bit strategy, and exploited the symmetrical structure of man-made strategies. The second half of the strategy was constructed by mirroring the evolved half using a selection of four different methods. Results indicated that all three approaches were able to elicit mutually cooperative behaviour in noiseless environments. The neural network approach showed an inherent genetic weakness, resulting in so-called 'catastrophic collapses'. The BinPSO and Symmetry approaches did not suffer from the same weakness, but struggled to consistently maintain high payoffs, with performance graphs showing highly turbulent best-performing particles. Testing under environments with 5% noise indicated a complete failure by the neural network approach, a constant decline in performance for the Symmetry approach, and only a lowered performance rating for the BinPSO approach. The BinPSO approach maintained an equilibrium within the population, and defecting strategies were not allowed to flourish, nor completely die out.

Overall the PSO-based coevolutionary training approach seems to be a successful game learning technique. A large set of possible extensions and future work can be investigated based on the results of the selected experiments performed in this study.

9.2 Future work

Evolving specialised strategies through niching

Multi-modal function optimisation involves finding a set of optimal solutions, rather than a specific global best solution. Solving multi-modal problems through evolutionary computation has traditionally been done through a technique known as speciation (or niching). Recently, the first niching techniques were developed for PSO algorithms by Brits [20].

A good game player does not only rely on a single strategy to continuously beat its opponents. Instead, the player is able to adapt to the different playing styles of the various opponents. Each strategy can be seen as an optimum in a multi-modal search space. Applying niching to a game learning problem then involves the discovery and use of the various optimal strategies.

Initial work on applying niching to game learning have been done by Kim *et al.* [79] in 2002. The work was restricted to a genetic algorithm implementation, making use of a crowding algorithm for speciation. It should be interesting to extend the niching approach to a PSO-based game learning context, and a bucket-brigade voting scheme might be employed to handle the selection of the specific strategy to apply for every move.

Some of the previously mentioned coevolutionary scoring schemes invented by Rosin and

Belew [112] also have some inherent niching characteristics, and their application to the game learning problem could certainly prove valuable.

Using other PSO approaches

Since the introduction of the original PSO algorithm in 1995, a large number of improved PSO algorithms have been developed. For the experimental work in this study, only the GCPSO algorithm by Van den Bergh [140] and the Binary PSO by Kennedy and Eberhart were investigated as possible alternatives.

Van den Bergh has also introduced the notion of cooperative swarms [138] [139], and the technique has shown to be successful in training product unit neural networks. In the case of neural network training, the principle of cooperative swarms lie in the assignment of a specific swarm to optimise a single layer of the neural network in seclusion from the other layers. It might be interesting to compare the performance of neural networks trained using the normal PSO approach to networks trained using the cooperative PSO approach, with specific application to the game learning domain.

Increase the configuration size

A large part of the parameter-specific experiments were conducted using a baseline configuration of 15 particles and 3 hidden nodes due to speed and time limitations. It stands to reason that using this small configuration will probably not produce the most intelligent player, as illustrated by the gains in performance when increasing the swarm size and hidden layers in certain instances. Extending the baseline configuration will definitely yield interesting experimental results, and will specifically impact a series of tests conducted in this study, including:

- The effectiveness of GCPSO.
- The performance of the sliding window technique.
- The application of the GPX scoring system.
- The effectiveness of the particle dispersement operator.
- The influence of the Hall of Fame.

Extend the tree depth

Given enough time, processing power, and the implementation of an optimised and robust game engine, extending the tree depth beyond 2-ply will also yield interesting results. It will then be possible to answer the research questions listed in chapter 7, specifically:

- Revisiting HITECH versus LOTECH on deeper ply-depths and searching for diminishing returns.
- What PSO parameters will enable good scalability of the neural network evaluation method in larger tree depths?
- Would it be possible to compress game-tree knowledge into a neural network?

Extending the tree depth will also allow for more thorough testing against commercial software, as was done by Chellapilla and Fogel [27].

Expand to other games

Extending the PSO-based coevolutionary game learning technique to even more challenging games like Go and the recently developed Arimaa [7] should prove interesting. These games have proven to not be easily 'solvable' using brute-force search methods such as game trees, due to the sheer exponential growth in size of the number of possible moves for each game state. Creating capable intelligent players without relying on human game expertise will be very difficult, and represent the future of game learning research.

