Coevolution of Neuro-controllers to Train Multi-Agent Teams from Zero Knowledge

by

Christiaan Scheepers

Submitted in partial fulfillment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

July 2013

Publication data:

Christiaan Scheepers. Coevolution of Neuro-controllers to Train Multi-Agent Teams from Zero Knowledge. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, July 2013.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

http://cirg.cs.up.ac.za/

http://upetd.up.ac.za/UPeTD.htm

Coevolution of Neuro-controllers to Train Multi-Agent Teams from Zero Knowledge

by

Christiaan Scheepers
E-mail: cscheepers@acm.org

Abstract

After the historic chess match between Deep Blue and Garry Kasparov, many researchers considered the game of chess solved and moved on to the more complex game of soccer. Artificial intelligence research has shifted focus to creating artificial players capable of mimicking the task of playing soccer. A new training algorithm is presented in this thesis for training teams of players from zero knowledge, evaluated on a simplified version of the game of soccer. The new algorithm makes use of the charged particle swarm optimiser as a neural network trainer in a coevolutionary training environment. To counter the lack of domain information a new relative fitness measure based on the FIFA league-ranking system was developed. The function provides a granular relative performance measure for competitive training. Gameplay strategies that resulted from the trained players are evaluated. It was found that the algorithm successfully trains teams of agents to play in a cooperative manner. Techniques developed in this study may also be widely applied to various other artificial intelligence fields.

Keywords: Cooperative coevolution, competitive coevolution, neural networks, charged particle swarm optimiser, zero knowledge, multi agent system, simple soccer.

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

"The ability to learn faster than your competitors may be the only sustainable competitive advantage."

Arie de Geus (1930)

"If you want to be incrementally better: Be competitive. If you want to be exponentially better: Be cooperative."

Anonymous

"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is most adaptable to change."

Anonymous

Acknowledgements

- **My dad, mother, and brother** without whose patience and support this work would never have been possible.
- **Professor Andries Engelbrecht** for his invaluable guidance and insight.
- **My colleagues at CIRG** for asking insightful questions and always challenging my results.
- **My friends** for all their support and always being interested in what I was doing.

Contents

List of Figures

List of Graphs

List of Algorithms

List of Tables

Chapter 1

Introduction

> *It took half a century from the Wright Brothers' first aircraft to the Apollo mission that sent a man to the moon and safely returned him to Earth. It took half a century from the invention of the digital computer to the creation of Deep Blue, a computer that beat then world champion chess player Garry Kasparov. By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, against the winner of the most recent World Cup, complying with the official rules of FIFA.*

On 4 July 1997 the NASA pathfinder mission performed a successful landing on the surface of Mars; the landing marked the deployment of the first autonomous robotics system, Sojourner. In 2004 two more autonomous rovers, Spirit and Opportunity, landed on Mars, much of their autonomous navigation system carried over from the Sojourner [98, 112]. Robots are being used more and more in situations where it would be either too dangerous or too impractical to send human beings. Space exploration is one example where direct control of the robot becomes impractical: the signal simply takes too long to reach earth and conditions might change before a new command can be sent to the robot explorer. Search-and-rescue robots can explore mines after accidents without risking more human lives, exploring areas where signals cannot penetrate. Self-learning automation systems allow for problems to be overcome without each problem being specifically designed for. The objective of this thesis is to develop such a self-learning algorithm, that would allow teams of agents to compete against one another without prior

knowledge of the game being played. In order to achieve this objective a coevolutionary cooperative and competitive particle swarm-based training algorithm will be developed.

1.1 Motivation

Even though the Mars rovers were considered *state-of-the-art*, their navigational algorithms allowed them to travel only extremely short distances without human interaction [98]. The Mars rovers' limited navigational system is a clear example of why better algorithms that allow robots to solve problems autonomously are needed. Better autonomous behaviour would allow for more complex missions to be conducted in shorter time frames.

The Robocup [91] initiative was created to promote research in the areas of robotics and artificial intelligence by offering a publicly appealing but formidable challenge. The techniques applied in training a team to win the game of soccer can be mapped to the techniques capable of solving real-world problems, such as further automating space exploration robots. The training technique presented in this thesis makes use of the particle swarm optimiser algorithm. Particle swarm optimisers have proved successful in training players for games such as Tic-Tac-Toe and Checkers [60]. These training techniques, however, often rely on knowing additional information about the problem domain being solved.

This study presents a new algorithm applying the particle swarm optimiser (PSO) as a neural network trainer, in a coevolutionary cooperative and competitive manner, capable of training soccer-playing robot teams in a simplified soccer game. In addition to training a team of players, the training is performed from zero knowledge, that is, no domain information is provided to the training algorithm; only the game outcome is known during training. Previous work has shown that the particle swarm optimiser combined with a competitive training mechanism has shown great potential in training neural networks as game agents [60, 108]. However, the complexities introduced by team based gameplay have not been explored before. The charged particle swarm optimiser is used in an attempt to further improve the training effectiveness of the standard particle swarm optimiser when used in a coevolutionary training environment.

1.2 Objectives

The main objective of this study is to develop a coevolutionary particle swarm-based algorithm to evolve gameplay strategies, specifically for Simple Soccer, using neuro-controlled gameplay agents. In working towards this goal, the following sub-objectives have been identified:

- to provide an overview of existing computational intelligence techniques that can be used in a coevolutionary algorithm to train neural networks.
- to provide an overview of the classic soccer-playing problem that captivates so many researchers and corporations.
- to develop a simulated soccer model that captures the complexity of the soccer problem while maintaining a low computational complexity. The low computational complexity is required due to the vast number of simulations conducted while evolving players in a coevolutionary fashion.
- to propose a training algorithm based on coevolution and particle swarm optimisation to train neuro-controllers gameplay strategies from zero knowledge.
- to investigate thoroughly the performance of the above mentioned algorithm and investigate methods of improving its performance while still complying with the zero-knowledge requirement. This investigation includes a discrete measurement analysis as well as a visual strategy analysis.

1.3 Contributions

The main contributions of this study are:

- The introduction of a new particle swarm optimisation-based coevolutionary algorithm capable of training teams of agents from zero knowledge. Previous particle swarm optimization-based coevolutionary training algorithms have focused on training individual agents and not teams of agents.

- The introduction of a new, generally applicable, relative fitness function that more accurately measures performance in a competitive coevolution environment. The additional accuracy is provided by taking into account the past performance of a player and is based on the official *FIFA league-ranking* system.
- The introduction of a soccer simulator satisfying the computational requirements in order to train agents in a coevolutionary training environment on today's hardware.
- The first application of the charged PSO algorithm in a coevolutionary framework to evolve soccer gameplay strategies.
- The discovery that the proposed algorithm resulted in clusters of particles forming in each swarm. Each cluster represents a different playing strategy. The clustered particles prevents convergence on a single playing strategy.
- The first application of using X-means clustering to cluster the particles from a PSO used to train players using neural networks. Each centroid found per swarm was shown to represent a unique player with its own playing strategy.
- The finding that the proposed training algorithm is capable of evolving teams of neuro-controlled players with different playing strategies.

1.4 Dissertation Outline

- **Chapter 2** covers all the relevant computational intelligence techniques and background on which the subsequent chapters build. PSO, competitive and cooperative coevolution, and artificial neural networks are discussed.
- **Chapter 3** gives a brief overview of the classic soccer-playing problem. The Simple Soccer model and enhancements specific to this work are introduced along with an analysis of its properties.
- **Chapter 4** presents the coevolutionary PSO-based training algorithm. Initial results are presented and analysed, enhancements are made to the original algorithm and PSO parameters are optimised.

- **Chapter 5** covers various relative fitness functions and introduces a new relative fitness function based on FIFA's league ranking. Parameter optimisation is repeated with the FIFA league ranking fitness function and results are discussed.
- **Chapter 6** focuses on identifying the various gameplay strategies that can be visually observed. The initial strategies appear weak, and possible reasons for the weak performance are explored. Neural network weight saturation is identified as one of the problems.
- **Chapter 7** focuses on improving the evolved gameplay strategies. Solutions to the neural network weight saturation problem are presented, as are additional enhancements to the algorithm. Clusters are identified in the particle swarms, each cluster centroid representing a different playing strategy.
- **Appendix A** provides a list of the important acronyms used or newly defined in the course of this work, as well as their associated definition.
- **Appendix B** lists and defines the mathematical symbols used in this work, categorised according to the relevant chapter in which they appear.
- **Appendix C** provides the algorithmic specifications for the simulations performed in this study.

Chapter 2

Background

"All men by nature desire knowledge..."

Aristotle (384 - 322 BC)

Training game agents to play intelligently from zero knowledge requires a number of artificial intelligence techniques. This chapter provides background insight into the various computational intelligence paradigms that influenced the work in this study. Artificial neural networks, evolutionary computation, particle swarm optimisation, and coevolution are covered. Work by other researchers that influenced this study is also discussed.

2.1 Introduction

The objective of this chapter is to provide the reader with an overview of the various computational intelligence techniques used throughout this study.

Artificial neural networks form the foundation for the neuro-controllers that control the actions of each agent. Section 2.2 discusses the various neural network architectures, initialisation strategies and learning paradigms. Typical applications of neural networks are also discussed in more detail.

Evolutionary computation is discussed in section 2.3. The various paradigms are briefly discussed to serve as background for the coevolution and related work sections.

Particle swarm optimisation is presented in section 2.4 as a stand-alone optimisation algorithm. The section takes an in-depth look at all the parameters involved in driving the particle swarm; at the same time, the various particle information sharing structures, variations of the standard particle swarm optimisation algorithm, and typical applications of the algorithm are discussed. Dynamic environments, that is, environments that change over time, such as the problem environment evaluated in this study pose a unique problem to optimisation algorithms. Variations of the particle swarm optimisation intended to deal with the challenges presented by dynamic environments are discussed.

The basic coevolution theory is presented in section 2.5. Both competitive and cooperative coevolution are discussed, as the work on zero knowledge training done in this thesis builds on both types of coevolution.

Finally, section 2.6 discusses the existing work that influenced this study. The training algorithm presented in this study is based on work done by a number of other researchers.

2.2 Artificial Neural Networks

The human brain can be seen as a vastly complex parallel computer performing thousands of computations every second to perform the everyday tasks of visual, auditory and touch processing to name but a few. Attempts to mimic the brain can be dated back to work done by Warren McCullough and Walter Pitts in the 1940s, who produced the first artificial neuron [104]. The neurons presented by McCullough and Pitts served as conceptual components that could be combined into circuits to perform computational tasks. Rosenblatt created a character recognition hardware neural network, called "Perceptron", in 1957 while at Cornel University [130]. Neural network research suffered a major setback after Minsky and Papert published their book "Perceptrons: An Introduction to Computational Geometry" in 1969 [110]. In the book, Minsky and Papert pointed out that perceptrons are only capable of learning linear separable patterns, making it impossible to learn the basic *XOR* function. A major decrease in funding for research was experienced, because of this publication, causing many researchers to leave the field.

In 1973 Grossberg demonstrated that multi-layer perceptrons were capable of learning the *XOR* function [65]. It was not until the 1970s with the discovery of the error back-propogation that interest and funding resumed [119, 133].

2.2.1 Artificial Neuron

An artificial neuron (AN), or neuron, is a mathematical model of a biological neuron [68]. Figure 2.1 depicts the model of a neuron. A neuron consists of three basic elements:

- A number of inputs, each associated with a weight, depicted by i_1, ..., i_J and w_1, ..., w_J in figure 2.1.
- An adder or multiplier to calculate the *net* input signal. If an adder is used, the value of *net* is calculated as $\sum_{j=1}^{J} i_j w_j$. This type of unit is known as a *summation unit*. If a multiplier is used, the value of *net* is calculated as $\prod_{j=1}^{J} i_j^{w_j}$. This type of unit is known as a *product unit*.
- An activation function f_{AN} and a threshold θ to calculate the output signal for the neuron.

A large number of activation functions exist. The choice of activation function f_{AN} for a neuron is largely problem-dependent. A collection of commonly used activation functions are listed here [46]:

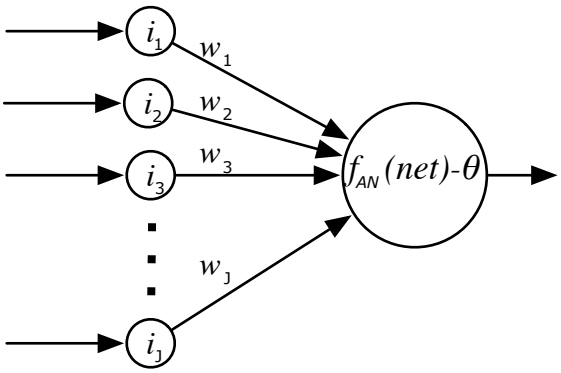


Figure 2.1: Artificial neuron.

- **Linear function:** $f_{AN}(net) = \beta net$, which produces a linear mapping scaled by a factor of β.
- **Step function:** $f_{AN}(net) = \begin{cases} \beta_1 & \text{if } net \geq 0 \\ \beta_2 & \text{if } net < 0 \end{cases}$, which produces a stepped output with lower bound β_1 and upper bound β_2, generally the step would be from 0 or -1 to 1.
- **Ramp function:** $f_{AN}(net) = \begin{cases} \beta & \text{if } net \geq \beta \\ net & \text{if } |net| < \beta \\ -\beta & \text{if } net \leq \beta \end{cases}$, which produces a combined step and linear output. Output is in the range of $-\beta$ to β with a linear function output for the domain $(-\beta, \beta)$.
- **Sigmoid function:** $f_{AN}(net) = \frac{1}{1+e^{-\lambda net}}$, which produces a continuous output between 0 and 1 with λ controlling the steepness of the function; normally $\lambda = 1$. The sigmoid function can be considered a continuous version of the ramp function.
- **Hyperbolic tangent function:** $f_{AN}(net) = \frac{e^{\lambda net} - e^{-\lambda net}}{e^{\lambda net} + e^{-\lambda net}}$ or $f_{AN}(net) = \frac{2}{1+e^{-\lambda net}} - 1$, which produces a hyperbolic tangent function with continuous output between -1 and 1 with λ controlling the steepness of the function; normally $\lambda = 1$. The hyperbolic tangent function can also be considered a continuous version of the ramp function.
- **Gaussian function:** $f_{AN}(net) = e^{-\frac{net^2}{\sigma^2}}$, which produces a Gaussian function with mean net where σ^2 is the variance of the Gaussian distribution.

The next section describes how multiple artificial neurons can be combined to form neural networks.

2.2.2 Artificial Neural Network Architectures

Most real-world problems are not linearly separable and cannot easily be solved by a single artificial neuron or a collection of independent artificial neurons. Artificial neural networks (ANN) consist of a number of artificial neurons that are connected together, usually in layers. The output from one neuron can be connected to the input of another

neuron. Three basic classes for interconnecting neurons exist, namely single-layer feed-forward, multi-layer feed-forward, and recurrent neural networks [76].

Single-layer feed-forward neural networks

Single-layer feed-forward neural networks (FFNNs) consist of an input layer of neurons connected directly to an output layer of neurons. Since no computation is performed on the input layer, only the output layer is counted [68].

Multi-layer feed-forward neural networks

Multi-layer feed-forward neural networks consist of an input layer of neurons connected to a hidden layer of neurons. The hidden layer of neurons can in turn be connected to either another hidden layer of neurons or to the output layer of neurons. Feed-forward networks allow for links that skip one or more layers, as long as the links between the neurons remain directional towards the output layer. This allows for an input neuron to connect directly with an output neuron, but not *vice-versa*.

Figure 2.2 depicts a three-layer feed-forward neural network with J input units, K hidden units, and L output units. The $(J+1)-th$ input unit and $(K+1)-th$ hidden unit are bias units with a value fixed to -1. The bias units represent the threshold values, θ, for the neurons of the next layer. Changing the weight connecting a bias unit with a neuron allows for the activation threshold to be changed for that neuron. The output values for the neural network can be calculated as follows, assuming that summation units are used:

$$o_l = f_{AN}(\sum_{k=1}^{K+1} v_{k,l} h_k) \tag{2.1}$$

with the hidden units:

$$h_k = \begin{cases} f_{AN}(\sum_{j=1}^{J+1} w_{j,k} i_j) & \text{if } k \in \{1, .., K\} \\ -1 & \text{if } k = K+1 \end{cases} \tag{2.2}$$

and the input units:

$$i_j = \begin{cases} i_j & \text{if } j \in \{1, .., J\} \\ -1 & \text{if } j = J+1 \end{cases} \tag{2.3}$$

This study makes use of three-layer FFNNs as neuro-controllers.

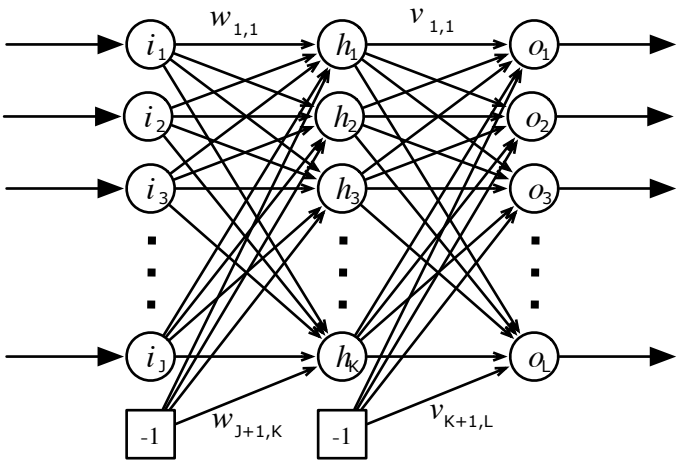


Figure 2.2: Basic feed-forward artificial neural network.

Recurrent neural networks

Recurrent neural networks (RNNs) allow for a feedback loop to exist between hidden (or output) neurons and input neurons. This feedback loop introduces a memory of sorts, increasing the network's learning capability when the network's input patterns exhibit temporal characteristics. The response of the neural network becomes dependent on the previous inputs and responses. Two well known types of recurrent neural networks are:

- **Jordan RNNs**: The activation values of the output neurons are passed back into the input layer by introducing a number of state units [78].
- **Elman RNNs**: The activation values of the hidden neurons are passed back into the input layer by introducing a number of context units [43].

Although not per definition a RNN time delay neural networks (TDNNs) use an input vector that includes the inputs from a number of discrete time steps (also referred to as the "time window") [162].

The next section describes the different learning paradigms that are employed to train a neural network.

2.2.3 Learning Paradigms

Artificial neural network training algorithms can be divided into three distinct paradigms, namely *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised Learning

Supervised learning requires that target outputs are available for all input patterns. Weights are adjusted proportional to the error between the neural network's predicted output and the target output.

Data patterns are divided into a *training*, a *generalisation*, and usually a *validation* set. The training phase makes use of the training set patterns; the generalisation set is used to quantify the neural network's ability to correctly classify unseen data patterns (this is known as the neural network's ability to generalise); the validation set can be used to stop the training process once the error is below a specified threshold. A well-trained neural network generally demonstrates a good generalisation ability.

Overfitting can occur if the architecture of the neural network is too large, choosing a non-representive training set containing noise, and over-training after optimal generalisation has been reached. Once overfitting occurs, the generalisation performance degrades as the training performance improves; essentially, the neural network memorises the noise in the training set [44].

Serveral algorithms have been developed to train neural networks in a supervised manner. Werbos developed one of the most popular learning algorithms based on gradient descent optimisation, called backpropagation [164]. Conjugate gradient optimisation [69] and LeapFrog optimisation [145] approaches have also been developed, though the details of these methods are beyond the scope of this study and the interested reader is referred to [7, 143, 144]. Global optimisation algorithms such as the particle swarm optimiser have been applied successfully to train neural networks [38, 66, 71, 83, 137, 153, 155, 156, 168, 169]. A detailed discussion of the particle swarm optimiser is deferred to section 2.4.

Unsupervised Learning

In situations where no target output vector exist for a specified input vector, unsupervised learning methods can be applied. Unsupervised learning methods find associations among input vectors that can be used, e.g. to perform clustering.

Kohonen developed one of the most popular unsupervised learning algorithms, called the learning vector quantizer (LVQ) [93]. An LVQ variant suited for unsupervised learning is the LVQ-I [93]. Kohonen also developed the self-organising feature map (SOM) [93]. The details of these methods are beyond the scope of this study and the interested reader is referred to [93] for more detail.

The particle swarm optimiser has also been used to train neural network game agent controllers in a coevolutionary fashion [58, 59, 60, 61]. In this case there is no target output vector. A more detailed discussion of coevolutionary learning with PSO is presented in section 2.6.2. The work in this study builds on this concept and presents a particle swarm optimiser-based training algorithm where neural networks directly control the individual game agents.