Controlled training and improved benchmarking

As already mentioned, improving and optimising the game engine will allow for proper benchmarking against commercial software at deeper ply-depths. Some additional training and benchmarking options could involve the solving of Checkers end-game datasets – a technique sometimes used to test the performance of Chess engines. To date, the author have been unable to locate an electronic set of Checkers test problems. It is expected that either substituting or interleaving the coevolutionary training scheme with the dataset training will result in improved intelligent play. An investment in studying proper Checkers theory and strategic game analysis will also aid in providing a clearer understanding of the level of expertise attained by the evolved players.

Investigate PSO with multi-player IPD

As previously mentioned, the traditional IPD can be extended from a one-on-one encounter to a multi-player game. Extending the PSO-based strategy generation schemes to a multi-player IPD game will also be worthwhile investigating.

Bibliography

[1] S. G. Akl and M. M. Newborn. The principal continuation and the killer heuristic. In *Proceedings of the 1977 annual conference*, pages 466–473. ACM Press, 1977.

[2] American Checkers Federation, Official Rules. http://www.acfcheckers.com/rules.htm, Available [On-line], Accessed on 6 June 2003.

[3] T. S. Anantharaman, M. Campbell, and F.-H. Hsu. Singular extensions: Adding selectivity to brute-force searching. *Artificial Intelligence*, 43(1):99–109, 1990.

[4] P. J. Angeline. Using selection to improve particle swarm optimisation. In *IEEE International Joint Conference on Neural Networks (IJCNN1999)*, 1999.

[5] P. J. Angeline and J. B. Pollack. Coevolving high-level representations. In C. Langton, editor, *Proceedings of the 3rd Artificial Life Meeting*, 1994.

[6] P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms (GA-93)*, pages 264–270, 1994.

[7] Arimaa. Available [On-line], http://www.arimaa.com, Accessed on 6 June 2003.

[8] R. Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[9] R. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 32 –41. London: Pitman, and Los Altos, CA: Morgan Kaufman, 1987.

[10] R. Axelrod. Tit-for-tat strategies. In *Routledge Encyclopedia of International Political Economy*. Routledge, June 2001.

[11] R. Axelrod and L. D'Ambrosio. Announcement for bibliography on the evolution of cooperation. *Journal of Conflict Resolution*, 39:190, 1995.

[12] T. Bäck, D. B. Fogel, and T. Michalewicz, editors. *Basic Algorithms and Operators*, volume 1 of *Evolutionary Computation*. Institute of Physics Publishing, Bristol and Philidelphia, 1999.

[13] D. Beal. Null moves. *Advances in Computer Chess*, V, 1987.

[14] H. J. Berliner, G. Goetsch, M. S. Campbell, and C. Ebeling. Measuring the performance potential of chess programs. *Artificial Intelligence*, 43(1):7–21, 1990.

[15] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[16] T. M. Blackwell. Swarms in dynamic environments. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03)*, volume 2723 of *Lecture notes in Computer Science*, pages 1 – 12, 2003.

[17] T. M. Blackwell and P. J. Bentley. Don't push me! Collision-avoiding swarms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, 2002.

[18] A. D. Blair and J. B. Pollack. What makes a good co-evolutionary learning environment? *Australian Journal of Intelligent Information Processing Systems*, 4:166–175, 1997.

[19] C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases, 2002. University of California, Irvine, Department of Information and Computer Sciences, http://www.ics.uci.edu/∼MLRepository.html.

[20] R. Brits. Niching strategies for particle swarm optimisation. Master's thesis, Department of Computer Science, University of Pretoria, South Africa, 2002.

[21] A. L. Brundo. Bounds and valuations for abridging the search for estimates. *Problems of Cybernetics*, 10:225–261, 1963. Translation of Russian original in Problemy Kibernetiki, 10:141-150, May 1963.

[22] M. Buro. Toward opening book learning. In H. Iida, J. Schaeffer, J. W. H. M. Uiterwijk, and Y. Saito, editors, *Proceedings of the IJCAI-97 Workshop on Using Games as an Experimental Testbed for AI Research*, Nagoya, Japan, 1997.

[23] M. S. Campbell. Knowledge discovery in deep blue. *Communications of the ACM*, 42(11):65–67, November 1999.

[24] E. Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report IlliGAL report no. 95007, Illinois Genetic Algorithms Laboratory (IlliGAL), Urbana Champaign, IL, USA, July 1995.

[25] K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. In *Proceedings of the IEEE*, pages 1471–1496, 1999.

[26] K. Chellapilla and D. B. Fogel. Evolving neural networks to play checkers without expert knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, 1999.

[27] K. Chellapilla and D. B. Fogel. Anaconda defeats hoyle 6-0: A case study competing an evolved checkers program against commercially available software. In *Proceedings of Congress on Evolutionary Computation*, pages 857–863, La Jolla Marriot Hotel, La Jolla, California, USA, July 2000.

[28] ChessBase. Available [On-line], http://www.chessbase.com, Accessed on 3 October 2003.

[29] J. Churchill, R. Cant, and D. Al-Dabass. A new computational approach to the game of go. In *Proceedings of the 2nd Annual European Conference on Simulation and AI in Computer Games (GAME-ON-01)*, pages 81–86, London, 2001.

[30] Computational Intelligence Research Group - Swarm Intelligence Research, University of Pretoria. Available [On-line], http://cirg.cs.up.ac.za, Accessed on 28 November 2003.

[31] M. Clerc. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, volume 3, pages 1951–1957, Washington DC, USA, July 1999.

[32] M. Clerc and J. Kennedy. The particle swarm – explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, February 2002.

[33] Sir A. Conan Doyle. *The Complete Sherlock Holmes*. Gramercy, September 2002.

[34] F. A. Dahl. Honte, a go-playing program using neural nets. In J. Fürnkranz and M. Kubat, editors, *Workshop Notes: Machine Learning in Game Playing*, Bled, Slovenia, 1999. 16th International Conference on Machine Learning (ICML-99).

[35] P. J. Darwen and X. Yao. On evolving robust strategies for iterated prisoners dilemma. *Progress in Evolutionary Computation, LNAI*, 956:276–292, 1995.

[36] P. J. Darwen and X. Yao. Speciation as automatic categorical modularization. *IEEE Transactions on Evolutionary Computing*, 1(2):101–108, July 1997.

[37] P. J. Darwen and X. Yao. Does extra genetic diversity maintain escalation in a co-evolutionary arms race? *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3):191–200, July 2000.

[38] P. J. Darwen and X. Yao. Co-evolution in iterated prisoner's dilemma with intermediate levels of cooperation: Application to missile defense. *International Journal of Computational Intelligence and Applications*, 2(1):83–107, 2002.

[39] C. R. Darwin. *On the origin of species by means of natural selection or the preservation of favoured races in the struggle for life.* Murray, London, 1859. (New York: Modern Library, 1967).

[40] D. DeCoste. The future of chess-playing technologies and the significance of Kasparov versus Deep Blue. In *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence: Papers from the 1997 AAAI Workshop*, pages 9–13. AAAI Press, 1997.

[41] Deep junior - the reigning world computer chess champion. Available [On-line], http://www.x3dworld.com, Accessed on 22 September 2003.

[42] W. Durham. *Co-Evolution: Genes, Culture and Human Diversity.* Stanford University Press, Stanford, 1994.

[43] R. C. Eberhart and J. Kennedy. *Swarm Intelligence.* Morgan Kaufmann, 2001.

[44] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 84–89, San Diego, USA, 2000.

[45] A. P. Engelbrecht. *Sensitivity analysis of multi-layer feedforward neural networks.* PhD thesis, Department of Computer Science, University of Stellenbosch, 1999.

[46] A. P. Engelbrecht. *Computational Intelligence: An Introduction.* Wiley and Sons, October 2002.

[47] S. L. Epstein. Toward an ideal trainer. *Machine Learning*, 15:251–277, 1994.

[48] Fédération Internationale de l'Automobile, 2004 Formula One Sporting Regulations. Available [On-line], http://www.fia.com/sport/Regulations/f1regs.html, Accessed on 19 April 2004.

[49] R. Feldmann, P. Mysliwietz, and B. Monien. Distributed game tree search on a massively parallel system. In *Data Structures and Efficient Algorithms: Final Report on the DFG Special Joint Initiative*, volume 594 of *Lecture Notes in Computer Science*, pages 270–288. Springer-Verlag, 1992.