Reinforcement Learning

The final learning paradigm is reinforcement learning based on the idea of rewarding correct outputs and penalising incorrect outputs [150]. Sutton developed the TD(λ) algorithm in 1988 based on temporal difference learning which can be considered a reinforcement learning algorithm [149]. In 1992 Tesauro implemented the TD(λ) algorithm in his backgammon playing program, TD-Gammon [151]. Reinforcement learning is a slower process than the other paradigms; however, it is well suited to scenarios where not all of the training data is available at the same time.

2.3 Evolutionary Computation

Evolutionary computation (EC) refers to a number of population-based search and optimisation methods [5, 6] that simulate Darwinian evolution [31]. EC methods can be grouped into a number of different paradigms: genetic algorithms, genetic programming, evolution strategies, evolutionary programming, and differential evolution. Section 2.3.2

describes the different paradigms in more detail. Algorithm variations belonging to the different EC paradigms are referred to as evolutionary algorithms (EA). Each EA is based on the following fundamental principles of Darwinian evolution [31]:

- Organisms have a finite lifetime. Survival of the species requires offspring to be produced.
- Offspring vary to some degree from their parents.
- Organisms better adapted to their environment stand a better chance of surviving for longer and producing more offspring.
- Organisms inherit characteristics from their parents. Through natural selection this allows the species to adopt traits beneficial to their survival.

Each of the above principles can be directly mapped to an algorithmic approach to simulating evolution in order to solve an optimisation problem.

2.3.1 Evolutionary Process

An evolutionary algorithm runs over a finite number of generations. The evolutionary environment is represented by an optimisation problem. Each individual in the population represents a candidate solution to the optimisation problem. Individuals are considered fitter than another individual if they represent a better solution. At the end of each generation, selected individuals produce offspring to repopulate the population through a process called *reproduction*. Reproduction serves to preserve the traits that led an individual to a high level of fitness by passing some of the individuals' genetic material to the offspring. A *selection* operator determines which individuals produce offspring and survive to the next generation - this selection mechanism mimics the "survival of the fittest" aspect of biological evolution. As generations progress more and more, diversity is lost, as only the fit individuals survive. Less fit individuals are faced with extinction. To help reduce premature convergence and improve population diversity, a *mutation* operator may be applied to modify the offspring. Typically, this would modify a small number of genes randomly. Mutations may serve to increase or decrease the fitness of an individual.

The evolutionary process is repeated until the maximum number of generations is reached, an acceptable solution to the optimisation problem is found, or the fitness of the population does not increase for a number of generations, among others.

Each individual in the population poses two sets of evolutionary information, categorised as the *genotype* and the *phenotype*. The genotype represents the information required to calculate the fitness of the individual, encoded as the *genes* of the individual. The genes are passed from the parents to the offspring. In the case of a mathematical function the genes would be a real-valued vector representing all the variables required to evaluate the function. The phenotype represents the behavioural traits of an individual in a specific environment.

2.3.2 Evolutionary Computation Paradigms

A wide variety of evolutionary algorithms exist that implement the evolutionary process described above. A selection of the more popular paradigms are discussed below.

Genetic Algorithms

Holland popularised the genetic algorithm (GA) in 1975 [72]. Individuals are represented by *chromosomes* - typically, a bit string representation for the genotype would be used. Reproduction, selection, and mutation operators are used to drive the evolutionary process, as described in section 2.3.1. Evolution continues until a suitable solution has been found.

Many variants of Holland's GA have been developed [64, 67, 81]. These variants make use of different individual representations, selection operators, reproduction operators, and mutation operators, but still follow the same general evolutionary process as Holland's original GA [5, 6].

Genetic Programming

Koza [94, 95] extended the work done by Cramer [30], Hicklin [70], and Fujiki [63] in order to evolve executable programs. This led to the introduction of genetic programming (GP). GP represents the genotype of an individual as an executable program tree.

Elements from the terminal set, containing variables and constants, form the leaf nodes of the tree, while elements from the function set, containing mathematical, arithmetic, and/or boolean functions, form the non-leaf nodes of the tree. Similar to GAs, reproduction, selection, and mutation, operators are used. Reproduction involves randomly swapping subtrees to create offspring. Mutation involves randomly changing a node's values, deleting nodes, or adding new nodes to the tree.

Fitness calculation for GP is highly problem-dependent, but typically involves traversing the tree and recording the output using a sample of input test cases. The average performance over the samples can then be used as the fitness value.

Evolution Strategies

Originally devised by Rechenberg [126] and Schwefel [136], these strategies model the *evolution of evolution* with a focus on optimising the evolutionary process itself [127]. An evolutionary strategy (ES) evolves both the genotypic and the phenotypic representation of individuals, with a focus on the phenotypic evolution. ES make use of both reproduction and mutation to search both the search space and the strategy parameter space simultaneously.

Evolutionary Programming

Fogel [56] introduced evolutionary programming (EP) to evolve finite state-machines for use in time series prediction [53, 54]. Unlike EAs, EP does not make use of reproduction; instead, only mutation and selection are used. Mutations are randomly applied to the individuals to produce offspring. Fitness is calculated using a relative fitness measure, not an absolute fitness measure. Fogel and Fogel [55] extended EPs to allow for more general problems to be solved, such as the *travelling salesman problem* and real-valued vectors for function optimisation. Chellapilla and Fogel successfully used EP in a competitive coevolutionary model to train the Checkers program, *Anaconda* [22, 23] and Checkers program, *Blondie24* [52].

Differential Evolution

Although differential evolution (DE) does not strictly model any form of evolution it is typically listed along side EAs [147]. DE is a population-based search strategy where offspring is generated using a discrete cross-over operator and mutation. The mutation operator requires three parents to be randomly selected and mutation is implemented by augmenting one of the parents with a step size proportional to the difference vector between the other two parents. Parents are replaced in the population by their offspring only if the offspring is more fit than the parent.

2.4 Particle Swarm Optimisation

The particle swarm optimisation (PSO) algorithm [85] is a recently developed population-based optimisation method, with its roots in the simulation of the social behaviour of birds within a flock. First developed by Kennedy and Eberhart [85] in 1995, the PSO algorithm has been more successful in solving complex problems than traditional EC algorithms [87]. The basic PSO algorithm is presented in section 2.4.1. Various information sharing structures used by the PSO are discussed in section 2.4.2. Variations of the PSO algorithm are discussed in section 2.4.3. Applications for the PSO algorithm are discussed in section 2.4.4. Dynamic environments along with PSO variations that were developed for use in dynamic environments are discussed in section 2.4.5. Finally, more PSO variations are presented in section 2.4.6.

2.4.1 Basic PSO Algorithm

The population of a PSO algorithm, referred to as a swarm, consists of individuals referred to as particles. Each particle is represented by an n-dimensional vector $\vec{x}_i$ representing a candidate solution to an optimisation problem. The quality of the candidate solution represented by a particle is determined by evaluating a fitness function, $\mathcal{F}(\vec{x}_i)$. Changes to particle positions are based on a social component, a cognitive component, and an inertia velocity component.

The cognitive component is a weighted difference between the current position and

previously found best position, referred to as the personal best position of the particle. An information-sharing structure, represented by a neighbourhood topology, allows for information such as the particle positions to be shared with neighbouring particles. Information can be shared between particles only if they are defined as neighbours based on the information-sharing structure. The information-sharing structure is discussed in more detail in section 2.4.2. The social component, representing the socio-psychological tendency to emulate the success of neighbour particles, is calculated as a weighted difference between the current position and the neighbourhood best position. The position of each particle is updated based on its current position and velocity. The velocity in turn is based on the current velocity (the inertia component), a randomly weighted distance from the personal best position, and a randomly weighted distance from the neighbourhood best position.

The global best particle swarm optimisation (gbest PSO) algorithm allows each particle to share information, e.g. the best found position, with every other particle. For the gbest PSO all particles are considered neighbours of each other. The gbest PSO algorithm is shown in Algorithm 2.1.

The basic PSO velocity update equation is:

$$\vec{v}_i(t) = \vec{v}_i(t-1) + \vec{\rho_1}(\vec{x}_{pbest_i}(t) - \vec{x}_i(t)) + \vec{\rho_2}(\vec{x}_{gbest}(t) - \vec{x}_i(t)) \tag{2.4}$$

where $\vec{v}_i(t)$ is particle i's velocity at iteration t, $\vec{\rho_1}$, $\vec{\rho_2}$ are vectors each randomly uniformly distributed on $[0,1]^n$, $\vec{x}_i(t)$ is particle i's position at iteration t, $\vec{x}_{pbest_i}$ is the personal best position of particle i, and $\vec{x}_{gbest}$ is the global best particle position. The further away a particle's current position $\vec{x}_i(t)$ is from the personal best position $\vec{x}_{pbest_i}$ or global best position $\vec{x}_{gbest}(t)$ the larger the change to the particle's position to move back to those better-performing regions of the hyper-dimensional space. Kennedy further studied the vectors of random variables $\vec{\rho_1}$ and $\vec{\rho_2}$ and defined them as $\vec{\rho_1} = \vec{r_1}c_1$, $\vec{\rho_2} = \vec{r_2}c_2$ where $\vec{r_1}, \vec{r_2} \sim U(0,1)^n$ and $c_1, c_2 > 0$ are acceleration constants.

2.4.2 Information Sharing

PSO uses social interaction as the driving force behind the optimisation algorithm. The information sharing structure, also referred to as the neighbourhood topology, determines

Initialize the swarm, $O(t)$, of particles such that the position $\vec{x}_i(t)$ and personal best position $\vec{x}_{pbest_i}(t)$ of each particle $P_i(t) \in O(t)$ is uniformly randomly distributed within the hyperspace, let $\vec{v}_i(t) = 0$ with $t = 0$.

repeat:

for all *particles* $P_i(t)$ *in the swarm* $O(t)$ **do**

Evaluate the performance $\mathcal{F}(\vec{x}_i(t))$, using the current position $\vec{x}_i(t)$.

Compare the performance to the personal best position found thus far:

if $\mathcal{F}(\vec{x}_i(t)) < \mathcal{F}(\vec{x}_{pbest_i}(t))$ then

$\vec{x}_{pbest_i}(t) = \vec{x}_i(t)$

Compare the performance to the global best position found thus far:

if $\mathcal{F}(\vec{x}_{pbest_i}) < \mathcal{F}(\vec{x}_{gbest}(t))$ then

$\vec{x}_{gbest}(t) = \vec{x}_{pbest_i}(t)$

end for

for all *particles* $P_i(t)$ *in the swarm* $O(t)$ **do**

Change the velocity vector of the particle:

$\vec{v}_i(t) = \vec{v}_i(t-1) + \vec{\rho}_1(\vec{x}_{pbest_i}(t) - \vec{x}_i(t)) + \vec{\rho}_2(\vec{x}_{gbest}(t) - \vec{x}_i(t))$

Move the particle to a new position:

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

$t = t + 1$

end for

until *all particles converge* **or** *iteration limit is reached.*

Algorithm 2.1: PSO algorithm to minimise the value of objective function $\mathcal{F}$.

which particles are allowed to communicate with one another.

It is noteworthy that the neighbourhood of a particle is usually constructed using indices assigned to the particles and not geometrical information such as position or distance measures of any sort. Using indices to construct the particle neighbourhood allows for information to be exchanged between particles irrespective of their current position. The particle neighbourhood can also be kept constant as particle indices do not change, ensuring information is shared in a predetermined structure to facilitate exploration.

The remainder of this section describes a sample of commonly found neighbourhood

structures in more detail.

No neighbourhood structure

The individual best velocity model does not make use of any neighbourhood structure, because the velocity update equation does not make use of the social component. Therefore, no exchange of information takes place in the individual best PSO. Effectively, the behaviour is that of multiple hill-climbers. Particles may converge on different solutions. This model generally performs worse than any of the other PSO models [153].

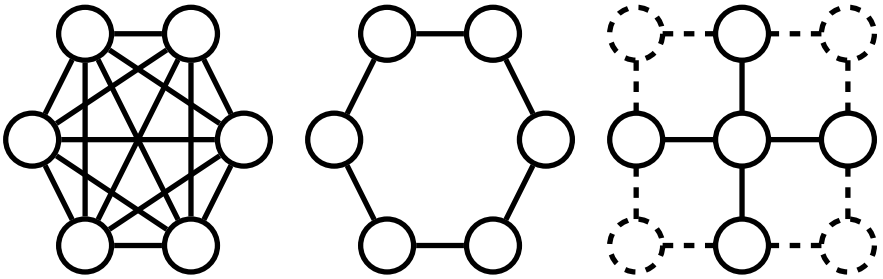
Star neighbourhood structure

The star neighbourhood structure connects all particles with all other particles as illustrated in figure 2.3(a). The entire swarm forms one neighbourhood. Each particle imitates the best solution found by the entire swarm by moving towards the global best position. Because of the fast information sharing, the star neighbourhood leads to faster convergence than other neighbourhood structures. A PSO which uses the star neighbourhood structure is referred to as the global best, or *gbest*, PSO. The fast convergence of the *gbest* PSO makes it susceptible to getting stuck in local optima [153].

Ring neighbourhood structure

The ring neighbourhood structure connects each particle with its m immediate neighbours. In the case of $m = 2$ a particle communicates with only the immediately adjacent neighbours as illustrated in figure 2.3(b). Each particle attempts to imitate its best neighbour by moving towards the best position found in the neighbourhood. It should be noted that the neighbourhoods overlap as illustrated in figure 2.3(b). This overlap in neighbourhoods facilitates the exchange of information between all the particles, and convergence on a single solution. Convergence is typically slower than that of the star neighbourhood, but solution quality for multimodal problems is typically higher as more of the search space is explored [153]. A PSO that uses the ring neighbourhood structure is referred to as a local best, or *lbest*, PSO.

It should be noted that the gbest PSO is a special case of the lbest PSO where $m = |O(t)| - 1$.



(a) Star (b) Ring (c) Von Neumann

Figure 2.3: PSO neighbourhood structures.

Von Neumann neighbourhood structure

The Von Neumann model, first introduced by Kennedy and Mendes [88], makes use of a grid-like neighbourhood structure as illustrated in figure 2.3(c), extending the one dimensional structure of the *lbest PSO* to two dimensions. Each particle shares information with its direct neighbours based on grid positions (above, below, left, and right). Emperical studies have shown that the Von Neumann neighbourhood structure outperforms other structures in a number of problems [88, 157], including game agent training problems [58, 59, 60].

Other neighbourhood structures

A wide variety of neighbourhood structures has been studied by researchers in the field.

Mendes, Cortez and Rocha [105] also presented the pyramid, four clusters and square neighbourhood structures, showing performance improvements when applied to certain optimisation problems. Kennedy and Mendes [88] presented the idea of random created neighbourhoods where no fixed neighbourhood structure existed beforehand.

Watts and Strogatz [163] introduced small world networks, based on the small world phenomenon [109]. Small world networks are obtained by randomly rewriting connections, and two particles are considered neighbours if they are connected. Kennedy [84] and Kennedy and Mendes [88] introduced the small world principle to PSO neighbourhood structures and found that these random connections in the neighbourhood improved performance for certain problems.

Suganthan [148] introduced a neighbourhood structure not based on indices, but introduced dynamic neighbourhoods based on Euclidean distances between particles. Starting with a neighbourhood of size one, the neighbourhood size gradually increases until the neighbourhood contains all the particles in the swarm. This growing of the neighbourhood allows the PSO to exploit the exploration ability of lesser connected neighbourhoods with the exploitation ability of fully connected neighbourhoods. Hu and Eberhart [73] introduced a dynamic neighbourhood structure with connections based on particle fitness instead of using the particle indices.

No one neighbourhood structure has been proven to perform better than all others for all problems. It has, however, been noted that the fully connected neighbourhood structures perform better for unimodal problems, while the lesser-connected neighbourhood structures perform better for multimodal problems [84, 88, 105, 157].

2.4.3 PSO Variations

Empirical results have shown that the standard PSO algorithm is capable of solving a variety of problems [153]. However, empirical results also show that the PSO algorithm does not consistently converge on optima in hyper-dimensional search spaces [153]. Divergent particle behaviour as well as exploding particle velocities have been observed. To address these problems a variety of variations have been introduced to improve the convergence of the PSO. A few of the basic variations or modifications that have been made to the standard PSO are discussed.

Velocity Clamping

Analysis of the PSO velocity update equation, as shown in equation (2.4), showed that extreme differences between the personal best position, $\vec{x}_{pbest_i}$, or the neighbourhood best position, $\vec{x}_{nbest_i}$ and the particle position, $\vec{x}_i$, may lead to an explosion in velocity, $\vec{v}_i(t)$ [153]. Similary, large inertia values may cause a gradual increase in velocity leading up to an explosion in velocity. This explosion in velocity may cause particles to leave the search space boundaries.

The first solution to delay the explosion of particle velocities is to simply clamp the velocity [42]. Clamping involves specifying a maximum velocity $V_{max,j}$ and a min-

imum velocity $-V_{max,j}$ for each dimension j of the hyper-dimensional search space. If a particle's velocity exceeds the maximum or minimum for any dimension, the velocity for that dimension is restricted to the maximum or minimum value. Small values for $V_{max,j}$ restrict exploration and promote exploitation; however, particles might get stuck in local optima without the velocity to escape. Large values for V_{max} restrict exploitation and promote exploration. This may cause particles to simply "jump" over good regions. $V_{max,j}$ is usually selected proportionally to the domain of each dimension such that $V_{max,j} = \delta(x_{max,j} - x_{min,j})$, where $\delta \in (0, 1]$; $x_{max,j}$ and $x_{min,j}$ are respectively the maximum and minimum domain values for dimension j. Clamping the velocity of a particle may also lead to a change in direction since not all dimensions are clamped proportionally to each other.

Emperical studies have shown that the optimal value for δ is problem-dependent [114, 139]. Using a dynamically changing value for the maximum velocity improves the optimisation algorithm performance in some cases. Schutte *et al.* proposed to change the maximum velocity if no improvement in the swarm is seen for a predefined number of iterations [135]. Schutte *et al.* proposed that the new maximum velocity should be calculated as $V_{max,j}(t) = \beta V_{max,j}(t-1)$, where β decreases from 1 to 0.01. The decrease in velocity promotes exploitation towards the end of the search by slowing down the particles. Fan used a similar approach where the maximum velocity decreases based on an exponential decay rate [49].

Clerc [25], and Clerc and Kennedy [27] found that V_{max} is not necessary if a constriction coefficient, $\mathcal{K}$, is added to the standard velocity update equation as follows:

$$\vec{v}_i(t) = \mathcal{K}(\vec{v}_i(t-1) + \vec{\rho}_1(\vec{x}_{pbest_i}(t) - \vec{x}_i(t)) + \vec{\rho}_2(\vec{x}_{nbest}(t) - \vec{x}_i(t))) \tag{2.5}$$

where

$$\mathcal{K} = 1 - \frac{1}{\rho} + \frac{\sqrt{|\rho^2 - 4\rho|}}{2} \tag{2.6}$$

with $\rho = \rho_1 + \rho_2 > 4$. Small values for the constriction coefficient, $\mathcal{K} \approx 0$, encourage local exploitation and faster convergence. Larger values for the constriction coefficient, $\mathcal{K} \approx 1$, encourage exploration. Shi and Eberhart concluded that the constriction coefficient approach should be preferred above explicit velocity clamping, although, clamping is still necessary in many cases [39].

Inertia Weight

Shi and Eberhart introduced the inertia weight parameter, w, to address the velocity explosion problem by controlling the trade-off between exploration and exploitation within the swarm [138]. Initially, the goal was to remove the need for velocity clamping in addition to controlling the *exploitation-exploration* trade-off [40]. It should be noted, however, that this goal was not completely achieved as velocity clamping is still required in certain cases [153].

The inertia weight controls the *exploitation-exploration* tradeoff by weighing the momentum term as follows:

$$\vec{v}_i(t) = w\vec{v}_i(t-1) + \vec{\rho_1}(\vec{x}_{pbest_i} - \vec{x}_i(t)) + \vec{\rho_2}(\vec{x}_{nbest} - \vec{x}_i(t)) \tag{2.7}$$

Values of $w < 1$ reduce the contribution made by the momentum term and depending on the values of c_1 and c_2 lead to a velocity of zero [153]. Small velocities in turn encourage local exploitation. Values of $w > 1$ increase the momentum and will eventually lead particles to achieve maximum velocity. Particles will diverge and leave the search space. From this it becomes clear that large values facilitate exploration.

Shi and Eberhart also investigated the optimal value for w and found that it should be tuned per problem for optimal results [139]. Shi *et al.* also developed a fuzzy system to find optimal values for w. A slight performance improvement was noted using this fuzzy system [141]. Linear decreasing values for the inertia weight have been proposed by Naka *et al.* [111], Ratnaweera *et al.* [125], Suganthan [148], and Yoshida *et al.* [167], where w is linearly decreased from 0.9 to 0.4 for good results. Various nonlinear decreasing strategies have also been developed. Peram *et al.* [122] and Naka *et al.* [111] proposed using the formula $w(t+1) = \frac{(w(t)-0.4)(n_t-t)}{n_t+0.4}$ with $w(0) = 0.9$ and n_t equal to the total number of iterations. Venter and Sobieszczanski-Sobieski [159, 160] proposed the formula $w(t+1) = \alpha w(t')$ where t' is the iteration of the previous inertia change, $\alpha = 0.975$, $w(0) = 1.4$, and $w(n_t) = 0.35$. Venter and Sobieszczanski-Sobieski adjusted the inertia weight only if the variation in fitness for a 20% sample of particles is below a predefined threshold. Schutte *et al.* [135] used the same inertia update equation as Venter and Sobieszczanski-Sobieski to update the inertia weight when no improvement in the global best is seen in a predefined number of iterations. Clerc [26] proposed a

scheme where the inertia weight is changed proportional to the relative improvement of the swarm. This approach required less iterations to find an optimum. Zheng *et al.* investigated the effect of increasing inertia from 0.4 to 0.9 [170, 171].

Van den Bergh and Engelbrecht [154], and Trelea [152], found that the parameters w and c_1 and c_2 are dependent on one another. To guarantee convergent trajectories, these parameter values have to satisfy the following relation:

$$w > \frac{c_1 + c_2}{2} - 1 \tag{2.8}$$

2.4.4 Applications

PSO has been successfully applied to a variety of mathematical function minimisation and maximisation problems [140], multi-objective optimisation, constraint based optimisation, and many real-world problems to name a few areas [45].

Kennedy and Eberhart applied the PSO early on to train neural networks to learn the XOR function [85]. In order to train a neural network, the mean squared error function is used as the PSO fitness function. Each particle in the swarm represents the weight vector and biases for the whole neural network. The swarm can thus be seen as a collection of neural networks, each with a different performance in terms of accuracy. PSO has been been the focus of much research in the area of neural network training. The interested reader is referred to [42, 47, 48, 58, 59, 60, 61, 86, 155, 156].

2.4.5 Dynamic Environments

The PSO was originally developed to solve static optimisation problems. That is, the search space does not change during the optimisation process. It has been shown that the PSO is extremely efficient in optimising problems in static environments [153].

In contrast to static enviornments, dynamic environments have objective functions that change over time [45]. These changes may cause position changes in optima, new optima may appear, and existing optima may disappear.

The standard PSO suffers from two major limitations in the presence of dynamic environments [36]:

- **Diversity loss**: At convergence particles oscillate around the global best position [115, 116, 154, 156]. If a minor environmental change causes the optimum to move slightly or even if a new optimum is introduced close to the global best position, the swarm automatically adapts towards the new optimum. However, if a new optimum is introduced far away from the swarm's global best position, the swarm has little chance of finding the new optimum due to the lack of exploration taking place. As the search progresses, the particle dispersion, or swarm diversity, decreases [153]. Without a high diversity the swarm does not explore the search space. Techniques that either maintain or reintroduce diversity after an environmental change are therefore required.

- **Outdated memory**: After an environmental change the personal best position of particles may no longer represent good positions. If the optimum close to the personal best position is removed due to the environmental change, the swarm might be pulled by the particle's cognitive components back towards an area that performs more weakly. If no position that performs better than the previously seen personal best positions can be found, the swarm may never recover to find a good optimum. The same problem holds for the neighbourhood best position, if used. Techniques that recalculate the personal best positions are needed to counter this problem.

Various modifications to the PSO have been proposed to adapt the PSO for use in a dynamic environment. A brief summary of a few of the techniques is provided below, but for a more in-depth review of the PSO algorithm in dynamic environments the interested reader is referred to [36].

Particle Reinitialisation Strategies

Carlisle and Dozier [17] demonstrated that the cognitive component, or particle memory, and neighbourhood best position lead a particle to previously known good positions. However, in the presence of changing optima a previous good position might be considerably worse after a change in the environment. This is a major cause of the problems experienced in dynamic environments [17]. Both Carlisle and Dozier [17] and Hu and

Eberhart [74] proposed to reset the personal best positions of all particles to the current position after a change is detected.

Hu and Eberhart [75] further noted that clearing a particle's memory is effective only if the swarm has not yet converged. In a converged swarm, where $\vec{v}_i(t) \approx \vec{0}$, exploration is no longer taking place. To increase exploration, Hu and Eberhart combined the memory reset with reinitialisation of the swarm. Reinitialisation requires that the particle position and personal best position be set to uniformly random chosen positions. Resetting the personal best position prevents the swarm from returning to no longer valid optima, while full particle reinitialisation further increases the diversity of the swarm by moving particles to new positions, improving exploration ability. Eberhart and Shi [41] identified three strategies for swarm reinitialisation, choosing a suitable strategy requires some knowledge about the environmental change that occurred:

- **No reinitialisation:** If the change in the optimal position is small and the swarm has not yet converged, then no reinitialisation is necessary.

- **Partial reinitialisation:** Reinitialising only a percentage of the swarm allows for previous good positions (previous global best position) to be retained while increasing the diversity of the swarm. An experimental study by Hu and Eberhart [75] showed that partial reinitialisation with small reinitialisation percentages was effective for small changes to the environment, while larger reinitialisation percentages were required for larger changes to the environment.

- **Complete reinitialisation:** Reinitialising the complete swarm is mostly inefficient: as the swarm loses all its memory, the swarm will automatically search again from the start, increasing computational complexity, and the swarm can in worst cases converge on a worse optimum that it had found previously. Complete reinitialisation is recommended only for extreme changes to the environment where partial initialisation does not work.

Carlisle and Dozier [18, 16] further noted that resetting the personal best position might not always be the best choice as the personal best might still be a better position than the current position after a change in the environment. They proposed that the

personal best and current positions should both be re-evaluated after a change in the environment is detected in order to determine the new personal best position.

The above mentioned reinitialisation strategies all require knowledge of changes to the environment. Carlisle [19], and Carlisle and Dozier [18, 16] proposed the use of a sentry particle or sentry particles to detect changes in the environment. Each sentry particle stores a copy of its fitness value from the previous iteration. If the stored fitness value differs from the re-calculated fitness value for the current iteration it could only be due to an environmental change. A single sentry particle returns information about a small region of the search space. This small region may be static in the presence of certain environmental changes causing changes not to be detected. Randomly choosing sentry particles each iteration may improve detection performance. Hu and Eberhart [74, 75] proposed using the global best position and global second best position as sentry particles instead of randomly selecting sentries.

Modified Inertia Update

Eberhart and Shi [41] proposed using the standard PSO along with a randomly selected inertia coefficient, as well as removing velocity clamping, for use in dynamic environments. The inertia weight, $w(t)$, is randomly selected from a Gaussian distribution at each iteration such that $w(t) \sim N(0.72, \sigma)$; σ is chosen such that $w(t)$ is not predominantly greater than one.

As mentioned in section 2.4.3, the inertia weight, in dependence with c_1 and c_2, controls the PSO exploration versus exploitation trade-off. Velocity clamping is removed as it only serves to limit exploration (as previously discussed in section 2.4.3). In a dynamic environment it is not known whether exploration or exploitation is desired at any timestep - randomly selecting the inertia and thus random switching between favouring exploration or exploitation proved efficient for certain dynamic problems.

Split Adaptive Particle Swarm Optimisation

Coelho *et al.* developed the Split Adaptive PSO for use in dynamic environments [28]. The *split adaptive PSO* divides the swarm into two sub-swarms, the social swarm and the cognitive swarm. The social swarm particles are updated according to the social only

model. Particles in the social swarm are attracted to the best neighbourhood position without a tendency to move to previous good positions allowing to explore the search space. The velocity update equation for the social swarm is defined as:

$$\vec{v}_{social(i)}(t) = w\vec{v}_{social(i)}(t-1) + c_2\vec{r}_2(\vec{x}_{gbest}(t) - \vec{x}_i(t)) \tag{2.9}$$

The cognitive swarm particles are updated according to the cognition only model, with the exception that the momentum term is that of the social swarm. Particles in the cognitive swarm are attracted to their previous good positions allowing exploitation of these good regions in the search space. The velocity update equation for the cognitive swarm is defined as:

$$\vec{v}_{cognative(i)}(t) = w\vec{v}_{social(i)}(t-1) + c_1\vec{r}_1(\vec{x}_{pbest_i}(t) - \vec{x}_i(t)) \tag{2.10}$$

If a change in the environment is detected, the personal best position, $\vec{x}_{pbest_i}$, is set to the current particle position if the recalculated fitness is worse than the current fitness. Both swarms used acceleration constants $c_1 = c_2 = 2$ with inertia $w \sim U(0.4, 0.9)$.

Attractive and Repulsive Particle Swarm Optimisation

Vesterstrøm and Riget [161] developed the attractive and repulsive particle swarm optimisation (ARPSO) algorithm. Swarm diversity, d_{swarm}, is measured continuously. Once the swarm diversity drops below a certain threshold, d_{low}, the behaviour of the swarm is modified to a repulsive force by inverting the cognitive and social velocity equation components. Once the swarm diversity is above another predefined threshold, d_{high}, the swarm behaviour is restored by changing back to the standard PSO velocity equation. The *attractive and repulsive PSO* velocity equation switching condition and resulting formulas are defined as follows:

$$\vec{v}_i(t) = \begin{cases} \vec{v}_i(t-1) + \vec{r}_1 c_1(\vec{x}_{pbest_i} - \vec{x}_i(t)) + \vec{r}_2 c_2(\vec{x}_{nbest} - \vec{x}_i(t)) & \text{once } d_{swarm} > d_{high} \\ \vec{v}_i(t-1) - \vec{r}_1 c_1(\vec{x}_{pbest_i} - \vec{x}_i(t)) - \vec{r}_2 c_2(\vec{x}_{nbest} - \vec{x}_i(t)) & \text{once } d_{swarm} < d_{low} \end{cases} \tag{2.11}$$

The repulsion phase continuously "injected" diversity into the swarm, allowing the swarm to continue to search for new, more optimal areas in the hyper-dimensional search space. The ARPSO performed significantly better on multimodal problems and only marginally worse on easy problems when compared to the standard PSO.

Fine-Grained Particle Swarm Optimisation

Li and Dam [100] tested the Von Neumann neighbourhood structure along with the standard PSO position and velocity update equations. It was shown that the Von Neumann neighbourhood structure restricts the interaction between particles enough to maintain a higher degree of diversity than other neighbourhood structures. The restricted flow of information causes a more gradual convergence, leading to better performance in dynamic environments.

Charged Particle Swarm Optimisation

Blackwell and Bentley [10, 11, 12] developed the charged particle swarm optimisation (CPSO) algorithm to improve the standard PSO's performance when used in dynamic environments. The CPSO is based on the analogy of electrostatic energy with charged particles. The CPSO introduces a repulsion force into the standard PSO, leading to two opposing forces: (1) an attraction to the centre of mass of the swarm, which facilitates convergence (as defined by the standard PSO); and (2) a repulsion force that preserves swarm diversity in an attempt to avoid premature convergence on a non-optimal solution in case the environment changes.

The CPSO repulsion force is introduced by adding a new acceleration term, $\vec{a}_i$, to the PSO velocity equation. The inter-particle repulsion is determined based on the distance between particles: particles closer than the perception limit, R_{plimit}, and further away than the core limit, R_{core}, influence the magnitude of the repulsion force. Blackwell [10] defined a constant repulsion for the case where the distance become less than R_{core} in order to avoid extremely large repulsion forces, which in turn would cause an explosion effect, driving particles very far away from the global best position. The perception limit helps to improve convergence on the global best position by limiting the repulsion force on far away particles.

The PSO velocity update equation is defined as follows:

$$\vec{v}_i(t) = w\vec{v}_i(t-1) + \vec{\rho_1}(\vec{x}_{pbest_i} - \vec{x}_i(t)) + \vec{\rho_2}(\vec{x}_{gbest} - \vec{x}_i(t)) + \vec{a}_i(t) \tag{2.12}$$

with the new acceleration term defined as follows:

$$\vec{a}_i(t) = \sum_{l=1, i \neq j}^{|O(t)|} \vec{a}_{il}(t) \tag{2.13}$$

where $\vec{a}_{il}(t)$, defining the magnitude of the repulsion force between particle $P_i \in O(t)$ and particle $P_l \in O(t)$, is defined as follows:

$$\vec{a}_{ij}(t) = \begin{cases} (\frac{Q_i Q_l}{||\vec{x}_i(t) - \vec{x}_l(t)||^3})(\vec{x}_i(t) - \vec{x}_l(t)) & \text{if } R_{core} \leq ||\vec{x}_i(t) - \vec{x}_l(t)|| \leq R_{plimit} \\ (\frac{Q_i Q_l (\vec{x}_i(t) - \vec{x}_l(t))}{R_{core}^2 ||\vec{x}_i(t) - \vec{x}_l(t)||}) & \text{if } ||\vec{x}_i(t) - \vec{x}_l(t)|| < R_{core} \\ 0 & \text{if } ||\vec{x}_i(t) - \vec{x}_l(t)|| > R_{plimit} \end{cases} \tag{2.14}$$

where Q_i is the charge magnitude of particle P_i.

Three types of swarms were identified:

- **Neutral swarms:** $Q_i = 0$ for all particles $P_i \in O(t)$; neutral swarms occur where the charge magnitude for every particle in the swarm is zero, and thus no repulsion occurs. The standard PSO can be seen as a neural swarm.
- **Charged swarms:** $Q_i > 0$ for all particles $P_i \in O(t)$; charged swarms occur where every particle in the swarm has a change magnitude greater than zero, and all particles experience a repulsion force provided that the distance between two particles $\in [R_{core}, R_{plimit}]$.
- **Atomic swarms:** Half the swarm is charged ($Q_i > 0$) and the other half is neutral ($Q_i = 0$).

Experimental results have shown that the inclusion of the CPSO repulsion maintains better diversity than the ordinary PSO [9, 10, 12]. It was also shown that the CPSO performs efficiently when used in dynamic environments. Experimental results also showed that atomic swarms tended to outperform the other swarm types [12]. Blackwell also conducted a theoretical analysis of the CPSO. For more details the interested reader is referred to [8]. All the experimental work in this study makes use of the CPSO with atomic swarms due to the efficiency of the CPSO in the presence of changing optima.

Quantum Swarm Optimisation

Blackwell and Branke [13, 14] found that the main drawback of the CPSO was that the algorithm has a computational complexity of $O(n^2)$. The quantum swarm optimisation (QSO) algorithm was developed to reduce the computational cost of the CPSO. The QSO contains both *neutral particles* and *quantum particles*. Similar to how the CPSO's charged particles are based on the analogy of electrostatic energy, the QSO's quantum particles are based on the concept of the quantum cloud. Each iteration the quantum particles are placed randomly within a multi-dimensional sphere, B_n, of radius r_{cloud} centered at the global best particle. This random placement of the quantum particles eliminates the need for a repulsion force calculation and leads to reduced computational cost.

The QSO algorithm makes use of the standard particle swarm velocity update equation as defined in equation (2.4). The position update equation is modified to incorporate the random movement of the quantum particles as follows:

$$\vec{x_i}(t) = \begin{cases} \vec{x_i}(t-1) + \vec{v_i}(t) & \text{if } Q_i = 0 \\ U(\hat{y} - r_{cloud}, \hat{y} + r_{cloud})^n & \text{if } Q_i > 0 \end{cases} \tag{2.15}$$

Empirical research has shown that the QSO tends to outperform other dynamic PSO algorithms for many dynamic environment problems [36].

2.4.6 Other PSO variations

Silva *et al.* introduced a PSO algorithm based on the *predator-prey* concept [142]. The *predator-prey PSO* divides the swarm into two groups: predator particles and prey particles. Predator particles are attracted to the prey particles, while prey particles are driven by the *standard PSO* velocity equations, except for when predator particles are close by, in which case the prey particles are "pushed" away from the predator particles. The predators serve to increase swarm diversity and avoid premature convergence in the presence of changing optima.

Løvberg and Krink introduced a PSO algorithm based on the *self-organised criticality* principle [103]. A criticality rating is calculated for each particle based on the level of convergence of the particle on the current best position. Once the criticality rating for a

particle surpasses a certain threshold, the particle is repelled using one of two methods. The first is to simply reinitialise the particle, thus erasing the particle's memory. The second is to increase the particle's velocity based on a ratio of the level of criticality of the particle, thus forcing the particle far away again and allowing for more exploration to take place.

The interested reader is advised to consult the literature for additional adaptations made to the PSO for use in dynamic environments [34, 36, 90, 99, 101, 154].

2.5 Coevolution

The training algorithm presented in this study builds on the concepts of coevolution [37]. This section outlines the concepts of coevolution as applied to this study. An overview of coevolution in general is provided, followed by a detailed discussion of both competitive and cooperative coevolution.

2.5.1 Overview

The Oxford Dictionary defines coevolution as the influence of closely associated species on each other in their evolution [33]. Coevolution, as applied in this study, is a computational model of the biological coevolutionary processes.

Coevolution does not rely on an explicit fitness function to determine the fitness of the individuals. Instead, a relative fitness is calculated in relation to the other individuals in the population or populations.

Coevolutionary algorithms can be divided into two classes [153]:

- **Cooperative:** These algorithms model an environment where an increase in the fitness of one species leads to an increase the fitness of another species. An example from nature would be the symbiotic relationship between orchids and insects. Orchids require insects to fertilise other orchids in order to reproduce [32]. Without the insects the orchids would be unable to reproduce. Without reproduction, the orchids will systematically become extinct.

- **Competitive:** These algorithms model an environment where the individuals of one or more population compete for victory against one another. An example from nature is the feeding behaviour of cheetahs. In order to survive, the impala needs to run faster and be more alert, otherwise the cheetah will catch the impala. The cheetah in turn needs to run even faster than the impala, otherwise the cheetah will starve to extinction. As the impala improves itself, the cheetah must also improve itself. Both species increase their fitness at the cost of the other.

This study makes use of both competitive and cooperative coevolution to train game agents. Sections 2.5.2 and 2.5.3 provides more detail of each.

2.5.2 Competitive Coevolution

Competitive coevolution deals with individuals from one or more species co-evolving in competition. In competitive coevolution, each individual attempts to improve itself in relation to the other individuals in order to survive.

A relative fitness function must be developed to drive the evolutionary process [2]. The relative fitness function measures the performance of an individual relative to the performance of individuals in the other populations. The resulting measurement is a relative value, quantifying how well an individual performs relative to other individuals. No absolute measure of how close to the optimal solution an individual is, is needed - this property makes competitive coevolutionary algorithms extremely useful for cases where no absolute measure of optimality exists or even for situations where the optimal solution is truly unknown.

The lack of an absolute fitness function does introduce drawbacks. An obvious drawback was noted by Blair and Pollock [15]. A random mutation may allow one individual (or lucky novice) to beat the best individual (or reigning champion). Although the lucky novice beats the champion the playing strategy might exploit a single weakness and lack robustness. Without robustness the playing strategy would fail to win against other strategies without the specific weakness. Blair and Pollock named this phenomenon the *Buster Douglas Effect* after James "Buster" Douglas who became world champion for nine months in 1990 after knocking out Mike Tyson. Tyson was the undefeated champion and the favourite at the time. Douglas had a 1 in 35 chance of winning, according

to The Mirage Casino in Las Vegas. Douglas retired from boxing after losing the first match defending the championship against Evander Holyfield. Techniques that counter the *Buster Douglas Effect* were developed by Rosin and Belew [131, 132], where improved ways of evaluating the relative fitness value for an individual were developed.

Two important aspects that play a role in competitive coevolution are relative fitness evaluation and fitness sampling. The next two sections discuss each in more detail.

Relative Fitness Evaluation

Rosin and Belew [131] introduced methods to reward individuals in relation to individuals in the competing population(s). Three approaches capable of achieving this were presented:

- **Simple fitness:** A sample of individuals is selected, using a fitness sampling technique, from the individuals in the competing population(s) to serve as a competition pool for the current population's individuals' fitness evaluations. Each individual is evaluated against the individuals in the competition pool. The resulting fitness values are simply the sum of individuals from the competition pool that the current individual beat. Simple fitness can also be calculated within a single population by measuring against individuals from within the population.