[50] S. G. Ficici, O. Melnik, and J. B. Pollack. A game-theoretic investigation of selection methods used in evolutionary algorithms. In *Proceedings of the Congress on Evolutionary Computation (CEC 2000)*, La Jolla Marriot Hotel, La Jolla, California, USA, 2000.

[51] M. Fierz. Basics of strategy game programming: part iii - scientific questions. Available [On-line], http://www.fierz.ch/strategy3.htm, Accessed on 17 June 2003.

[52] D. B. Fogel. Evolving behaviors in the iterated prisoner's dilemma. *Evolutionary Computation*, 1:77–97, 1993.

[53] D. B. Fogel. *Blondie24: Playing at the edge of A.I.* Morgan Kaufmann Publishers, 2001.

[54] L. J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[55] N. Franken and A.P Engelbrecht. Comparing PSO structures to learn the game of Checkers from zero knowledge. In *Proceedings of the Congress on Evolutionary Computation (CEC2003)*, Canberra, Australia, 2003.

[56] N. Franken and A.P Engelbrecht. Evolving intelligent game playing agents. *South African Computer Journal*, 2004. Accepted for publication (to appear in June 2004).

[57] N. Franken and A.P Engelbrecht. PSO approaches to co-evolve IPD strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2004)*, Portland, Oregon, USA, 2004.

[58] A. S. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10:484–491, 1957.

[59] J. Fürnkranz and M. Kubat. Chapter 2. In *Machines that Learn to Play Games*, pages 11–59. Nova Scientific Publishers, Huntington, New York, 2001.

[60] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[61] D. E. Goldberg and J. Richardson. Genetic algorithm with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[62] R. D. Greenblatt, D. E. Eastlake III, and S.D. Crocker. The Greenblatt chess program. In *Fall joint computing conference proceedings*, volume 31, pages 801–810, San Francisco, 1967. New-York ACM.

[63] P. Grosso. *Computer Simulations of Genetic Adaption: Parallel Subcomponent Interaction in a Multilocus model.* PhD thesis, University of Michigan, 1985.

[64] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, December 1968.

[65] P. G. Harrald and D. B. Fogel. Evolving continuous behaviors in the iterated prisoner's dilemma. *BioSystems*, 37:135–145, 1996.

[66] E. A. Heinz. Darkthought goes deep. *ICCA Journal*, 21(4), 1998.

[67] J. H. Holland. *Adapation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[68] J. H. Holland. Echo: Explorations of evolution in a miniature world. In J. D. Farmer and J. Doyne, editors, *Proceedings of the second conference on Artificial Intelligence.* Addison-Wesley, 1990.

[69] X. Hu. Particle swarm optimisation: Bibliography. http://www.swarmintelligence.org, Available [On-line], Accessed on 22 May 2004.

[70] X. Hu and R. Eberhart. Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Proceedings of the IEEE World Congress on Evolutionary Computation (CEC 2002)*, pages 1677–1681, Honolulu, Hawaii, 12–17 May 2002.

[71] IBM Research. Deep Blue. Available [On-line], http://www.research.ibm.com/deepblue, Accessed on 12 March 2003.

[72] A. Ismail and A. P. Engelbrecht. Training product units in feedforward neural networks using particle swarm optimization. *Proceedings of the International Conference on Artificial Intelligence*, pages 36–40, 1999.

[73] G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In *Proceedings of the Congress on Evolutionary Computation (CEC 2001)*, pages 995–1002, COEX Center, Seoul, Korea, May 2001.

[74] J. Kennedy. The behaviour of particles. In V. W. Porto, N. Saravanan, and D. Waagen, editors, *Proceedings of the 7th International Conference on Evolutionary Programming*, pages 581 – 589, 1998.

[75] J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1931–1938, Washington DC, USA, July 1999.

[76] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia, 1995.

[77] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 Conference on Systems, Man, and Cybernetics*, pages 4104–4109, 1997.

[78] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of Congress on Evolutionary Computing (CEC 2002)*, Honolulu, Hawaii, 2002.