- **Fitness sharing:** A sharing function is introduced to measure similarity in the individual's. Simple fitness (as described above) is calculated for each individual in the primary population. The simple fitness value is then divided by the number of similar individuals in the population. *Fitness sharing* rewards extraordinary individuals more, improving their chance of survival. The higher chance of survival allows extraordinary individuals to survive longer which in turn leads to improved diversity in the population.

- **Competitive fitness sharing:** Fitness is calculated as the sum of the inverse of the number of individuals from the primary population capable of beating each individual from the opposing population(s). *Competitive fitness sharing* for an

individual $P_{1,n}$ of population P_1 is defined as

$$f(P_{1,n}, P_2) = \sum_{m=1}^{|P_2|} \frac{1}{N(P_1, P_{2,m})} \tag{2.16}$$

where the function $N(P_1, P_{2,m})$ is the number of individuals from population P_1 that beat the individual $P_{2,m}$ from population P_2 and $|P_2|$ is the total number of individuals in population P_2.

A good fitness evaluation reveals as much information about an individual's performance as possible. The more information the fitness evaluation provides, the more information is available to the optimisation algorithm about the fitness landscape.

Fitness Sampling

The above-described methods all require a selection of opponents for the competition pool. The selection method is not fixed and a number of different techniques exist. Rosin and Belew [131] investigated and proposed methods for selecting the sample of opponents for the competition pool. The following selection schemes were identified to select a sample of opponents:

- **All versus all selection:** Each individual is tested against all possible competing individuals. This scheme is computationally expensive as a population of n individuals will result in a total of $\frac{n(n-1)}{2}$ competitions to take place when calculating the relative fitness values. It should however be noted that this scheme also provides the most accurate information about the relative fitness of the individuals.

- **Random selection:** Each individual is evaluated against a random selection of competing individuals. The sample size for this scheme can be chosen as one or more individuals. Random selection is computationally less expensive than *all versus all* selection, but it also provides less accurate measurements of the relative fitness for the individuals. Less accurate measurements are the result of less competitions taking place as less information about an individual's performance can be obtained.

- **Tournament selection:** Each individual is evaluated against a sample of opposing individuals, selected by making use of a "mini" tournament to select the group of competing individuals based on their relative fitness. The purpose of the "mini" tournament is to select good performing individuals for the competition pool. The selection of only good performing individuals improve the overall quality of the measurement against the competition pool.
- **All versus best selection:** Each individual is evaluated against the competing individual with the highest relative fitness value (the most fit or strongest competing individual).
- **Bipartite selection:** For bipartite selection the competing individuals are divided into two teams. Each individual is then compared against only one of these two teams. This scheme requires less computational power than the *all versus all* selection scheme while attempting to match the performance due to the high diversity of individuals in each team.

The above described fitness sampling methods allow for a competition pool to be constructed from a subset of individuals. The competition pool in turn is used as part of the relative fitness evaluation. The relative fitness function can be described as noisy when using any fitness sampling scheme other than the *all versus all* scheme in which case all the individuals are in the competition pool.

Hall of Fame

Rosin and Belew [131, 132] introduced a technique to aid in exploring the search space even further and avoiding early stagnation due to exploitation by extending the elitism mechanism as used in standard evolutionary algorithms to coevolution in the form of the "Hall of Fame" (HOF). The HOF maintains a collection of the best performing individuals from each generation since the evolutionary process has started. Individuals from the current generations compete against the HOF members for entrance into the "Hall of Fame". This collection of previous best performing individuals helps to combat the loss of exploration of the search space introduced by the "moving target" (or changing optimal position in the search space) by preserving past good characteristics.

Competitive Coevolutionary Particle Swarm Optimisation

Messerschmidt and Engelbrecht introduced a competitive coevolutionary PSO-based training algorithm [106, 107]. Their approach was based on work done by Chellapilla and Fogel [21, 22, 23] and Fogel [52]. The algorithm was used to train game agents for the games of Tic-Tac-Toe and Checkers. Section 2.6.2 provides a detailed discussion of their work.

Messerschmidt and Engelbrecht assigned a fitness value to each particle by accumulating the score while playing against other particles in the swarm. Each game played was awarded a score based on the outcome of the game. Winning a game was rewarded by 1 point, drawing resulted in no points, and losing was penalised with a score of -2. The accumulated score provided enough performance information for the PSO algorithm. Using this approach, a competitive PSO algorithm was devised, as summarised in algorithm listing 2.2.

This study presents a variation on the approach presented by Messerschmidt and Engelbrecht to accommodate the training of a whole team of game agents where each team and player combination is treated as a separate population. This is in contrast to the above approach, where a single population trained players for both player 1 (first) and player 2 (second) positions.

2.5.3 Cooperative Coevolution

Cooperative coevolution models an ecosystem of multiple species living in symbiosis with one another [124]. Cooperation is encouraged between the different species (or populations) by rewarding each species based on how well the individuals cooperate with those of other species (or populations) to solve a target problem.

The divide-and-conquer problem-solving strategy can be seen as a form of cooperative coevolution, where a target problem is divided into multiple smaller problems. Each species (or subpopulation) represents and optimises a smaller and simpler component of the problem. All the different subsets are combined to form the optimised solution to the problem. A fitness sharing mechanism is used to introduce positive feedback such that the success of one subpopulation improves the fitness of the combined solution. In

Create and initialise a swarm, $O(t)$, of neural networks.
repeat:
    Add each personal best position to the competition pool.
    Add each particle to the competition pool.
    for each *particle* P_i *(or NN) in the swarm* $O(t)$ **do**
        repeat:
            Select an opponent from the competition pool.
            Play a game against the selected player, playing as first player.
            Record if the game was won, lost or drawn.
            Play a game against the selected player, playing as second player.
            Record if the game was won, lost or drawn.
        until *a predefined number of games has been played.*
        Determine a score for each particle based on number of wins, losses, or draws.
        Compute new personal best positions based on scores.
    end for
    Compute new neighbourhood best position.
    Update particle velocities.
    Update particle positions.
  end for
until *the swarm converges* **or** *the iteration limit is reached.*
Select the global best particle as the trained neural network game agent.

Algorithm 2.2: Competitive PSO algorithm to train neural network game agent (asynchronous implementation).

this study, each player in a team is evolved cooperatively in its own population. A fitness sharing mechanism is used to provide feedback based on the performance of the team.

Potter identified several categories of issues that arise when attempting to solve a problem in a cooperative coevolutionary manner [124]. Each of these issues is discussed in detail below. The applicability of these issues will become apparent in chapter 4 as they apply to this study.

Problem Decomposition

The first issue Potter identified is how to decompose the components of a problem into several subcomponents. The choice of the number of subcomponents and the role each subcomponent plays determines how well a problem can be solved through a cooperative coevolutionary approach. Potter referred to this as *"problem decomposition"*.

An example of problem decomposition in an optimisation problem is that of the *relaxation method*, also commonly referred to as the *sectioning method* or *coordinate strategy* [62, 146]. The relaxation method can be used to solve an optimisation problem with n variables by fixing $n-1$ of the variables while optimising the value of the remaining variable. Effectively, the relaxation method breaks the optimisation problem down into n subcomponents, each with one variable to optimise.

Finding the appropriate problem decomposition can be quite difficult. For example, if it is known that the n variables of an optimisation problem are mathematically independent of each other, then an approach such as the above suffices. However, more often that not the optimal breakdown is not known *a priori*. In cases where dependence exists between variables a more ideal breakdown would be to cluster variables with dependence on each other into the same subcomponents.

Potter suggested that if no appropriate decomposition can be determined before applying a cooperative coevolutionary approach, the search for the optimal decomposition can be made an explicit part of the process. A good decomposition should have a selective advantage over poor decompositions, which emphasises the importance of finding a suitable decomposition when attempting to solve a problem.

Interdependencies between Subcomponents

The second issue Potter identified is how to handle interdependencies that exist between subcomponents. Each subcomponent can be seen as a standalone problem, graphically moving to a higher position in its own *fitness landscape*. As mentioned previously, a problem decomposition may lead to having subcomponents with complex interdependencies - in effect, the fitness landscapes are linked to each other. A change in one subcomponent leads to "deforming" or "warping" of the linked subcomponent fitness landscapes [82]. Because of the adaptive nature of evolutionary algorithms, they are

generally well suited to problems with a single dynamic fitness landscape. In order to cater for multiple dynamic fitness landscapes, arising from the complex subcomponent interdependencies, the evolutionary computation paradigm must be extended to allow for interaction between dependent subcomponents.

The interdependency between subcomponents can easily be seen through an example [153]. Consider the objective function:

$$f(\vec{x}) = x_1^2 + x_2^2 \tag{2.17}$$

A simple decomposition for the function would be:

$$f(\vec{x}) = f_1(x_1) + f_2(x_2) \tag{2.18}$$

where $f_1(x_1) = x_1^2$ and $f_2(x_2) = x_2^2$. The function $f(\vec{x})$ is said to be linearly separable as it can be rewritten as the sum of n functions where $\vec{x}$ is a vector of n dimensions.

Adding a single term to the problem does not make the graphical representation any more difficult:

$$f(\vec{x}) = x_1^2 + x_2^2 + 0.25(2 - x_1)(1 - x_2) \tag{2.19}$$

However, it is no longer possible to separate the function linearly. The suitable decomposition now becomes:

$$f(\vec{x}) = f_1(x_1) + f_2(x_2) \tag{2.20}$$

where $f_1(x_1) = x_1^2 + 0.25(2 - x_1)(1 - X_2)$ and $f_2(x_2) = x_2^2 + 0.25(2 - X_1)(1 - x_2)$ with X_1 and X_2 being constants representing x_1 and x_2 in $f_2(x_2)$ and $f_1(x_1)$ respectively.

If the optimal value of x_1 changes, it might deform or warp the fitness landscape of x_2 and a new optimal value needs to be found for x_2, and *vice-versa*. The optimisation now requires multiple iterations for both subcomponents to be optimised.

Credit Assignment

The third issue Potter identified is how to determine the contribution each subcomponent makes to the collective solution. Potter referred to this as the *credit assignment problem*. The credit assignment problem can be traced back to work done by Samuel in 1959 [134]. Samuel attempted to apply machine learning techniques to the game of

checkers. He faced the credit assignment problem in determining how much credit or blame to assign to individual elements of an attribute vector for assessing the worth of different checkerboard configurations. The importance of assigning credit correctly is highlighted by the fact that Darwinian evolution passes characteristics or genes on to future generations based on their fitness [31]. A good credit assignment allows individuals that contribute to the success of a candidate solution to survive. Making the credit assignment as accurate as possible is critical to building a good, cooperative coevolutionary algorithm. The credit assignment function is also referred to as the *fitness sharing* function [45].

A common credit assignment approach is to make direct use of the fitness function for the combined solution. Using the fitness function requires a combined solution to be constructed. A template solution vector is constructed from the best individuals in each subcomponent and only the current subcomponent's individuals are substituted for the fitness evaluation. Variations of this approach include using more than one template vector as well as using the previously discussed selection techniques to select the individuals that make up the template vector. As the template vector stays constant during the evaluation of individuals from a single population, a ranking can be determined. The normal evolutionary operators can make use of this ranking to drive the evolutionary process.

Population Diversity

The fourth issue Potter identified is that of population diversity. According to the Darwinian principle of natural selection, in which highly fit individuals survive longer and reproduce more often, the population diversity drops as only the genes representing above average regions of the search space tend to survive. Standard evolutionary algorithms lose diversity, which does not really matter since diversity is needed only to facilitate a reasonable exploration of the search space in order to find an optimal area. Potter found that diversity in each of the subpopulations needs to be maintained when evolving in a cooperative coevolutionary fashion. Several techniques exist to maintain population diversity, including crowding and niching particularly adapted for use with the GA. Alternatively, diversity can be maintained through genetically isolated species.

Parallelism

The final issue Potter identified is that of parallel computing. When applying cooperative coevolution to increasingly difficult problems, the computational requirements increase accordingly. A solution to speed up the computation is required. Because evolutionary algorithms evolve a population of solutions rather than a single solution, Potter found them to be inherently parallel. Three parallel processing distribution techniques were identified:

- **Fitness evaluation:** If determining the fitness of an individual is computationally expensive, the fitness evaluation of all the individuals in the population can be done in parallel. This easy distribution will result in a near-linear speed-up.
- **Coarse-grain:** The coarse-grain approach allows for large subpopulations to evolve independently on a few processors with occasional migration of individuals between processors. If no interdependencies exist between subcomponents, this approach requires no interprocess communication until the final solution is assembled. Even with interdependencies, only occasional communication is required and the low communication overhead may still allow for near-linear speed-up.
- **Fine-grain:** The fine-grain approach allows for distributing individuals or small subpopulations on many processors, with interaction between processors and localized mating rules. This approach incurs a high communication overhead, which must be taken into account when designing the algorithm configuration for a specific problem.

Cooperative Particle Swarm Optimisation

Van den Bergh and Engelbrecht first introduced the cooperative particle swarm optimisation split (CPSO-S) algorithm [153, 155]. The CPSO-S algorithm applied the component breakdown (or divide-and-conquer) technique, making the modularity of the components explicit by using a separate PSO to optimise each component (or dimension) of the target problem. This decomposition meant that an n-dimensional problem required n one-dimensional swarms to solve.

Since each swarm represents only part of the solution to the target problem CPSO-S adds context in the form of a *context* vector, and each particle is substituted into the correct position in the *context* vector; the standard fitness function for the problem is then used to evaluate the fitness of this *context* vector. The resulting measured fitness value is then assigned as the fitness for the particle that was substituted into the *context* vector. CPSO-S constructs the *context* vector by combining the best-performing particles from each swarm; each particle being evaluated is evaluated along side the current best-performing particles from the other swarms. The measured fitness value can be used to score (or rank) the performance of any particle against that of the other particles within the same swarm. This score (or rank) is then used to perform the normal processing as defined by the PSO algorithm being used for each swarm; CPSO-S uses a round-robin process to process the different PSO algorithms in parallel.

Van den Bergh and Engelbrecht identified that the CPSO-S algorithm did not perform well in cases where correlated variables existed in the function being optimised. To address this problem they introduced the CPSO-S_K extension to the CPSO-S algorithm. Ideally, components in the vector that are correlated should be grouped in a single swarm; however, identifying the correlated components can be difficult or impossible beforehand. The CPSO-S_K algorithm attempts to address the correlated component problem by splitting the solution vector into K parts, where $K \leq n$. It should be noted that the CPSO-S algorithm is a special case of the CPSO-S_K algorithm where $K = n$; similarly, the normal PSO is a special case where $K = 1$. It should also be noted that no explicit restriction is placed on the PSO algorithm to be used for each of the K swarms.

Van den Bergh and Engelbrecht introduced another Cooperative PSO algorithm based on the CPSO-S_K called the CPSO-H_K algorithm. CPSO-H_K is a hybrid algorithm, switching each iteration between the CPSO-S_K algorithm and the global convergence PSO algorithm [153]. The CPSO-H_K algorithm attempts to exploit the benefits of both algorithms, namely faster convergence on certain functions and guaranteed convergence on a local minimiser. CPSO-H_K exchanges information between the two algorithms by replacing a randomly chosen particle (or particles) with well-performing particles from the other algorithm. This type of information exchange can be seen as a form of *blackboard information exchange*, as termed by Clearwater *et al.* [24]. Observations showed

that too much information exchange actually leads to degrading the performance of the CPSO-H_K algorithm.

2.6 Related Work

The previous sections provided background on various computational intelligence techniques as used throughout this study. The training algorithm presented in this study builds on work done by Chellapilla and Fogel [22], Messerschmidt and Engelbrecht [106], Franken and Engelbrecht [60], and Conradie and Engelbrecht [29]. The simulated soccer model on which the algorithm is tested is based on work done by Jeong and Lee [77], and Fehervari and Elmenreich [50]. A detailed discussion of these follows.

2.6.1 Evolving Neural Networks for Checkers

Chellapilla and Fogel presented a competitive coevolutionary programming-based approach capable of training Checkers agents [22]. No domain-specific expert knowledge was required by their approach. Their approach made use of neural networks, with hyperbolic tangent activation functions as game state evaluators. Each checkers board was represented by a vector of length 32. Each component of the vector represented a square on the board and assumed a value from $\{-K, -1, 0, 1, K\}$, where K, a real value in the range $[1.0, 3.0]$, is the value assigned to a king and 1 is assigned to any other piece. The sign indicated whether the piece belonged to the player or the opponent.

An alpha-beta game tree is calculated up to a specified ply-depth of d, at which point the neural network assigns a desirability factor to each of the leaf nodes. Typical ply-depths were 6, 8, or 10, chosen based on the time available per move. An agent then uses a standard minimax algorithm to determine the next move.

Individuals in the evolutionary algorithm represented all the neural network weights, the neuron biases, as well as the value of K. Chellapilla and Fogel ran their evolutionary algorithm for 250 generations with 15 individuals in the population. Only the fittest individuals survived to subsequent generations. Fitness was calculated for each individual by playing against five randomly chosen other individuals from the population. A score of $+1$ was awarded for a win, 0 for a draw, and -2 for a loss.

Chellipilla and Fogel took their best-evolved agent and over a period of two weeks played 90 games against human opponents on an internet gaming website. The most impressive result obtained was a draw against an expert player, then ranked number 18 out of 40000 registered players. Overall performance indicated the trained agent could play competitively against human players both winning and losing matches.

The use of minimax along with the limited ply-depth was identified as possible areas that could do with improvement. Games using larger ply-depths tended to perform better than games with smaller ply-depths, although computational complexity increased substantially with larger ply-depths. Without using larger ply-depths the agent failed to see moves that require pieces to move longer distances across the board, leading to poor play. In addition, the use of minimax assumes a human player would not make any mistakes and leads to very conservative gameplay; a more aggressive approach may yield more wins in many cases. Obviously, this would come at the risk of more losses.

The training approach presented by Chellipilla and Fogel became known as *Blondie24*, the screen name they used on the internet gaming website, in its final version [52]. Blondie24 was rated at an expert level, capable of beating most human players, including a master player at times.

2.6.2 Tic-Tac-Toe Competitive Learning with PSO

Messerschmidt and Engelbrecht presented a competitive coevolutionary PSO algorithm to train Tic-Tac-Toe agents [106]. Their approach made use of neural networks as game state evaluators. Traditional game trees are constructed up to a specified ply-depth, at which point the neural network assigns a desirability factor to each of the leaf nodes. An agent then uses a standard minimax algorithm to determine the next move.

Training of the neural networks is done using a competitive coevolutionary PSO. Each particle represents a candidate evaluation function implemented as a neural network. The components of each particle correspond to the neural network weights and biases. Sigmoid activation functions are used in each of the neurons.

Fitness is calculated by playing and accumulating the resulting game scores against a sample selected from a competition pool built from the current particles as well as their personal best positions.

In order to benchmark the performance of the trained agents a complete game tree for Tic-Tac-Toe was constructed and the leaf nodes were classified by outcome. The probability of reaching each of the leaf nodes was calculated (table 2.1 summarises the findings).

Table 2.1: Outcome probabilities for Tic-Tac-Toe.

Outcome	**Probability**
Draw	12.7%
Win as player 1	58.5%
Win as player 2	28.8%

The performance of trained players can be measured against the outcome probabilities by playing 10000 games as first player, and another 10000 games as second player against randomly behaving agents using the following equation:

$$M = (w_1 - 0.58) + (w_2 - 0.28) \tag{2.21}$$

where w_1 is the percentage of games won as player 1 and w_2 is the percentage of games won as player 2. If the value for M is above 0, the trained player performed better than a random player. Higher values for M indicate better and more desirable players.

Experimental results indicated that the PSO-based training approach produced agents with better performance than the ones generated by evolutionary algorithms, similar to that of Chellipilla and Fogel [22, 52]. Depending on the number of hidden nodes in the neural network, the M value ranged from around 5 to almost 19 for the local best PSO. Using more hidden nodes produced higher performance ratings. Messerschmidt and Engelbrecht also found that the local best guaranteed convergence PSO (GCPSO) performed best in all of their experimental work.

The algorithm presented by Messerschmidt and Engelbrecht was successfully extended and applied to the game of Checkers [108]. Experimental results indicated that the PSO-based training method performed on par with the EP approach presented by Chellipilla and Fogel. Both approaches managed to train agents successfully to play Checkers. Franken and Engelbrecht also successfully applied the PSO-based training method on the game of Checkers [58].