[79] K. Kim and S. Cho. Evolving speciated checkers players with crowding algorithm. In *Proceedings of the IEEE World Congress on Evolutionary Computation (CEC 2002)*, pages 407–412, Honolulu, Hawaii, 12–17 May 2002.

[80] W. Knight. Every move you make. *New Scientist*, 179(2403):40, July 2003.

[81] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6:293–326, 1975.

[82] T. Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences, 1995.

[83] T. Kojima, K. Ueda, and S. Nagano. Flexible acquisition of various types of knowledge from game records: Application to the game of go. In H. Iida, J. Schaeffer, J. W. H. M. Uiterwijk, and Y. Saito, editors, *Proceedings of the IJCAI-97 Workshop on Using Games as an Experimental Testbed for AI Research*, pages 51–57, Nagoya, Japan, 1997.

[84] R. E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[85] J. R. Koza. *Genetic Programming: On the programming of computers by means of natural selection.* MIT Press, 1992.

[86] S. Kuhn. Prisoner's dilemma. *Stanford Encyclopedia of Philosophy*, 1997. Available [On-line], http://plato.stanford.edu/entries/prisoner-dilemma, Accessed on 25 November 2003.

[87] K. Lindgren. Evolutionary phenomena in simple dynamics. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life 2*, volume 10 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 295–312. Addison-Wesley, 1991.

[88] M. Løvberg and T. Krink. Extending particle swarm optimisers with self-organized criticality. In *Proceedings of the Congress on Evolutionary Computation (CEC2002)*, 2002.

[89] M. Løvberg, T. K. Rasmussen, and T. Krink. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01)*, San Francisco, 2001.

[90] J. Loy. The Standard Laws of Checkers [with comments by Jim Loy]. Available [On-line], http://www.jimloy.com/checkers/rules.htm, Accessed on 6 June 2003.

[91] A. Lubberts and R. Miikkulainen. Co-evolving a go-playing neural network. In *Proceedings of the GECCO-01 Workshop on Coevolution: Turning Adaptive Algorithms upon Themselves*, pages 14–19, 2001.

[92] B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, Fairfax, WA, USA, 1989.

[93] T. A. Marsland. Relative performance of alpha-beta implementations. In *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI'83)*, pages 763–766, Karlsruhe, Germany, 1983.

[94] T. A. Marsland and A. Reinefeld. Heuristic search in one and two player games. Technical Report TR 93-02, Department of Computer Science, University of Alberta, 1993.

[95] E. Mayr. *Animal Species and Evolution.* Belknap, Cambridge, MA, 1963.

[96] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feedforward neural network training. In *Proceedings of the IEEE Joint Conference on Neural Networks*, pages 1895–1899, Honolulu, Hawaii, 12–17 May 2002.

[97] L. Messerschmidt and A. P. Engelbrecht. Learning to play games using a PSO-based competitive learning approach. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, 2002.

[98] S. Milgram. The small world problem. *Psychology Today*, 22:61–67, 1967.

[99] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Portland, Oregon, USA, 1997.

[100] D. Montgomery. *Engineering Statistics, 2nd edition*. Wiley and Sons, 2001.

[101] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, 1998.

[102] J. Pearl. Scout: A simple game-searching algorithm with proven optimal properties. In *Proceedings of AAAI-80*, pages 143–145, Stanford, California, 1980.

[103] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Nearly optimal minimax tree search? Technical Report 94-19, Department of Computer Science, University of Alberta, 1994.

[104] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87:255–293, 1996.

[105] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[106] K. V. Price. An introduction to differential evolution. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 79–108, 1999.

[107] A. Rapoport. Book reviews: J.C. Harsanyi and R. Selten's A General Theory of Equilibrium Selection in Games. *Behavioral Science*, 34:154–158, 1989.

[108] I. Rechenberg. Cybernetic solution path of an experimental problem. Royal Aircraft Establishment, Library translation no. 1122, Farnborough, Hants, UK, 1965.

[109] I. Rechenberg. Evolutionary strategy. In J.M. Zurada, R. Marks II, and C. Robinson, editors, *Computational Intelligence - Imitating Life*, pages 147–159. IEEE Press, 1994.

[110] A. Reineveld. An improvement of the scout tree-search algorithm. *Journal of the International Computer Chess Association*, 6(4):4–14, 1983.