2.6.3 PSO Approaches to Co-evolve IPD Strategies

Franken and Engelbrecht presented two PSO-based algorithms to train agents for the Iterated Prisoner's Dilemma (IPD) [60]. IPD is a non-zero-sum, non-cooperative game: players can receive a reward for not winning the game, as opposed to zero-sum games, such as chess, where a player either wins or loses. IPD is easily explained by means of an example: Two suspected criminals are arrested and interrogated separately by the police. Each criminal now has a choice to either keep to their story - thus cooperating with the other criminal - or make a deal with the police and implicate the other criminal - thus defecting. Agents are rewarded based on a reward matrix constructed from the response combinations. Franken and Engelbrecht made use of the reward structure that was introduced by Axelrod [4]. The four possible outcomes are rewarded as follows:

- **Both agents cooperate**: Both agents are awarded 3 points.
- **Both agents defect**: Both agents are awarded 1 point.
- **Agent 1 defects, agent 2 cooperates**: Agent 1 is awarded 5 points, agent 2 is awarded no points.
- **Agent 1 cooperates, agent 2 defects**: Agent 1 is awarded no points, agent 2 is awarded 5 points.

The game is played for a preset number of iterations, after which the total number of points for each agent is calculated. Franken and Engelbrecht used 151 iterations for their experiments.

Both training strategies made use of the same agent architecture. As historic moves play an important part of the strategy, each agent received the last three moves of both players as inputs. A 64-bit string was built where each bit represents the choice of cooperate or defect for each of the $2^{2\times 3}$ possible history states.

The first training strategy made use of a neural network to construct the agents. The neural networks were trained using a competitive coevolutionary PSO. Each particle represents a candidate evaluation function, implemented as a neural network. The components of each particle correspond to the neural network weights and biases. Sigmoid activation functions were used in each of the neurons. The six historic moves were

represented as binary inputs to the neural network. The agent is constructed by feeding each possible historic state through the neural network and recording the output into the bit string. The second training strategy used a binary PSO to adjust the agent bit string directly. Different PSO neighbourhood structures and swarm sizes were tested. Both strategies trained for 500 iterations, after which the best particle was selected for a benchmark against six well-known playing strategies.

Results indicated that both training strategies yielded agents that performed better than the well-known playing strategies. The neural network training strategies was much less tolerant to noise than the binary PSO training strategy. In addition the neural network strategy demonstrated that it was susceptible to an *always defect* strategy, causing a severe loss in performance before steadily recovering after several iterations. The binary PSO method was much more likely to result in highly cooperative agents.

2.6.4 Training Bao Agents using a Coevolutionary PSO

Conradie and Engelbrecht [29] extended the work done by Messerschmidt and Engelbrecht [106], and Franken and Engelbrecht [58, 59, 60], to the game of Bao. Bao is an African board game of the Mancala family of games. The state space of Bao is approximately 10^{25}, making it impractical to construct a complete game tree [35]. Bao adds a number of unique challenges that are not present in other games, namely:

- Bao is played in two phases, each with its own set of rules.
- Bao is not a pure turn-based game. Situations exist where a player can execute multiple moves until a condition is met.
- It is theoritically possible for endless moves to exist.
- No piece is ever removed from the board. This makes it nearly impossible to construct an endgame database using retrograde analysis [35].

A competitive PSO-based training algorithm is used to train neural networks as board state evaluators. Each particle represents a candidate evaluation function, implemented as a neural network. The components of each particle correspond to neural network weights. Sigmoid activation functions are used in each of the neurons.

Alpha-beta game trees are calculated up to a specified ply-depth, at which point the neural network assigns a desirability factor to each of the leaf nodes. An agent then uses the alpha-beta game tree to determine the best next move. Fitness is calculated by playing, and accumulating the score against a sample selected from a competition pool built from the current particles as well as their personal best positions.

Conradie and Engelbrecht made use of the global best PSO and found that particles reached a stable point relatively early on in the training process. Once a particle in the global best PSO reach a stable point, no further searching of the search space is conducted by that particle [27, 153]. To prevent this problem three models were investigated:

- **Model A**: During training each particle must compete with five randomly initialised opponents.
- **Model B**: A hand-crafted expert player is added to the competition pool from the start of the training process.
- **Model C**: A hand-crafted expert player is added to the competition pool from iteration 250 of the training process.

Conradie and Engelbrecht found that model B performed overall worst for all tested ply-depths. Model C performed best against random and expert players. The early injection of expert knowledge in model B degraded the trained agents' performance; model C corrects this problem by simply delaying the injection of expert knowledge. Model A performed equally well to models B and C against random opponents, but faired worst against expert players. Overall, model C produced the best Bao agents.

2.6.5 Evolving Neural Network Controllers for Self-organising Robots

Fehervari and Elmenreich presented an approach using an evolutionary algorithm and neural networks to evolve self-organising robots [50]. The sample problem was to evolve robot soccer players for a simulated soccer game similar to the official simulated RoboCup league [97, 129]. In contrast to the work presented in this study the training was not from zero-knowledge.

An evolutionary algorithm was used to evolve a neural network to control the robot players. Training was done for 500 generations with a population size of 60. Fehervari and Elmenreich designed a fitness function for the competitive training process that rewards the agents based on a number of different criteria, including field distribution, distance to the ball, number of kicks, ball distance to opposition goal, number of goals scored. A full analysis of their fitness function is deferred to section 5.2.1.

Comparing the performance of each individual in the popuplation with each other individual proved to be infeasible. To counter this problem Fehervari and Elmenreich made use of a Swiss system style tournament ranking. The number of games required each generation to build a ranking is $\frac{n}{2}(log_2 n)$ where n is population size, effectively reducing the number of games for a population of size 50 from 1225 to 150. Using the ranking system resulted in much information being lost, since only the best winner and worse loser positions are effectively determined. Surviving individuals were selected using roulette wheel selection with a selection probability proportional to the fitness - as determined by the Swiss system.

Different neural network architectures were compared. Specifically, different hidden layer sizes were compared, as were a fully connected network with a three-layer feed forward network. Experimental results showed that the fully connected network generally performed better than the lesser connected feed forward network. However, with more hidden nodes, the feed forward network matched the performance of the fully connected network. The fully connected network was in effect a recurrent neural network with a memory ability - this could explain the higher performance figures.

Different neural network input and output (I/O) strategies were compared. It was found that a cartesian coordinate-based I/O strategy produced better performing agents when compared to agents using a polar coordinate-based I/O strategy. The work done in this study uses the same neural network input architecture as the one used by Fehervari and Elmenreich.

Fehervari and Elmenreich visually analysed a number of games. A number of strategies that emerged were observed:

- The player closest to the ball moved to the ball.
- Players from the same team close to the ball stayed close, but did not converge

onto the same spot.

- The player with the ball kicks it towards the opposition goal.
- Players far from the ball spread out, building a defensive mesh in front of their own goal.
- Players sometimes stick to specific opponent players (also known as man-marking).

Given the simplified nature of the neural network, the strategies that emerged seemed quite impressive.

2.6.6 Evolving Multi-Agents using a Self-organising Genetic Algorithm

Jeong and Lee proposed using the self-organising genetic algorithm to evolve soccer agents [77]. A simplified soccer game was defined to test the performance of the training algorithm. Each team consists of two players. The game is played on a field that is 5 wide by 6 long. Additional goal blocks are added to each side. The simple soccer model used throughout this study is based on this simplified soccer model. Details of the simple soccer model are provided in section 3.2.1, where modifications to the model as used in this study are also discussed. The training algorithm presented in chapter 4 performs much better than the algorithm discussed here (refer to section 7.8 where results are provided).

Jeong and Lee made use of a rule table to control the agents. Each rule has the following format: C(...) L(...) R(...) T(...) B(...) $\rightarrow$ *action*. Each rule performs an action based on the object that is center, left, right, top, and bottom, of the agent. Three actions are allowed: move, dribble, and kick. Each action is coupled with an associated execution direction (left, right, top, or bottom). Individuals in the population of the genetic algorithm represented the complete rule table using a bit string of length 3300. All possible inputs and the associated outputs are represented in the bit string.

A population size of 50 chromosomes was used, the remainder of the genetic algorithm parameters being adjusted according to the self-organising genetic algorithm. Jeong and Lee developed a fitness function capable of measuring each agent's performance. An

in-depth discussion of the fitness function is deferred to section 5.2.1, where the possibly biased behaviour introduced by the fitness function is discussed.

Jeong and Lee measured the performance of the evolved agents in three situations:

- **Case 1**: Both opponent players are stationary and simply act as obstacles.
- **Case 2**: The first opponent player is stationary in front of the goal post. The second opponent player moves to capture the ball - thus interfering with the dribble and kick actions of the agents.
- **Case 3**: The ball is positioned to the left of the field. The same opposition team behaviour as in case 2 is reused.

The proposed approach was able to reach high fitness levels within 20 generations for case 1 and 10 generations for case 2. For case 3 it took around 120 generations before a solution was found. Games lasted an average of 10 moves per agent for cases 1 and 2, while case 3 took an average of 20 moves per agent to complete a game. The fitness oscillated especially in case 3 due to the increased difficulty of the problem.

2.7 Summary

The objective of this chapter was to provide background information on all the various computational intelligence techniques employed throughout this study. Specific focus was placed on *artificial neural networks*, *evolutionary computation*, *particle swarm optimisation*, and *coevolution*. Artificial neural networks were presented along with the various learning paradigms as used throughout this study. An overview of evolutionary computation was presented. Various forms of the particle swarm algorithm were presented and discussed, each with their own advantages and disadvantages. Coevolutionary training approaches were presented and discussed. Both competitive and cooperative coevolution were covered. Finally, existing work on particle swarm optimisers as trainers for neural networks was briefly discussed as it applies to this study. A brief discussion of the work that influenced the simulated soccer model presented in the next chapter was given.

The next chapter introduces the simple soccer benchmark problem for this study. Additional background information on why the problem was chosen as the benchmark

is also provided along with a detailed description of the benchmark problem itself and its rules. The above-mentioned techniques are combined in chapter 4 to create a new training algorithm that will be evaluated on the simple soccer problem.

Chapter 3

Simulated Soccer

> *"By the mid-21st century, a team of autonomous humanoid robots shall beat a human World Cup champion team under the official regulations of FIFA."*
>
> RoboCup's objective

The objective of this chapter is to provide background on the soccer-training problem used in this thesis and to present the details of the simulated soccer model used throughout this work. The computational intelligence techniques presented in the previous chapter are used to train agents in playing soccer intelligently. The Simple Soccer model presented in this chapter forms the basis on which this training is done. Section 3.1 provides background on Robot Soccer and the research and education initiatives driven by the RoboCup [129] and FIRA [51]. The choice of robot soccer as the game of choice for artificial intelligence and robotics research is also discussed. Section 3.2.1 is dedicated to providing background and details around the Simple Soccer model as used throughout the remainder of this study. The properties of the Simple Soccer model are investigated and it is shown that the Simple Soccer model represents a non-trivial, non-deterministic problem, ideally suited to testing the training algorithm presented in this study.

3.1 Robot Soccer

The 21st century saw the proliferation of massively parallel machines. In 1993, Kitano proposed that a Grand Challenge AI Application should be identified and pursued [91].

In 1997, Kitano *et al.* named RoboCup as a challenge for AI research [92]. Another view on robot soccer research is that it presents a standardized problem for theories, algorithms, and architectures to be tested on. Computer chess previously provided such a problem for research. However, with the accomplishment of IBM's Deep Blue beating the then world chess grandmaster Gary Kasparov, the challenge of chess is close to its finale. A new goal is needed that will initiate a next generation technology. Table 3.1 presents a comparison between the classic chess problem and robot soccer. The table makes the increase in complexity evident when moving from the chess problem to robot soccer.

Table 3.1: Comparison between robot soccer and chess.

	Chess	**Robot Soccer**
Environment	Static	Dynamic
State Change	Turn taking	Realtime
Info accessibility	Complete	Incomplete
Sensor Readings	Symbolic	Non-symbolic
Control	Central	Distributed

Robot soccer presents research challenges in broad areas of AI and robotics, including real-time sensors, reactive behaviours, strategy acquisition, learning, real-time planning, multi-agent systems, context recognition, computer vision, strategic decision making, motor control, and intelligent robot control [92].

The next two sections provide background on RoboCup and FIRA, the two most popular robot soccer initiatives.

3.1.1 RoboCup

"By the mid-21st century, a team of autonomous humanoid robots shall beat a human World Cup champion team under the official regulations of FIFA." These words sum up

the objective of an international research and education initiative called RoboCup (previously known as "The Robot World Cup Soccer Games and Conferences" and "Robot World Cup Initiative") [129]. The success of the RoboCup project would be a major achievement for the field of computational intelligence [3]. Yet, achieving the goal of winning a human championship team remains an elusive goal for the contestants that compete each year for the various RoboCup titles.

RoboCup is an attempt to advance the study into intelligent robotics and AI by providing a standardized problem space (the game of Soccer), for which a wide range of technologies needs to be examined, investigated and integrated. Soccer competitions form only a part of the larger RoboCup initiative, which includes technical conferences, RoboCup challenge programs, education programs and infrastructure development in addition to the RoboCup international competition and conferences that form the central pillar of their activity.

RoboCup Soccer is divided into five leagues:

- **Humanoid**: The humanoid league deals with autonomous robots that resemble humans in body build and senses. The challenges introduced in this league includes dynamic walking, running, and kicking a ball. Concurrently with these actions, the robot needs to maintain balance as well as visual perception of the ball and other players. Self-localization and team play are also researched as part of this league. The physical size of the robots subdivides this league further into three subleagues: kid size, teen size, and adult size.

- **Middle size**: The middle size league deals with robots that have a diameter of no more than 500mm. Teams of up to six robots play with a regular size FIFA soccer ball on a scaled-down version of a real soccer field. The robots carry all their sensors and are allowed to communicate using wireless networking technology. The research focus of this league is autonomy and cooperation at plan and perception levels.

- **Simulation**: The simulation league deals with virtual robots that play on a simulated field. The research focus of this league is artificial intelligence and team strategy. The league is further subdivided into 2D and 3D subleagues.

Within the 2D league, two teams, each with eleven players, play a virtual soccer game on a software platform called *SoccerServer* [113]. The server facilitates communication with each player, called an agent. Agents receive inputs for their virtual sensors from the server. The server also introduces noise into the sensor readings. Each agent can perform a number of basic commands including dashing, kicking, and turning. A game consists of 6000 cycles; each agent has 100ms per cycle to execute an action.

The 3D league introduces the complexity of having to deal with an extra dimension as well as more complex physics than those of the 2D league. The league started off with a simply spherical shaped agent and was extended in 2006 to include a model of the *Fujitsu HOAP-2* robot [97]. With the introduction of a model humanoid robot, the focus shifted away from gameplay strategy towards the low-level control of basic behaviours such as walking, kicking, turning, and standing up. In 2008 a model of the *Nao* robot was added [1]. The real Nao robot was used as the official robot for the standard platform league [128]. The introduction of a simulated version of the robot allowed researchers to test algorithms and ideas in a simulated environment before deploying them on their physical robots. With much progress being made in low level robot control, the focus is slowly starting to shift back towards high-level playing strategies built on tested low-level control strategies.

- **Small size**: The small size league, or F180 league, deals with robots with a diameter no larger than 180mm and no taller than 150mm. Games are played on a green carpeted field that is 6.05m long by 4.05m wide with an orange ball. Two cameras mounted above the field track the objects on the field using an open-source software system called *SSL-Vision*. Off-field computers send position information and referee commands to the robots using wireless technology. The league focusses on multi-agent cooperation and control in a hybrid centralized and decentralized system with a highly dynamic playing environment.

- **Standard platform**: The standard platform league provides identical robots to all the contestants. The league was previously known as the four-legged league while using the *Sony Aibo* [20] robot from 1999 up to 2008 . In 2008 the *Nao* [1] robot

built by Aldebaran Robotics became the new standard platform robot, replacing the aging and discontinued Aibo [128]. The robots operate fully autonomously - there is no communication or external control. The robot controlling algorithms must trade vision resources between self-localization and ball localization as omni-directional vision is not available. Owing to the level playing field provided by using a standardized robot, research focus is placed on the software development part of robot control and coordination.

Each year the rules that govern the games within each of these leagues are adapted and updated by a committee to keep the complexity of the games relevant for the level of technology and techniques that exist. This allows for a gradual increase in complexity as the different competing teams learn to cope with and manage the unique challenges that each league presents.

RoboCup is designed to create a challenge of maintaining an affordable problem size and research cost while still handling the need for real-world problem complexities.

3.1.2 FIRA

The Federation of International Robot-soccer Association (FIRA) [51], similar to RoboCup, is an international research and education organisation. Founded in 1997, FIRA grew out of the Micro-Robot World Cup Soccer Tournament (MiroSot). The MiroSot annual event was initiated in 1995 by Prof Jong-Hwan Kim [79] of the Korea Advanced Institute of Science and Technology (KAIST) [80].

FIRA's objective is to help the young minds of today to better understand and appreciate the scientific concepts and technological developments involved when working with robotics. FIRA's goal is that some of this interest will drive further advancement of scientific and engineering skills that would ultimately serve mankind in a variety of ways. Working towards achieving this goal, FIRA runs a number of initiatives. The FIRA congress, being one such initiative, is run along side the FIRA Cup and allows participants to share their expertise and technological developments for building soccer robots. The international robot olympiad is another initiative with the goal of reaching promising youngsters before university age, to which most of the other initiatives are limited to. FIRA summarizes their objectives as follows [51]:

- To take the spirit of science and technology to the young generation and laymen.
- To promote the development of autonomous multi-agent robotic systems that can cooperate with each other and to contribute to the state-of-the-art technology improvement in this specialized field.
- To bring together skilled researchers and students from different backgrounds such as robotics, sensor fusion, intelligent control, communication, image processing, mechatronics, computer technology, artificial life, etc. into a new and growing interdisciplinary field of intelligent autonomous soccer-robots to play the game of soccer.
- To organize the FIRA Robot World Cup and Congress every year.
- To work together to establish the FIRA Robot World Cup as a Science and Technology World Cup.

Since its inception, FIRA's flagship event has become the annual FIRA Robot World Cup, also called the FIRA Cup. The FIRA Cup is divided into a number of categories in which tournaments are played. Each category has its own rules that dictate the physical robot specifications as well as the environmental conditions. The current tournament categories for the FIRA Cup are:

- **MiroSoT**: The micro-robot soccer tournament deals with robots up to 75mm per dimension in a cubical shape. The soccer match is played between two teams, each with three robots. Each team is allowed one computer primarily dedicated to vision processing and position identification. The robots communicate using wireless technology. An orange golf ball is used. The soccer field size can be either 4m long by 2.8m wide for the large league or 2.2m long by 1.8m wide for the middle league. The robots need to operate fully autonomously, cooperating and coordinating with each other to win the match. The research focus of the MiroSoT is devising good strategies using AI techniques, and the development of sharp sensing and precise real-time control for the robot players.

- **SimuroSoT**: The simulated robot soccer tournament deals with virtual robots that play on a simulated field [89]. The SimuroSoT can be viewed as the MiroSoT without all the hardware elements. No robot or vision hardware is required, eliminating the required work to perform sensing and executing actions. Two larger leagues also exist that allow for five versus five and eleven versus eleven matches to be played. The research focus of the SimuroSoT is complex strategy development using AI techniques.

- **HuroSoT**: The humanoid robot soccer tournament deals with biped-robots up to 1.5m in hight and weighing up to 30kg. The soccer field ranges in size from 3.4m to 4.3m long by 2.5m to 3.5m wide. Given the current state of technology, robots in this category do play a game of soccer. Instead, participants are rated on more basic tests, for example, the ability to walk steady, obstacle avoidance, and taking penalty shots. A human trainer provides guidance to the robot.

- **AmireSoT**: The AmireSoT game deals with miniature fully autonomous robots up to 110mm in size. Robots play in a team of one, two, or three [166]. The robots are equipped with infrared and colour image sensors for sight. A yellow tennis ball is used on a soccer field 1.3m long by 0.9m wide.

- **NaroSoT**: The nano-robot soccer tournament deals with robots of up to 40mm squared by 55mm in hight. The extremely small robots play on a field of 1.3m long by 0.9m wide with an orange ping-pong ball. Five robots with one optionally being a goalkeeper form a team. Each team is allowed one computer primarily dedicated to vision processing and position identification. The robots communicate using wireless technology.

- **AndroSoT**: The android soccer tournament deals with biped androids of up to 600mm in height. The field is 2.2m long by 1.8m wide. An orange golf ball is used. Each team is allowed one computer primarily dedicated to vision processing and position identification. The robots communicate using wireless technology and are controlled by the off-field computer, with the exception that the goalkeeper may be fully autonomous. Teams consist of at most three robots.