[111] C. W. Reynolds. Flocks, herds and schools: a distributed behavioural model. *Computer Graphics*, 21:25 – 34, 1987.

[112] C. D. Rosin and R. K. Belew. Finding opponents worth beating: Methods for competitive co-evolution. In *Proceedings of the 6th International Conference on Genetic Algorithms*, 1995.

[113] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[114] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[115] J. Schaeffer. Marion Tinsley: Human Perfection at Checkers? Available [On-line], http://www.wylliecheckers.pwp.blueyonder.co.uk/Tinsley.htm, Accessed on 22 September 2003.

[116] J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(11):1203–1212, 1989.

[117] J. Schaeffer. The games computers (and people) play. In M. V. Zelkowitz, editor, *Advances in Computers*, volume 50, pages 189–266. Academic Press, 2000.

[118] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2–3):273–290, 1992.

[119] M. Schoenauer and Z. Michaelewicz. Evolutionary computation. *Control and Cybernetics*, 26(3):307–338, 1997.

[120] H.-P. Schwefel. Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Diplomarbeit, Technische Universität Berlin, 1965.

[121] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[122] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE World Conference on Computational Intelligence*, pages 69–73, Anchorage, Alaska, May 1998.

[123] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimisation. In *Proceedings of Evolutionary Programming VII (EP98)*, pages 591–600. Springer Verlag, 1998.

[124] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimisation. In *Proceedings of the Congress on Evolutionary Computation (CEC1999)*, pages 1945 – 1950, 1999.

[125] Y. Shi and R. C. Eberhart. Fuzzy adaptive particle swarm optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001)*, volume 1, pages 101–106, 2001.

[126] A. Silva, A. Neves, and E. Costa. An empirical comparison of particle swarm and predator-prey optimisation. In *Proceedings of the 13th Irish Conference on AI and Cognitive Science*, volume 2464 of *Lecture notes in Artificial Intelligence*, page 103, 2002.

[127] G. C. Stockman. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12:179–196, 1979.

[128] P. N. Suganthan. Particle swarm optimizer with neighbourhood operator. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1958–1961, July 1999.

[129] C. E. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[130] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–278, 1992.

[131] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[132] E. Thiémard. Economic generation of low-discrepancy sequences with a b-ary gray code. Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

[133] K. Thompson. Computer chess strength. In M. R. B. Clarke, editor, *Advances in Computer Chess 3*, pages 55–56. Pergamon, 1982.

[134] K. Thompson. 6-piece endgames. *International Computer Chess Association Journal*, 19(4):215–226, 1996.

[135] A. Turing. Digital computers applied to games. In B. Bowden, editor, *Faster than Thought*, pages 286–295. Pitman, 1953.

[136] F. van den Bergh. Particle swarm weight initialization in multi-layer perceptron artificial neural networks. In *Development and Practice of Artificial Intelligence Techniques*, pages 41–45, Durban, South Africa, September 1999.

[137] F. van den Bergh. *An Analysis of Particle Swarm Optimizers.* PhD thesis, Department of Computer Science, University of Pretoria, South Africa, 2002.

[138] F. van den Bergh and A. P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, 26:84–90, November 2000.

[139] F. van den Bergh and A. P. Engelbrecht. Training product unit networks using cooperative particle swarm optimizers. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 126–132, Washington DC, USA, July 2001.

[140] F. van den Bergh and A. P. Engelbrecht. A new locally convergent particle swarm optimizer. *IEEE Conference on Systems, Man and Cybernetics*, October 2002.

[141] J. van Rijswijck. Learning from perfection: A data mining approach to evaluation function learning in awari. In T. A. Marsland and I. Frank, editors, *Computers and Games: Proceedings of the 2nd International Conference (CG-00)*, volume 2063 of *Lecture Notes in Computer Science*, pages 115–132, Hamamatsu, Japan, 2001. Springer-Verlag.

[142] J. S. Vesterstrøm and R. Riget. Particle swarms: extensions for improved local, multimodal, and dynamic search in numerical optimization. Master's thesis, Department of Computer Science, University of Aarhus, 2002.

[143] J. von Neumann. Zur theorie der gesellshaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

[144] J. von Neumann and O. Morgenstern. *Theory of Games and Economic behaviour.* Princeton University Press, 1944.

[145] P. J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioural sciences.* PhD thesis, Harvard University, Boston, USA, 1974.

[146] L. F. A. Wessels and E. Barnard. Avoiding false local minima by proper initialization of connections. *IEEE Transactions on Neural Networks*, 3(6):899–905, 1992.

[147] O. Williams. Garry Kimovich Kasparov biography. http://www.x3dworld.com, Available [On-line], Accessed on 22 September 2003.

[148] J. Wu and R. Axelrod. How to cope with noise in the iterated prisoner's dilemma. *Journal of Conflict Resolution*, 39(1):183–189, 1995.

[149] H. P. Young and D. Foster. Cooperation in the short and in the long run. *Games and Economic Behavior*, 3:145–156, 1991.

[150] D. Zuckerman. Can genes help helping? *Psychology Today*, 19(80), 1985.

APPENDIX A

Glossary

This appendix provides brief descriptions for acronyms and commonly used terms in this thesis.

ALLC:	Always Cooperate. Naïve IPD strategy that always cooperates with the opponent.
ANN:	Artificial Neural Network.
AOP:	Average Opponent Payoff. Used to measure the performance of strategies in the IPD.
APP:	Average Personal Payoff. Used to measure the performance of strategies in the IPD.
ATP:	Average Total Payoff. Sum of AOP and APP, used to measure the performance of strategies in the IPD.
BinPSO:	Binary PSO. Discrete form of the traditional PSO, invented by Eberhart and Kennedy. Used in this thesis to evolve IPD strategies.
Coevolution:	Competitive population-based training strategy, mimicking predator-prey interactions in nature that lead to escalating arms races – resulting in stronger individuals.
Experimental configuration:	The term used to refer to the selection of parameters for a specific experiment, encompassing PSO, neural network and coevolutionary-specific settings.
FSM:	Finite State Machine. Employed by Fogel to represent IPD strategies.

GA: Genetic Algorithm. Originally developed by Fraser and later formalised by Holland. Computational technique that models biological evolution by generating potential solutions (represented as chromosomes) to an optimisation problem.

GBest PSO: Global Best PSO. Original PSO neighbourhood structure, wherein all the particles share information about the best solution found thus far.

GCPSO: Guaranteed Convergence PSO. Developed by Van den Bergh, improves the performance of standard GBest PSO by not converging prematurely on local optima. Compared to other PSO algorithms to train intelligent Checkers players.

GPX: Grand Prix. Elements from Formula One Grand Prix racing are applied in the form of a coevolutionary racing scheme, as well as a particle dispersement operator to more accurately measure playing performance.

HOF: Hall of Fame. Coevolution technique that maintains previous best solutions in a fixed-size list. Currently evolved players compete against the HOF in order to maintain robustness and counter-act drift.

IPD: Iterated Prisoner's Dilemma. In its standard form, a two-player non-cooperative non-zero sum game. Defined by a payoff matrix, with the ultimate goal of mutual cooperation.

LBest PSO: Local Best PSO. Improvement on GBest PSO by restricting connectivity to a one-dimensional lattice structure, only allowing information sharing among immediate neighbours in variable space.

PSO: Particle Swarm Optimisation. Particle-based evolutionary technique developed by Eberhart and Kennedy, inspired by the flocking behaviour of birds.

PVLV: Pavlov IPD strategy developed to cope with noisy environments. Changes move if last move was not profitable.

RAND: Strategy in the IPD that makes moves at random.

STFT: Suspicious Tit-for-tat. Similar to TFT, but starts with a defection instead.

TDL: Temporal Difference Learning. Technique first used in the game-playing domain by Tesauro to train intelligent Backgammon players. A form of neural network-based machine learning.

TFT: Tit-for-tat. Winning IPD strategy submitted by Anatol Rapoport. Cooperates on the first move, and repeats opponent's last move thereafter.

TFTT: Tit-for-two-tats. A more generous version of TFT, invented to improve TFT's performance in noisy environments. Only defects after two successive defections by the opponent.

TTT: Tic-tac-toe. Intellectually simple 2-player game represented by a 3x3 grid. First player to align 3 pieces (X's or O's) in any direction wins. Used as initial test-bed for PSO training.