- **RoboSoT**: The robot soccer tournament deals with robots of up to 350mm square with no height limit [89]. Two teams of up to three robots each, compete. This category differs from MiroSot and NaroSot in that the overhead camera is dropped in favour of vision processing systems onboard each robot. The robots can be fully autonomous or semi-autonomous; in the semi-autonomous case vision processing is done by an off-field host computer. Owing to the unique challenges presented by this category, a number of challenges are presented to contestants, namely:
  - **Vision challenge**: A robot must exhibit capabilities in at least one of the following areas: identification, localisation, or classification. The goal is the creation of an effective and efficient vision system.
  - **Co-operative challenge**: A robot must exhibit capabilities in communication or co-operative task-related skills that are essential to team functionality.
  - **Motion challenge**: A robot must exhibit capabilities in the areas of motion control, ball handling, or path planning.
  - **Open creative challenge**: Participation in this challenge is optional. Teams are allowed to demonstrate the unique abilities of their robot in application areas not necessarily directly related to robot soccer but rather related to robotics in general. The best presentation is declared the winner of this challenge.
  - **Robot soccer competition**: Play the game of soccer on a field 6m long by 4m wide according to the official rules of RoboSoT.

Each category targets a different area for future research and development [89]. The rules, as with RoboCup, evolve with the progress being made by the competing teams to keep the game's research objectives relevant for the current technology level.

3.2 Simulated Robot Soccer

The previous section showed that both the RoboCup and the FIRA Cup made a software platform available for simulated robot soccer. Computational complexities and time

restrictions made it impractical to use the official RoboCup or SimuroSoT simulators for training the game agents in the context of this study. The work conducted in this study require thousands of games to be played. A simpler and more efficient simulation model was needed. An overview of available models previously used for training genetic algorithms in the soccer problem domain is provided by Plant *et al.* [123].

Owing to its simplicity, the Simple Soccer model introduced by Jeong and Lee is the best suited to the goals of this research [77]. Simple soccer is loosely based on the effector automata introduced by Lohn and Reggiat [102]. The low computational requirements of Simple Soccer make it possible to conduct thousands of experiments within an acceptable timeframe. The next section provides detail on the Simple Soccer simulation model as well as the modifications made specifically for this study.

3.2.1 Simple Soccer

The following subsections provide detail on the Simple Soccer field, team composition, agent sensors, agent actions, and overall game rules.

Soccer field

Games are played on a two dimensional virtual soccer field of size $l \times w$. Figure 3.1 depicts the soccer field as it is used in this study. The size of the field is 5 wide by 6 long. The sizing of the field is similar to the field size used by Jeong and Lee [77]. Two additional blocks that represent the goal areas are added to the field, one on each side.

Team composition

Two teams compete against each other. Each team consists of two agents. Agents A_1 and A_2 form team A while agents B_1 and B_2 form team B. In the original Simple Soccer model agents had identical behaviour. In this study each agent is allowed to exhibit its own unique behaviour.

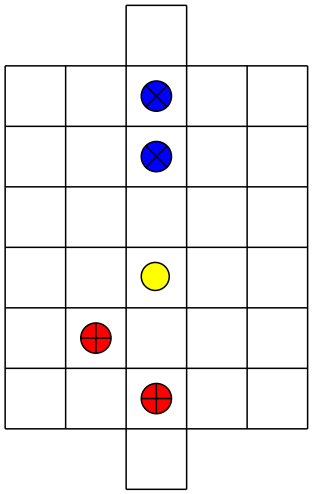


Figure 3.1: 5×6 Simple Soccer field with the ball and players.

Agent sensors

Agents in the original Simple Soccer model received inputs from the surrounding blocks. Figure 3.2 depicts the blocks that could provide inputs to an agent. The surrounding blocks are divided into four categories: O, I, II and III. O represents the block where an agent is located; the other blocks are relative to this position. The different block categories determine what can be sensed by the agent:

- **O**: Opposition team members, team members, and goal posts can be sensed.
- **I**: Opposition team members, team members, and goal posts can be sensed.
- **II**: Team members and goal posts can be sensed.
- **III**: Only goal posts can be sensed.

In addition to the above sensory inputs, an agent always receives a general ball direction input. The ball's position is indicated as either left or right, and forward or backwards, from the players' current position. Each of the input types per block can easily be mapped to an input unit for a neuro-controller.

Fehervari and Elmenreich demonstrated the importance of well-designed sensors to serve as neural network inputs when attempting to train neuro-controllers for a soccer-like environment [50]. Fehervari and Elmenreich proposed the use of four neurons to

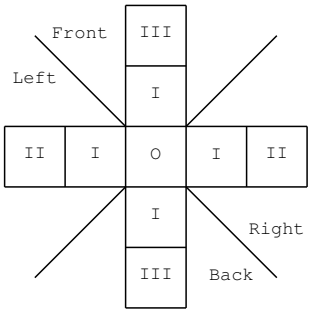


Figure 3.2: The original Simple Soccer agent sensors.

detect each class of object. Classes of objects are the ball, nearest team-mate, nearest opponent, and wall. The four neurons represent the distance to the object in a specific direction, i.e. north, south, east or west. The input functions for the four neurons are defined as follows:

$$f(north) = \begin{cases} \frac{\Delta y}{d^2} & \text{if } y > 0 \\ 0 & \text{if } y \leq 0 \end{cases} \tag{3.1}$$

$$f(south) = \begin{cases} \frac{\Delta y}{d^2} & \text{if } y < 0 \\ 0 & \text{if } y \geq 0 \end{cases} \tag{3.2}$$

$$f(west) = \begin{cases} \frac{\Delta x}{d^2} & \text{if } x < 0 \\ 0 & \text{if } x \geq 0 \end{cases} \tag{3.3}$$

$$f(east) = \begin{cases} \frac{\Delta x}{d^2} & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{3.4}$$

where Δx is the distance along the east-west direction and Δy is the distance along the north-south direction; d is defined as $d = \sqrt{\Delta x^2 + \Delta y^2}$.

The proposed four-neuron input strategy can be applied directly to the simple soccer model. Empirical testing revealed that this arrangement of sensors resulted in much better results. This is the reason why this study uses this input architecture rather than the one presented by Jeong and Lee. A complete analysis of different input architectures is beyond the scope of this study and is left as a possible future research topic.

Agent actions

Each agent can perform one of several actions per turn:

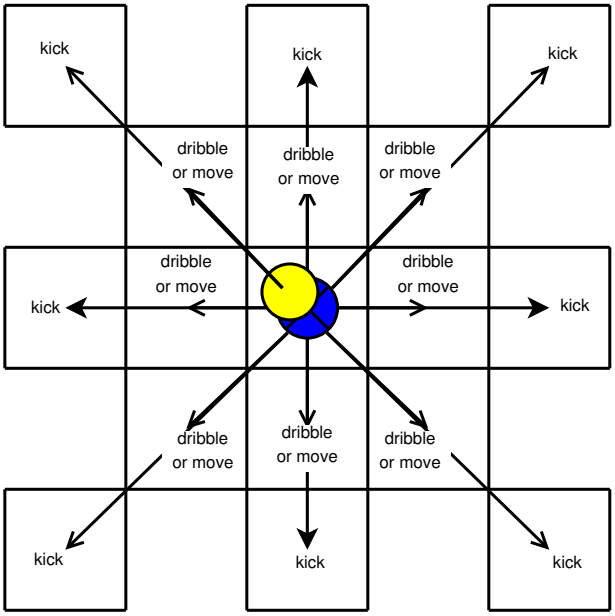


Figure 3.3: Simple soccer agent actions.

- **Move**: An agent can move one square to either of the eight adjacent squares.
- **Dribble**: An agent that has possession of the ball can move one square to either of the eight adjacent squares, taking the ball along with it.
- **Kick**: An agent that has possession of the ball can kick the ball two squares away in either the left or right, or forward or backward directions.

Figure 3.3 depicts the actions that an agent can perform, along with their directions. In the event that an agent executes an invalid action, for instance kicking without having possession of the ball, that action is simply ignored.

Rules

The rules that govern how each game is played are:

- For each team one agent starts the game in front of the goal area, the second agent starts in either of the three blocks in front of the first agent. The exact placement of the second agent is determined using a uniform random placement function.

- The ball starts in the middle of the field. If the middle is between two or four blocks, the placement is done uniform randomly within either of the blocks.
- The game stops once a goal is scored, or if a specified maximum number of iterations is reached.
- In the event that multiple agents occupy the same block as the ball, the ball ownership is determined using the probability $\frac{1}{n}$, where n is the number of agents occupying the block.
- A goal score is awarded once the ball enters the goal area on either side.
- If an agent is in possession of the ball and the move action is executed, the agent will automatically perform the dribble action.
- An agent cannot leave the field with either a move or dribble action.
- The ball cannot be kicked off the field. If a kick is performed that would result in the ball leaving the field, the ball is simply stopped at the field's boundary.

Jeong and Lee presented a fitness function to measure the performance of agents using the Simple Soccer model. The details and validity of this fitness function is deferred to section 5.2.1 of this study [77]. The next section evaluates the characteristics that the Simple Soccer model exhibits.

3.2.2 Simple Soccer Characteristics

The Simple Soccer model as described in the previous section is used for all the experimental work conducted in this study. The low computational requirements of games using the Simple Soccer model allows for thousands of simulations to be conducted in a relatively small space of time.

Each game agent in the Simple Soccer model poses only partial knowledge of its environment. The agent sensory inputs limit the perceived environment to a subset of the total knowledge of the game state. Game agents in different locations on the field may be able to sense different opposition players and any decision an agent makes will

be based on the limited information revealed to it by its sensors. An element of chance is introduced both through the randomised starting position of the forward game agent in each team and the randomised ball ownership function that takes place in the event the two game agents occupy the same position on the board along with the ball. The partial knowledge and chance elements of the game increase in complexity enough that this should not be considered a *toy problem*. The later chapters of this thesis which evaluate agent training performance will reinforce this notion.

Table 3.2 presents a comparison between the RoboCup soccer and Simple Soccer as used in this study. The concurrent turn-taking model used by Simple Soccer can be viewed as a computationally less expensive modelling of the realtime model used by RoboCup without instrumentally reducing the complexity of the problem. The dynamic environment, incomplete access to information, non-symbolic sensor readings, and distributed control all serve to increase the problem complexity enough for Simple Soccer to be an ideal problem to test training on.

Table 3.2: Comparison between the RoboCup and Simple Soccer.

	Simple soccer	**RoboCup**
Environment	Dynamic	Dynamic
State Change	Concurrent turn taking	Realtime
Info accessibility	Incomplete	Incomplete
Sensor Readings	Non-symbolic	Non-symbolic
Control	Distributed	Distributed

The Simple Soccer model was implemented using the CIlib framework [117, 118, 120]. CIlib, also known as the Computational Intelligence Library, is an open-source reusable generic framework for computational intelligence experimentation developed by the Computational Intelligence Research Group at the Department of Computer Science, University of Pretoria, South Africa, and can be downloaded from *http://www.cilib.net*.

3.3 Summary

The first objective of this chapter was to provide background information on the soccer training problem. RoboCup and FIRA were discussed to provide background on soccer and robotics in research and education. The importance of the soccer problem space and its choice for this study were clearly conveyed to the reader.

The second objective of this chapter was to introduce the simulated soccer model used through the remainder of this study. The Simple Soccer model was clearly described in the second part of this chapter. The rules governing play were also presented. Enhancements made to the original Simple Soccer model were presented. Finally, the attributes and properties of the Simple Soccer model were discussed, showing that it is a non-trivial, non-deterministic problem.

The next chapter introduces the cooperative competitive coevolutionary charged particle swarm optimisation training algorithm used to train the game agents to play the Simple Soccer game as presented in this chapter. Early results from the training are also presented and discussed in the next chapter.

Chapter 4

Cooperative Competitive Coevolution with Charged PSO

"All men can see these tactics whereby I conquer, but what none can see is the strategy out of which victory is evolved."

Sun Tzu (500 BC)

The second chapter presented existing computational intelligence techniques as well as their typical applications. The objective of this chapter is to present and analyse a new training algorithm, based on these computational intelligence techniques, capable of training multi-agent teams from zero knowledge. The simple soccer model presented in the previous chapter serves as a benchmark game implementation on which this new training algorithm is tested and evaluated. This chapter starts by covering competitive training, extending the approach into a new multi-population training algorithm. Benchmarking of the new training algorithm is briefly discussed, followed by parameter optimisation of the new training algorithm. Analysis is carried out on preliminary results to quantify the performance of the first prototype of the new training algorithm.

4.1 Introduction

An existing particle swarm optimisation-based coevolutionary training algorithm is presented in section 4.2. This algorithm forms the foundation on which the new multi-

population coevolutionary training algorithm, as presented in section 4.3, is based. The algorithm as well as relevant parameters, including neural network architectures and particle swarm optimisation parameters, are discussed in detail.

Section 4.4 covers the benchmarking techniques available to measure the algorithm's performance as a discrete measurement. The benchmarking measurement is used to perform parameter optimisation as described in section 4.5.

The early results presented in section 4.5 highlight a problem: during simulation, a number of the trained players had extremely low performance measurements. This resulted in high standard deviations in the reported performance. These outliers were analysed and alternative reporting measures such as median and trimmed mean were presented.

4.2 Competitive Training

Messerschmidt and Engelbrecht introduced a PSO-based approach to training neural networks in a coevolutionary environment [106, 107], based on work done by Chellapilla and Fogel [21, 22, 23] and Fogel [52]. Messerschmidt and Engelbrecht's approach uses a PSO to train a neural network as a game tree state evaluator, outputting a scalar value that indicates how desirable a game state is (refer to section 2.6.2 for more detail). This neural network game state evaluator can then be combined with existing game tree algorithms such as minimax or alpha-beta pruning. The training is done in a coevolutionary fashion, because the only information available is the game rules and the game outcome (win, loss, or draw). The lack of knowledge necessitates a coevolutionary training mechanism. The relative fitness function in this case is simply a score based on the outcome of a number of games played between the neural networks in the population being trained. Algorithm listing 2.2 summarised the approach presented by Engelbrecht and Messerschmidt. Note that their approach requires only a single population of neural networks, and that a player can play in both the first and second position of the game.

The next section presents a variation on the approach presented by Messerschmidt and Engelbrecht to accommodate the training of a whole team of game agents where each team and position combination is treated as a separate population. The game agents in

this study were trained using the simple soccer model presented in section 3.2.1.

4.3 Multi-population Competitive Training

Unlike most board games, many realtime games exist where there is no concept of a single first or second player; a team of players needs to specialise in different skill sets in order to win the game, often requiring cooperation between team members in order to win. The classic *Predator-Prey* game can be seen as an example of this, as the predator requires a different set of skills than the prey in order to win the game. Conversely, the prey requires a different set of skills than the predator in order to win the game.

The specialisation of player skill sets can be seen as different optimisation goals. Each optimisation goal can be represented using a different population trained by a PSO. These different populations compete against each other, in a coevolutionary fashion, to better themselves against their opponents. The concept of team members can easily be introduced into this model; the only alteration needed is that the relative fitness function now rewards players within a team the same, thus encouraging cooperation between players in a team.

This section presents and discusses in detail the multi-population competitive training algorithm used throughout this study. Section 4.3.1 presents the high level training algorithm. Section 4.3.2 discusses the neural network architecture used throughout this study. Finally, section 4.3.3 discusses how the PSO algorithm is used to train the neural networks.

4.3.1 Algorithm

Simple soccer, as presented in section 3.2.1, requires four players to play - two teams, each with two players. The four players each represent a different goal to be optimised. The goal is to find a good player for each position. Each goal requires a separate population. Let $O_s(t)$ represent the population (or particle swarm) at timestep t for player position s, where $s = \{1, 2, 3, 4\}$.

The relative fitness of each particle, $P_i \in O_s(t)$, is calculated by playing games against randomly selected team-members and opponents. Each game provides information on

the performance of the whole team. In order to extract the performance of an individual player, a number of games are played using different team compositions. The only constant player in each game is the player (or game agent) representing the particle P_i being evaluated. Algorithm 4.1 summarises the steps for training game agents in a team-based problem environment. The experimental work in this study follows this approach.

Create and initialise a swarm of neural networks for each game position.
repeat:
  for each *swarm* $O(t)$ **do**
    for all *swarms* $O_s(t) \neq O(t)$ **do**
      Add each personal best position to the competition pool for swarm $O_s(t)$.
      Add each particle to the competition pool for swarm $O_s(t)$.
    end for
    for each *particles* P_i *(or NN) in the swarm* $O(t)$ **do**
      repeat:
        Select team members and opponents from the competition pools.
        Play a game using the selected players.
        Record if game was won, lost or drawn.
      until *predefined number of games has been played.*
      Determine a score for each particle.
      Compute new personal best position based on score.
    end for
    Compute new neighbourhood best position.
    Update particle velocities.
    Update particle positions.
  end for
until *all swarms converge* **or** *iteration limit is reached.*
The Global best particle in each swarm is the trained neural network game agent for the corresponding team position.

Algorithm 4.1: Competitive coevolving team-based PSO (CCPSO) algorithm to train neural network game agents (asynchronous implementation).

Particle and Population Representation

One of the first applications of the particle swarm optimiser was to train neural networks [38]. Engelbrecht and Messerschmidt [106] used the trained neural networks as board state evaluators, outputting only a single scalar value indicating board state desirability. This desirability factor is then used in conjunction with traditional game tree algorithms to determine the next move for the game agent. The experimental work in this study does not follow this approach; instead, each particle represents a neural network game agent controller (also called a neuro-controller), directly controlling the actions of the agent.

Section 2.2 described how a neural network can be treated as a function to be optimised if the weights connecting the neurons are seen as the input vector of the function and the measured performance of the neural network, the function output value. Following this approach, each particle, P_i, represents a complete weight vector for a neural network controller. The dimension of the vector is determined by the neural network architecture.

Figure 4.1 provides a graphical representation of how a game agent playing simple soccer maps to a particle for the corresponding playing position. Using this approach, each player position has its own neural network architecture as each position has its own swarm of particles being optimised by individual PSOs.

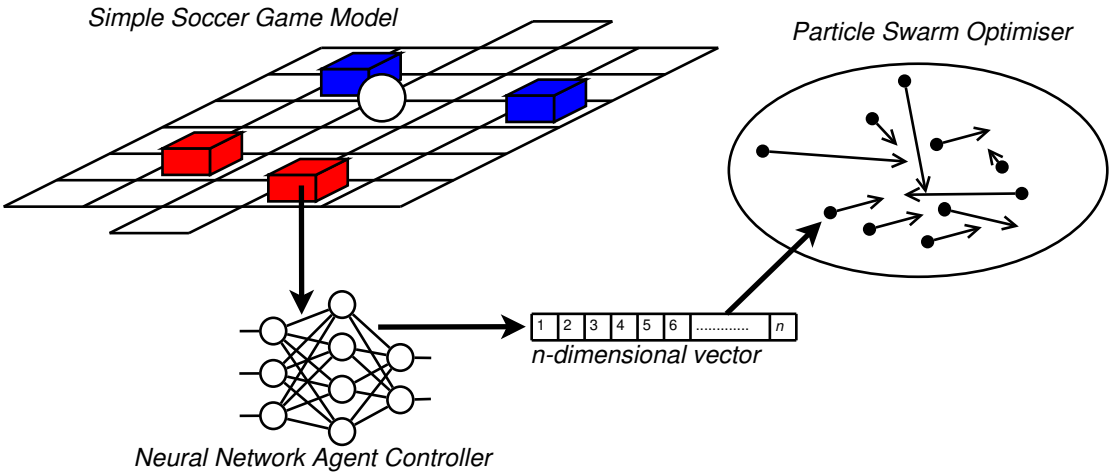


Figure 4.1: Population dynamics for Simple Soccer agents.

4.3.2 Neural Network Architecture

The experimental work in this study made use of the basic feed-forward neural network algorithm as neuro-controller for the simple soccer agents. The neural network made use of a single layer of hidden neurons with the hidden layer size left as a parameter to be optimised in section 4.5. The activation function used was the hyperbolic tangent function described in section 2.2.1.

Section 3.2.1 described in detail the sensory inputs and action outputs that comply with the simple soccer game rules. The neural network input consist of a 16-dimensional floating-point vector. The values for the input vector is calculated using equations (3.1), (3.2), (3.3), and (3.4). The input vector is constructed by combining the four equation outputs for each of the four object classes. The four object classes are the ball, the closest team member, the closest opponent, and the field boundary.

The neural network output consists of an eight-dimensional floating-point vector with values scaled between 0 and 1. The vector is constructed by combining the movement and kick vectors. The movement vector indicates the movement direction, and the kick vector indicates the direction to kick the ball. The four values per vector are desirability factors to move or kick in each of the primary directions: forward, backward, left, and right. A desirability factor larger than 0.5 indicates a desire to move or kick in the corresponding direction. If desirability is indicated in two conflicting directions (i.e. both left and right), the direction with the larger desirability factor is given precedence. If two conflicting desirability factors are equal, no desire is indicated and no action will be taken. For example, the vector $\{0.6, 0.8, 0.2, 0.4, 0.3, 0.4, 0.1, 0.2\}$ represents a desire to move backward.

4.3.3 PSO Architecture

Training neural networks using PSO in a coevolutionary fashion can be seen as optimisation within a dynamic environment. The problem space changes as the opponents (and team members) change their behaviour. Changes occur with each iteration which render the standard PSO presented by Eberhart and Kennedy unsuitable for this problem. It was previously shown that the PSO does not deal well with changes to the search space

[36]. Section 2.4.5 reviewed the various variations made to the PSO in order to compensate for changes in dynamic environments. The experimental work in this study made use of the CPSO developed by Blackwell and Bentley due to the CPSO's ability to deal with changes in the search space [10, 11, 12]. The CPSO parameters are evaluated and discussed in section 4.5.

Relative Fitness Function

Under normal, non-coevolutionary conditions the PSO algorithm requires an absolute fitness function capable of producing a scalar value that indicates the performance of the particle being evaluated. Closer inspection of the PSO algorithm reveals that the absolute fitness value is used only for comparisons between different particles in order to determine which particle is performing better than another. This property of the PSO algorithm allows for the absolute fitness function to be replaced by a relative fitness function that indicates only which particle performs better than another (refer to section 2.5.2 where relative fitness functions were discussed). The replacement with a relative fitness function is necessary in situations where no absolute fitness function exists. The simple soccer problem in this study represents such a problem where no absolute fitness function exists.

Initially, the relative fitness function developed for the simple soccer model was based on the work done by Chellapilla and Fogel [22, 23], and Fogel [52], where a constant reward or score is assigned for winning $(+1)$, drawing (0), and losing (-2). This scoring strategy penalises or punishes more for a loss than it rewards a player (or team) for a win. This should in effect lead the training to teach players to draw more often than lose in cases where winning is not possible. For simple soccer, an equal number of goals by each team is considered a draw for both teams, whereas an unequal number of goals results in one winning and one losing team.

This study also investigates the effect of a more domain-specific relative fitness function developed for the simple soccer model. This function uses the sum of the *goal difference* between the two competing teams of a number of *competitions*. Each particle (or game agent) being evaluated plays a predefined number of competitions against randomly selected competitors. The goal differences for these games are then added

together, and used as the particle's relative fitness.

Particle Re-evaluation

A side-effect of storing the relative fitness value for each particle is that the personal best position becomes stale over a number of iterations. That is, the value stored as the personal best position's fitness is no longer valid due to the changes of the search space. To counter this problem, a partical re-evaluation strategy was added to the algorithm, where all personal best positions were re-evaluated for each $c_{re-evaluate}$ iteration, where $c_{re-evaluate} \geq 1$ is a predefined constant natural number. High values for $c_{re-evaluate}$ will lead to particles remaining stale. The longer a particle remains stale, the slower the search. The slowdown is due to the swarm being draw to previously good areas of the search space which may no longer be good. Small values for $c_{re-evaluate}$ increases the computational complexity of the algorithm as more competitions need to take place with each iteration. Particles remain stale for shorter periods allowing the swarm to move to better regions in the search space faster.

Hall of Fame

In order to avoid early stagnation and promote exploitation of best found solutions, the hall of fame is also applied in this study. It was previously noted in section 2.5.2 that the hall of fame helps to combat the loss of exploration of the search space introduced by the "moving target" during competitive coevolution.

4.4 Benchmarking Player Performance

Section 4.3.1 discussed the algorithm that was used for training the game agents. Measuring the performance of the trained agents was not covered. As stated in section 2.5.2, the primary advantage of competitive coevolution is that training can take place in the absence of an absolute fitness function. Unfortunately, the lack of an absolute fitness function creates a problem for performance analysis and parameter optimisation: A player performing well against its peers might simply be exploiting a weakness in them.

This introduces a new question: How to determine whether a trained player is truly a good player.

Two benchmarking strategies are proposed in order to objectively measure the performance of a trained agent. These strategies are discussed in the following sections.

4.4.1 Random Opponent Benchmarking

The first benchmarking strategy makes use of random players to benchmark against, allowing for a generic performance measurement to be taken independently of the problem being optimised. Three approaches for benchmarking against random opponents are described here. All three approaches function on the premise of playing a large number of games against opponents that make random moves. The outcome of each game is recorded and used to calculate a performance measure for each game agent.

Messerschmidt Performance Measure

Messerschmidt and Engelbrecht [106, 107] introduced a performance measure for two player board games. Section 2.6.2 described the M measure in detail. The M measure relies on the fact that each board state can be exhaustively expanded and evaluated for the outcome probability to be calculated. For board games such as *tic-tac-toe* this can be seen as practical; however, for simple soccer it is simply not practical due to the increased complexity of the simple soccer game. The stochastic nature of the simple soccer ball ownership rule makes it unpractical to calculate all the possible outcomes for use in the M measure.

Franken Performance Measure

Franken [60] introduced a performance measure loosely based on the work done by Messerschmidt and Engelbrecht [106, 107], referred to as the F measure. The F measure simplifies the M measure by removing the board state outcome probability components and only relying on the actual game outcomes when playing against random players.

Franken defined the F performance measure as follows:

$$F = \sum_{i=1}^{3} x_i f(x_i) \tag{4.1}$$

where x_i is a weight associated with either losing, drawing, or winning and $f(x_i)$ represents the corresponding outcome as calculated by playing against N random agents in both first and second player positions. For statistical significance, Franken measured the average of F over 15 simulations playing as the first player and another 15 simulations playing as the second player. Each simulation in turn consisted of playing 10000 games played against random opponents. The resulting value of F is then scaled to between 0 and 100. This value is referred to as the "Franken performance measure" or F measure.

The F measure worked well for more complicated games such as chess and checkers, as employed by Franken. The F measure incorporated the importance of drawing into the measurement which the M measure did not. Applying the F measure directly to the simple soccer model is not possible as the measurement also incorporates the concept of playing in the first and second positions, a concept that does not apply to many game agent problems, including simple soccer. The first team to kick the ball is calculated randomly. There is no concept of a first or second team in simple soccer.

Modified F Performance Measure

Both the M and the F measures made use of random opponents to benchmark against, and both incorporated the concept of a first and second player in the game. Building on the technique of benchmarking against random opponents, a measurement can be defined for games where there is no notion of first or second player positions:

$$S = \frac{n_{won}}{n_{total}} \tag{4.2}$$

where n_{won} and n_{total} is the number of games won and the total number of games played respectively.

The S measure as defined above does not take into account draws in the game, but can easily be expanded to include a weighted strategy similar to the F measure as follows:

$$S_{expanded} = \frac{w_{won} n_{won} + w_{draw} n_{draw}}{n_{total}} \tag{4.3}$$

where w_{won} and w_{draw} is the weight assigned to games won and drawn respectively. Similarly, losses can be taken into account by adding another term $w_{loss}n_{loss}$. For this study only games won and drawn were taken into account as losses automatically factor in when considering the measurement over the total number of games played. The need for taking draws into account is analysed in the next chapter. Experimental work in this study calculates the S measure over 15000 games.

4.4.2 Domain-Specific Benchmarking

Typically, each problem allows for unique domain-specific measurements to be recorded. For example, function minimisation problems allow for the current function value (domain specific measurement) to be calculated from the candidate solution. The current function value can be compared with the known optimal function value to calculate the current fitness value. For function minimisation problems the smaller the function value, the better the quality of the candidate solution. Two or more function values can be compared and it is easy to determine which function value is better and thus which candidate solution is better. The comparison of the function value with the known optimal function value provides an objective performance measure. In the case of function minimisation problems, using the domain specific measurement makes sense.

In the case of simple soccer, various measurements such as the average number of iterations per goal or the number of goals scored per game can be recorded. However, unlike in the function minimisation case, these measurements are not useful to measure the true performance of the game agents. A high value for number of goals scored per game does not reveal any information about the strategy used to score those goals. The number of goals scored per game is not a objective measurement of the players performance. In the case of simple soccer, using domain specific measurements does not make sense.

In the absence of an objective measurement for simple soccer the parameter optimisation done in this study is performed using the S measure explained in the previous section.

4.5 Parameter Optimisation

This section describes the process used to determine good values for the control parameters of the algorithm.

In order to select good parameter values, existing research was consulted for known well-performing parameter values. The remaining parameters were then optimised using a visualisation technique based on parallel coordinates, described below.

As discussed in section 2.4.5, Blackwell and Bentley [12] found that atomic swarms where 50% of the swarm is charged tended to outperform the other swarm types. Based on this result, the charged particle swarm optimisers used in this study were chosen to be atomic swarms.

As discussed in section 2.4.5, Li and Dam [100] found that the Von Neumann particle swarm optimisation neighbourhood structure tended to outperform the other neighbourhood structures. Based on this result, the neighbourhood structure for the experimental work in this study has been chosen to be the Von Neumann neighbourhood structure.

As discussed in section 2.4.3, Eberhart and Shi [39] introduced the inertia weight parameter to the PSO algorithm. Further investigation by Clerc and Kennedy [27] and Van den Bergh [154] found values that work well for a wide range of problems for the inertia weight and acceleration coefficients to be $w = 0.729844$ and $c_1 = c_2 = 1.49618$. Based on this result, the experimental work in this study sets the inertia weight parameter and constricting coefficient to these values.

All the particles are initialised with a velocity vector set to zero in order to avoid premature explosion of velocities. The choice of a zero initial velocity can also be seen as staying true to the original model of a flock of birds: the velocity at takeoff for a bird is zero.

Each particle swarm had 20 particles. The number of hidden units was set to five. Empirical analysis showed that larger hidden layers did not provide a significant enough improvement in performance to justify the increase in computational cost. Five hidden nodes provided a good tradeoff between computational cost and performance. Particle positions are randomly initialised in the range $(-1, 1)$. Some studies have suggested initialising particles into the range $(\frac{-1}{fanin}, \frac{1}{fanin})$, where $fanin$ is the number of incoming connections for the corresponding neurons [165]. However, this initialisation strategy was

devised for gradient-based training methods, such as back-propagation learning, and no conclusive study has been made as to optimal particle position initialisation for particle swarm optimisation. Table 4.1 summarises the parameter choices.

For the remaining control parameters, as listed in table 4.2, a visual inspection technique based on parallel coordinate visualisation is used to perform parameter optimisation. Franken [57] presented a visual analysis technique where parameters are mapped onto parallel coordinates along with the resulting performance score. Using different colours, relationships between well-performing particles and their corresponding parameter values can be highlighted visually. The parallel coordinate visualisations in this study show the top 5% and 20% particles using highlights with different colours. In contrast to the recommended random sampling technique described by Franken [57], the parameter values in this study were instead sampled from predefined parameter value sets. The parameter value sets are listed in table 4.3. The values for each of the sets were carefully selected to investigate the largest range of values. The large gaps between values in some of the sets allow for exponential traits to be investigated.

Graph 4.1 shows the parallel coordinate visualisation with the region formed by the 5% best performing parameters, as determined by their average S measure, highlighted. A visual analysis makes it apparent that some of the parameters being optimised are

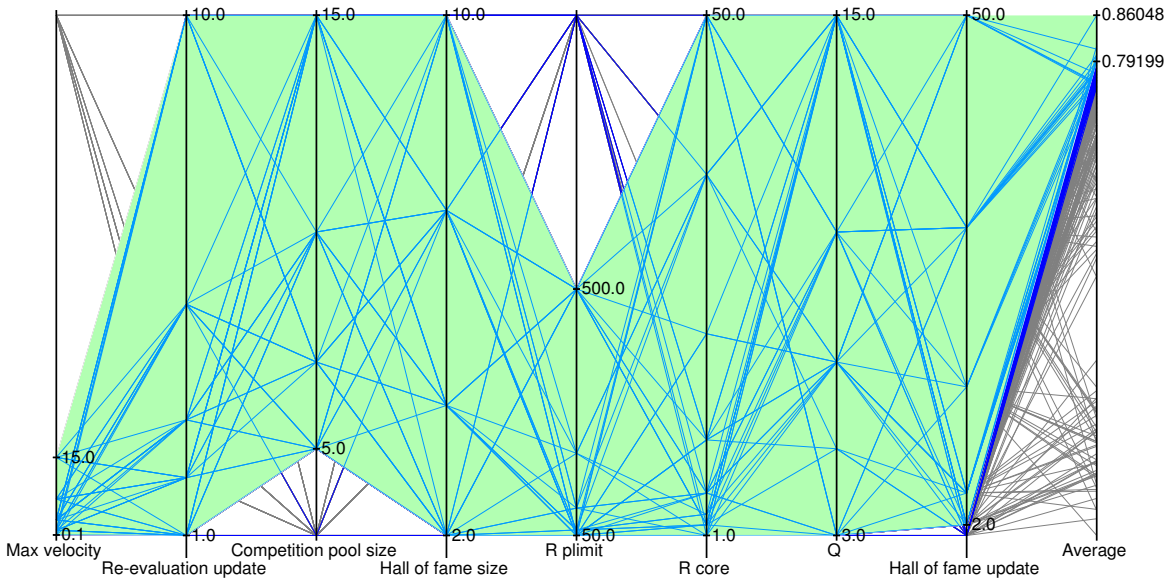
Table 4.1: Fixed algorithm parameter choices.

Parameter	**Value**
Swarm type	Atomic (50% charged)
Neighbourhood structure	Von Neumann
Inertia weight (w)	0.729844
Social constricting coefficient (c_1)	1.49618
Cognitive constricting coefficient (c_2)	1.49618
Initial particle velocity	0
Initial particle positions	$R(-1, 1)$
Swarm size	20 paticles
Hidden layer size	5 hidden nodes

Table 4.2: Control parameters.

Parameter	Meaning
R_{plimit}	Perception limit.
R_{core}	Perception core.
Q	Charge magnitude.
Hall of fame size	Number of particles added to the hall of fame.
Hall of fame update	Number of iterations between hall of fame additions.
$V_{max,i}$	Maximum velocity per dimension of the velocity vector.
Competition pool size	Size of the competition pool.
Re-evaluation update	Number of iterations between personal best position re-evaluations.

non-sensitive to change; that is, all the sample values evaluated performed equally well.



Graph 4.1: Average S measure value sampled for parameter optimisation.

Table 4.3: Parameter value sets to sample from for optimisation.

Parameter	**Set**
R_{plimit}	$\{50, 60, 100, 200, 500, 1000\}$
R_{core}	$\{1, 2, 3, 5, 10, 20, 35, 50\}$
Q	$\{0.001, 0.01, 0.1, 1, 4, 10, 16, 100\}$
Hall of fame size	$\{2, 4, 7, 10\}$
Hall of fame update	$\{1, 2, 5, 15, 30, 50\}$
$V_{max,i}$	$\{0.001, 0.01, 0.1, 1, 3, 7, 15, 100\}$
Competition pool size	$\{3, 5, 7, 10, 15\}$
Re-evaluation update	$\{1, 2, 3, 5, 10\}$

Non-sensitive parameters include the hall of fame size, the charged perception core R_{core}, the charge magnitude Q, and the hall of fame update.

$V_{max,i}$ showed sensitivity with well-performing values sampled in the range $[0.1, 15]$. Closer inspection of the data revealed that no particles with a $V_{max,i} > 15$ or $V_{max,i} < 0.1$ performed well.

Larger competition pool sizes performed well, specifically competition pools larger than five lead to well-performing agents. Larger competition pool sizes increase the quality of the particles' relative fitness measure as more information become available when a particle is measured against a larger group of opponents during the training process. It should be noted that, depending on the problem being optimised, large competition pools increase the computational cost of the optimisation algorithm. For the simple soccer agent training process, a competition pool of size five leads to five games taking place per particle in order to calculate the particles' relative fitness. Taking into account that there were four swarms, each with 20 particles, five games per particle lead to $5 \times 4 \times 20 = 400$ games being played every iteration. Increasing the competition pool size to 15 increases the number of games being played to $15 \times 4 \times 20 = 1200$. Iterations where the personal best positions are re-evaluated increase the number of games played to 800 and 2400 respectively. It is easy to see how the number of games, and thus the computational complexity, starts to explode for large competition pools.

The charged particle swarm perception limit parameter, P, performed well for values between 50 and 500.

The top 5% from the sampled parameters showed S measure values in the range $[0.79199, 0.86048]$. The best parameter configuration resulted in 86.048% wins when playing against random opponents. Reporting the S measure alone does not show consistency in the performance measurement. A good parameter configuration should consistently lead to the training of well-performing game agents. In order to investigate the consistency, the standard deviation of the measured performance over the 30 simulations is calculated.

Table 4.4 contains the average S measure and corresponding standard deviation over the 30 simulations for the 10 best-performing parameter configurations. High standard deviations were reported for each of the top 10 parameter configurations, the average standard deviation for the top 10 results being 0.1697333. This standard deviation translates into an average performance variance of up to 17% over the 30 simulations. A low variance would have indicated the training algorithm consistently resulted in well-performing game agents. A high variance makes it clear that the training algorithm does

Table 4.4: Top 10 performing parameter configurations average S measure and standard deviation over the 30 simulations.

Parameter configuration #	**Average S measure**	**Standard deviation**
1	0.860482222	0.118019517
2	0.81516	0.157688564
3	0.810233333	0.17393864
4	0.803328889	0.171130587
5	0.801373333	0.216877893
6	0.797237778	0.165403035
7	0.793715556	0.193927449
8	0.791993333	0.162750292
9	0.791793333	0.202701309
10	0.790115556	0.134895304

not consistently result in well-performing game agents. Before selecting the optimised parameter values a more in-depth analysis of the performance variance is required to identify the cause of the high variance.

The next section investigates the performance variance. A possible cause is identified as to why the performance variance is so high.

4.6 Performance Variance Analysis

Sample 5 in table 4.4 showed a 21.69% variance. In order to determine why the variance is so high over the 30 simulations, the results for each of the 30 simulations were explored. Table 4.5 contains the resulting S measure for each of the 30 simulations performed. Each simulation used the same parameter configuration. The S measure reported here was calculated over 15000 games.

A quick inspection of the individual simulation results, as listed in table 4.5, shows the presence of outliers in the data (bolded in the table for easy reference). Simulation 25 trained the team being measured up to an S measure of only 0.057333333. On average

Table 4.5: S measure values for all 30 individual simulations showing the outliers in the recorded measurement values.

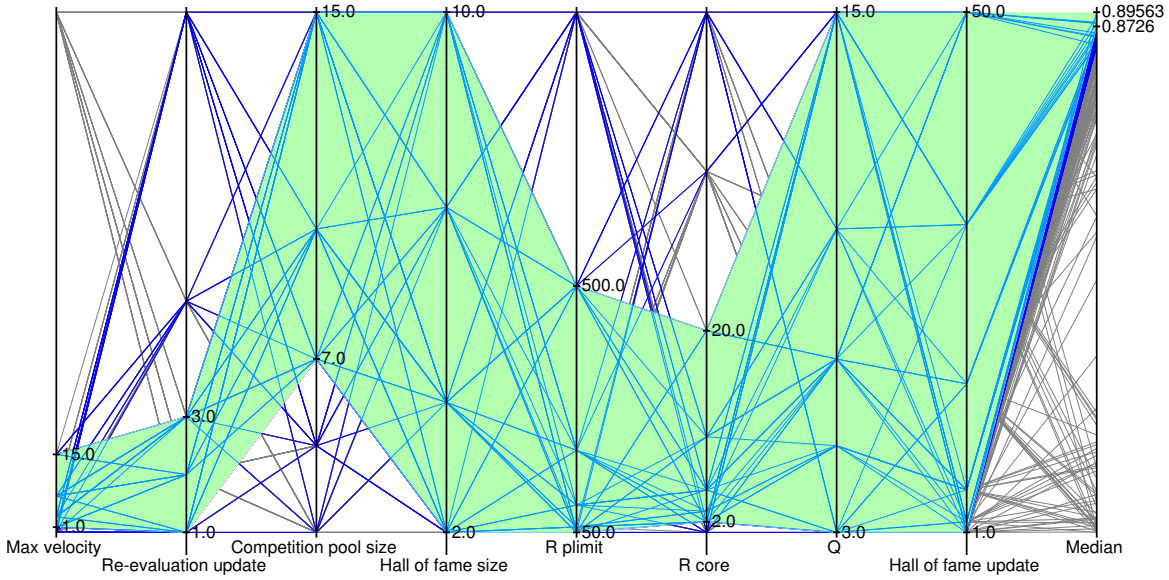
#	S measure	#	S measure	#	S measure
1	0.955333333	11	0.993666667	21	0.840266667
2	0.918733333	12	0.8836	22	0.860133333
3	0.849866667	13	0.9206	23	0.8222
4	0.873666667	14	**0.454666667**	24	**0.657666667**
5	0.930333333	15	**0.277333333**	25	**0.057333333**
6	0.750866667	16	0.869866667	26	0.8812
7	0.998933333	17	0.8862	27	0.742333333
8	0.909733333	18	0.8856	28	0.922533333
9	0.8978	19	0.970933333	29	0.8646
10	0.886533333	20	**0.516**	30	0.762666667

the team won less than 6% of the games when playing against random opponents. On the other hand, teams composed of randomly acting players won 94% of the games against the trained team. In order for a team to win, one of the players would need to move towards the ball and kick it in the direction of the opposing teams' goal. When playing against a randomly acting team, this simple strategy should make it quite easy to win more than 6% of the games. In this instance, the team did not even manage to win 6% of the games. The collected data unfortunately shows only the measured performance for one of the two teams being trained as only one team was measured in the competitive environment; a more complete analysis of the outliers is required to form any theories as to the root cause of the outliers.