Von Neumann PSO: Extension of LBest PSO by increasing communication network to a two-dimensional lattice structure, only allowing information sharing among immediate neighbours in two-dimensional variable space.

VMax: Maximum velocity. Places a restriction on the maximum velocity of a particle, limiting explosive behaviour.

APPENDIX B

Definition of Symbols

This appendix list symbols used in this thesis along with their explanations.

ρ_i: Payoff i in the general IPD payoff matrix, where $i \in \{R, S, P, T\}$

ψ: Mirroring parameter used to construct the bottom half of a complete 64-bit strategy in the IPD, assuming the use of a custom (sin/cos-based) objective function.

β: The total number of cooperations for the best individual during a single epoch in the IPD.

τ: The average total number of cooperations for the population during a single epoch in the IPD.

a: Horizontal shift of a sin/cos function to generate the top half of an IPD strategy.

b: Maximum frequency for the sin part of the IPD generation function.

c: Frequency of the cos part of the IPD generation function. Can also be seen as the rate of change for the sin function's frequency.

d: Vertical shift of a sin/cos function to generate the top half of an IPD strategy.

R: Reward for mutual cooperation in the IPD. Set to 3 according to Axelrod's payoff matrix.

S: Sucker payoff in the IPD. Set to 0 according to Axelrod's payoff matrix.

P: Punishment payoff for mutual defection. Set to 1 according to Axelrod's payoff matrix.

T: Temptation payoff in the IPD. Set to 5 according to Axelrod's payoff matrix.

M: A value representing the performance measure defined by Messerschmidt *et al.* to evaluate the performance of an evolved Tic-Tac-Toe player against a random moving player. Also extended to evaluate Checkers players.

F: A value representing the new Franken performance measure that incorporates the number of games drawn in addition to games won or lost. Applied to evaluate Checkers players' performance against various opponents.

n: The total number of weights in a specific neural network structure. Directly representative of the dimension of a particle in search space.

Ω: A n-dimensional search space, which may be real-valued or discrete.

Φ: A swarm of particles in search space Ω.

$\vec{x}_i$: A vector representing a particle i's position in search space Ω.

$\vec{y}_i$: A vector representing the personal best position of particle i in search space Ω.

$\vec{v}_i$: A vector representing the velocity of particle i in search space Ω.

$\vec{z}$: A vector representing the neighbourhood best particle position in search space Ω.

t: Time step during the PSO algorithm's progression.

f: Represents a particle's fitness value.

ϕ: Inertia weight used in the PSO velocity update equation.

κ: Constriction coefficient defined by Clerc *et al.* to limit the velocity values of particles.

α: Global best particle index used in the GCPSO algorithm.

δ: Local search term added in the GCPSO algorithm to combat premature convergence on suboptimal solutions.

$\hat{\sigma}$: The standard deviation for a particular player playing against a random-moving player, as defined by Messerschmidt *et al.*

$\hat{\pi}$: A probability to winning a game playing as player one or player two against a random-moving player.

ϵ: The resultant value for the piece-count Checkers evaluation function.

μ: A bias factor for the calculation of the piece-count advantage.

ω: A random tie-breaker term to resolve decisions between equal board state evaluations.

APPENDIX C

Derived Publications

This appendix lists all the papers that have been published, or are currently under review, that were derived from work done in this thesis.

1. N. Franken and A. P. Engelbrecht. Evolving intelligent game playing agents. *South African Computer Journal*, Accepted for publication (to appear in June 2004).
2. N. Franken and A. P. Engelbrecht. Analysis of PSO approaches to co-evolve IPD strategies. *IEEE Transactions on Evolutionary Computation*, Submitted for review (June 2004).
3. N. Franken and A. P. Engelbrecht. Evolving intelligent game playing agents. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT 2003)*, Johannesburg, South Africa, 2003.
4. N. Franken and A. P. Engelbrecht. State of the art in game learning: teaching 'Deep Blue' to think. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT 2003), Post-Graduate Symposium*, Johannesburg, South Africa, 2003.
5. N. Franken and A. P. Engelbrecht. Comparing PSO structures to learn the game of checkers from zero knowledge. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2003)*, Canberra, Australia, 2003.

6. N. Franken and A. P. Engelbrecht. PSO approaches to co-evolve IPD strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2004)*, Portland, USA, 2004.