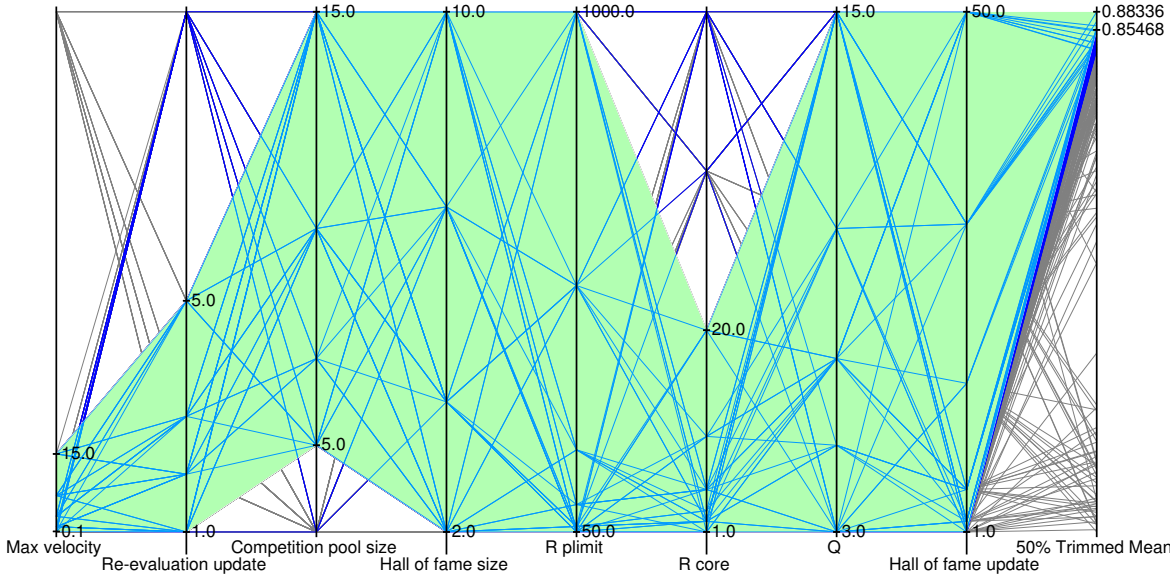
This section discusses the impact that the existence of outliers in the data have on the parameter optimisation process, followed by an in depth analysis of the outlier problem.

4.6.1 Parameter Re-optimisation

The parameter optimisation performed in section 4.5 made use of the average S measure over 30 simulations to determine which parameter values are best. The outliers that was shown in the previous section would influence the optimisation process by reducing the average S measure for a number of the tested parameter combinations. To avoid the outliers from skewing the parameter optimisation, two alternative measurements were evaluated. The first alternative is to plot the median S measure onto the parallel coordinate visualisation (graph 4.2 presents the result). The second alternative is to plot the trimmed mean (or trimmed average) S measure onto the parallel coordinate visualisation. The trimmed mean is calculated by taking the average over a percentage of the records, discarding records on either side of the mean. With a high enough percentage the trimmed mean will effectively discard all the outliers from the recorded data. It should be noted that not only are outliers discarded from the mean calculation, but non-outlier records will also be discarded. Graph 4.3 presents the parallel coordinate visualisation for a 50% trimmed mean. The 50% value can be seen as excessively high and was chosen to provide an extreme to compare the median against. Discarding half (50%) of the records should help to provide a clear, outlier free, representation of the recorded data.



Graph 4.2: Median S measure values along with the corresponding parameter values.



Graph 4.3: 50% trimmed mean S measure values along with the corresponding parameter values.

A comparison of graphs 4.2 and 4.3 with the earlier graph 4.1 reveals that two additional parameters, R_{core} and the *re-evaluation update*, showed sensitivity in influencing the S measure.

Smaller values for the *re-evaluation update* parameter cause personal best positions to be marked as stale much faster, leading to more frequent relative fitness re-evaluations. Each re-evaluation requires a competition to take place in order to calculate a new relative fitness score. The re-evaluation step is identical to the step required to calculate the relative fitness for the current particle position. Larger values for the *re-evaluation update* might be desirable in cases where computational power is not readily available to reduce computational cost as fewer competitions need to be performed. Too large values may, however, degrade performance, as noted by Carlise and Dozier [17] in section 2.4.5 as the relative fitness stored for a particle is no longer valid and might actually be worse than the current position. For this reason smaller values are typically desirable for better performance as more information about the dynamic search space is available.

The charged swarm parameters R_{plimit} and R_{core} exhibit interesting behaviour over the three graphs. In graph 4.1 only R_{plimit} showed sensitivity with values in the range $[50, 500]$ leading to high S measure values. In graph 4.2 both R_{plimit} and R_{core} showed sensitivity with values in the ranges $[50, 500]$ and $[2, 20]$ leading to high S measure values. In graph 4.3 only R_{core} showed sensitivity with values in the range $[1, 20]$ leading to high S measure values. Since both R_{plimit} and R_{core} showed sensitivity, this study limits both parameters to the ranges $[50, 500]$ and $[1, 20]$ respectively. Table 4.6 summarises the optimised parameter value ranges; both computationally inexpensive as well as more accurate choices are shown. The experimentation conducted for this study made use of the computationally inexpensive parameter values.

4.6.2 Outlier Analysis

Graph 4.4 depicts the S measure for 30 simulations over 2000 iterations. A quick visual inspection of the S measure values at iteration 500 reveal that the majority of S measure values are clustered above 0.6. Two outliers can be seen at iteration 500 with S measure values of 0.3 and 0.4 respectively. Graph 4.5 depicts the outlier, team A, with an S measure value of 0.3 at iteration 500 along with the recorded S measure for the opposition

Table 4.6: Summary of optimised parameter values. Computationally inexpensive choices are listed as well as more accurate choices.

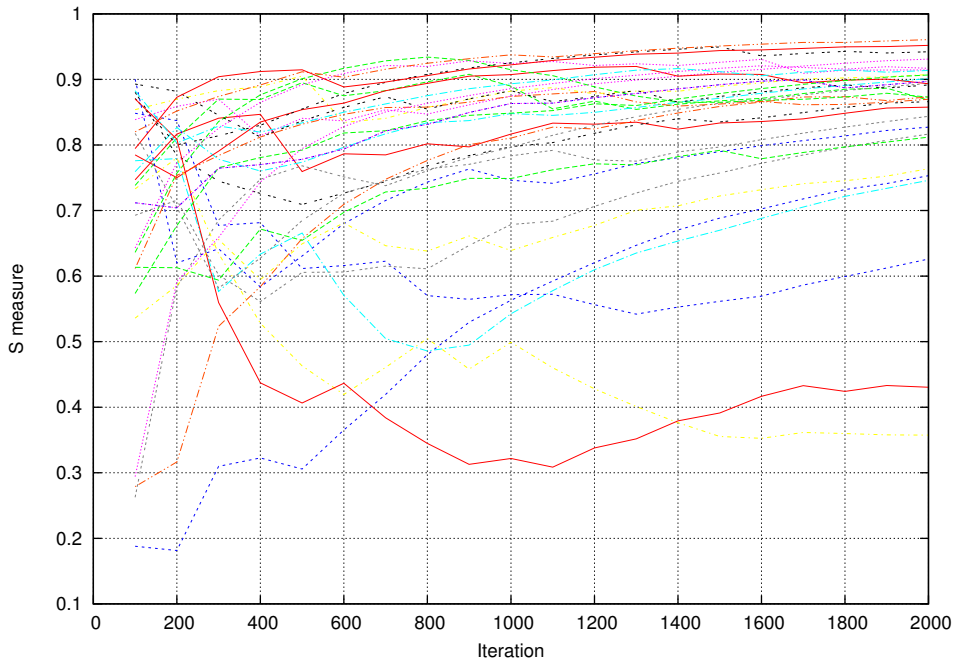
Parameter	Range	Inexpensive	Accurate
R_{plimit}	$[50.0, 500.0]$		
R_{core}	$[2.0, 20.0]$		
Q	$[3.0, 15.0]$		
Hall of fame size	$[2, 10]$	2	10
Hall of fame update	$[1, 50]$	50	1
V_{max}	$[0.1, 15.0]$		
Competition pool size	$[5, 15]$	5	15
Re-evaluation update	$[1, 5]$	5	1

team, team B. Team B has an S measure value of around 0.9 at iteration 500. Both team A and B used the same parameter configuration.

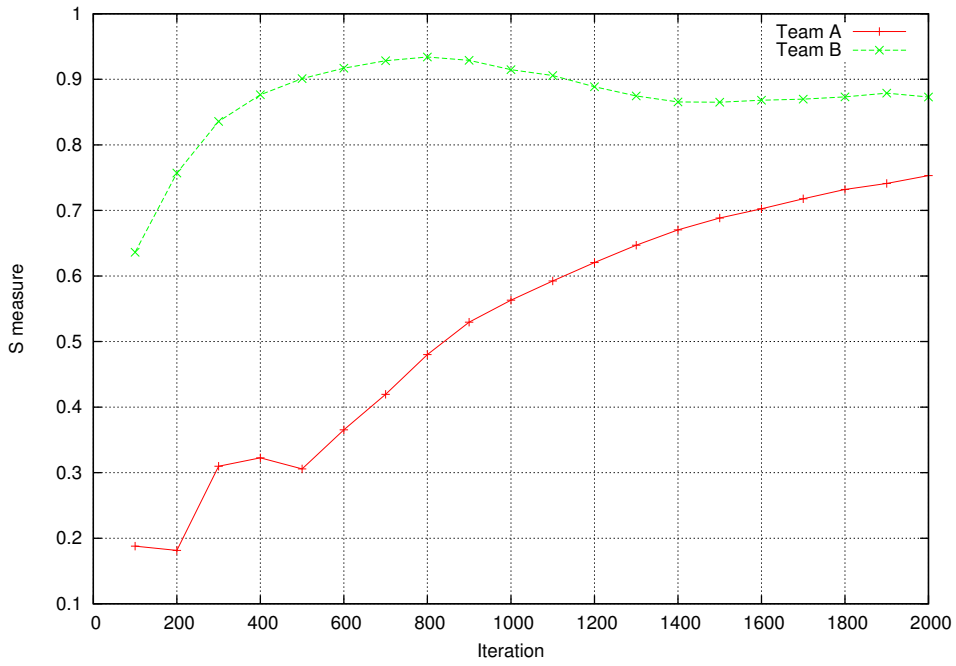
Graph 4.6 depicts both team A and team B in addition to the average, median, 50% trimmed mean and standard deviation over all 30 simulations. It is noteworthy that team B's performance values lies almost perfectly on the median and 50% trimmed mean from around iteration 1400. The effect of the outliers is also clearly visible on the average as it is notably lower than the median or trimmed mean. The lower value is due to the lower outlier values. The standard deviation shows no notable decrease over the 2000 iterations after the initial drop up to iteration 300. The relatively high standard deviation indicates the training algorithm is not consistently training teams to a high S measure.

The available information shows that the evolutionary process is not always able to drive the performance of the swarm high enough within 500 iterations. Increasing the iteration limit to 2000 showed a definite increase in performance when compared to an iteration limit of 500. However, even with an iteration limit of 2000 outliers remained visible in the data.

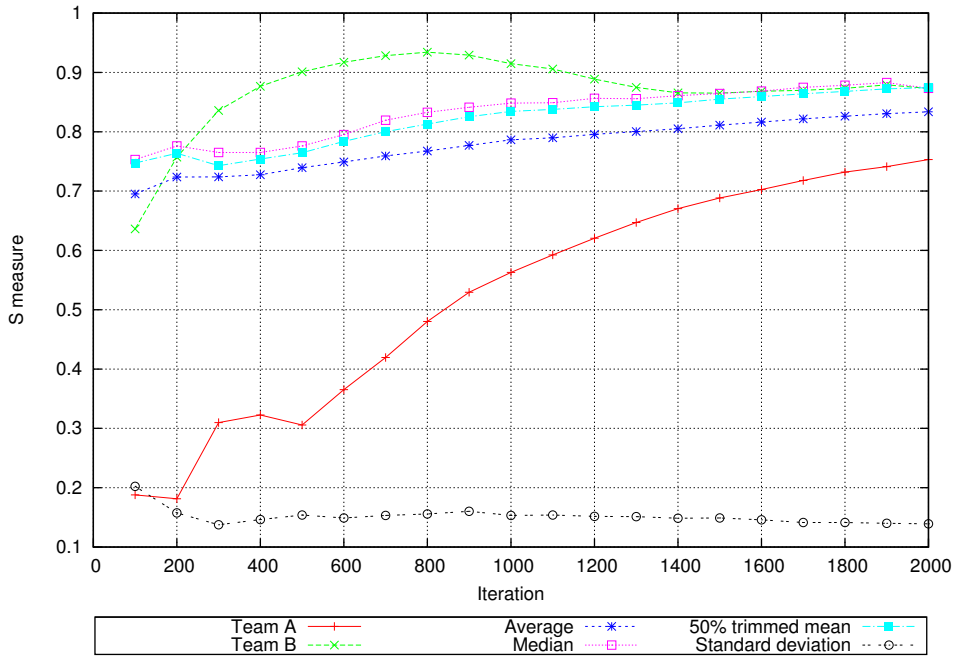
The outliers indicate the training algorithm is failing to consistently train well-performing simple soccer players using a given parameter configuration. One interpre-



Graph 4.4: S measures over 2 000 iterations for 30 teams.



Graph 4.5: Example of an outlier's S measure (team A) and the opposing team it trained against's S measure (team B).



Graph 4.6: Team A, team B, average, median, 50% trimmed mean S measure and standard deviation over all 30 simulations.

tation of these results might be that the relative fitness function does not reveal enough information to drive the evolutionary process fast enough to a better-performing area of the hyper-dimensional search space. Further analysis of the relative fitness function and the effect thereof on the training performance is required to confirm or deny this hypothesis. A detailed analysis of the relative fitness function is presented in the next chapter.

4.7 Summary

The objective of this chapter was to present and analyse a new training algorithm based on the computational techniques previously discussed and tested on the simple soccer model, as presented in the previous chapter. The CCPSO training algorithm was presented in section 4.3, the various parameters were discussed in detail. It was also shown how the training algorithm could be applied to the simple soccer model.

An overview of benchmarking measurements was provided, as used in previous works.

In the light of these benchmarking functions parameter optimisation was performed to optimise each of the variables in the new training algorithm. Early results obtained from the parameter optimisation revealed a critical weakness in the training algorithm, i.e. the presence of outliers in the training data. The outliers indicated the training algorithm failed to consistently train well-performing simple soccer players. It was hypothesised that the relative fitness function could be the root cause of the outliers.

The next chapter investigates the hypothesis that the relative fitness function is responsible for the weak performance. The FIFA league ranking fitness function developed for this study is presented to address the outlier problem.

Chapter 5

Relative Fitness

"Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted."

Albert Einstein (1879 - 1955)

The training algorithm presented in the previous chapter was analysed using the S measure by playing 15000 games against random players. The objective of this chapter is to present alternative relative fitness functions that can be used to improve the algorithms performance. A new relative fitness function, developed specifically for this study, is presented in the first part of this chapter. The second part of this chapter presents early results obtained using the new relative fitness function.

5.1 Introduction

The importance of the relative fitness function was discussed in section 2.5.2. The previous chapter made the hypothesis that the weak performance may be due to the relative fitness function not performing well enough. In order to investigate this hypothesis further, analyses of alternative relative fitness functions are discussed in this chapter. The goal of this chapter is to develop a more suitable and better-performing relative fitness function with the hope that it would solve the outlier problem.

Section 5.2 discusses various relative fitness functions. The relative fitness functions are categorised based on whether the function introduces a bias on the training process

or not. Section 5.3 repeats the parameter optimisation presented in the previous chapter, this time, using the newly developed FIFA league ranking relative fitness function. Finally, section 5.4 investigates whether the outliers noted in the previous chapter is still present in the training data.

5.2 Relative Fitness

As noted in section 2.5.2 the relative fitness function is the driving force behind the evolutionary process in a competitive training environment. The better the relative fitness function, the better the performance of the training algorithm. The goal of this section is to provide an overview of alternative relative fitness functions.

The concept of biasing behaviour towards a specific gameplay strategy through a fitness function is discussed in section 5.2.1. Fitness functions that do not introduce a bias in the behaviour of the trained agents is discussed in section 5.2.2. The *FIFA league ranking* fitness function that was specifically developed for this study is introduced in section 5.2.3.

5.2.1 Avoiding Biased Behaviour

One of the primary focuses of this study is to develop a training technique that trains playing strategies from zero knowledge. Zero knowledge implies that the behaviour of the players may not be predetermined or guided towards a specific playing style or strategy. It is of key importance that the relative fitness function therefor does not introduce a bias towards specific strategies by rewarding a certain gameplay style more than others.

The next two subsections discusses two alternative relative fitness functions to train game agents based on previous studies. Both of these relative fitness functions introduce a bias by rewarding certain gameplay styles more than others.

Simple soccer absolute fitness

Lee and Jeong developed an absolute fitness function for their GA to train simple soccer game agents [77]. The simple soccer absolute fitness function to be maximised is defined

as follows:

$$\mathcal{F} = \sum_{i=1}^{i=N} L'(i) \tag{5.1}$$

where

$$L'(i) = L_0 + \sum_{t=0}^{t_{final}} (f_{goal-A}(t) + f_{posses-A}(t) + f_{goal-B}(t) + f_{posses-B}(t)) \tag{5.2}$$

with

$$\begin{aligned}
f_{goal-A}(t) &= \begin{cases} 1000 & \text{if team } A \text{ scores at iteration } t \\ 0 & otherwise \end{cases} \\
f_{goal-B}(t) &= \begin{cases} -90 & \text{if team } B \text{ scores at iteration } t \\ 0 & otherwise \end{cases} \\
f_{possess-A}(t) &= \begin{cases} 10 & \text{if team } A \text{ possesses the ball at iteration } t \\ 0 & otherwise \end{cases} \\
f_{possess-B}(t) &= \begin{cases} -10 & \text{if team } B \text{ possesses the ball at iteration } t \\ 0 & otherwise \end{cases}
\end{aligned} \tag{5.3}$$

where i is the i'th simulation of the game, N is the number of simulations (Lee and Jeong defined N to be five for their work), t_{final} is the number of iterations per simulation until a goal is scored or 30 if no goal is scored in the first 30 iterations, t is the t'th iteration of the simulation, and L_0 is a constant set to 400.

Analysis of equation (5.2) reveals that the maximum value for $\mathcal{F}$ is achieved when $f_{possess-A}(t) = 10$ for $t < 30$ and $f_{goal-A}(t) = 1000$ for $t = 30$. This maximum fitness value can only be achieved by holding on to the ball for all iterations up to the last, at which point the team must score a goal. Because $\mathcal{F}$ is maximised the optimisation process will likely result in players exhibiting this specific behaviour. Thus introducing a bias towards this specific gameplay strategy.

This bias violates the *training from zero-knowledge* goal of this study.

Rampup absolute fitness

Fehervari and Elmenreich used an EA to evolve neural network controllers to play soccer in a simplified soccer simulation [50]. They made use of a fitness function, hereafter

referred to as the *rampup absolute fitness function*, to drive the evolutionary process. The rampup absolute fitness function was defined as follows:

$$F_{ramp} = W_p P_p + W_{bd} P_{bd} + (W_k P_k - W_{fk} P_{fk}) + W_{bg} P_{bg} + W_s P_s \tag{5.4}$$

with the parameter choices as summarised in table 5.1, where P_p, P_{bd}, P_k, P_{fk}, P_{bg} and P_s represent the number of points and W_p, W_{bd}, W_k, W_{fk}, W_{bg} and W_s are the weighting of these points.

Table 5.1: Rampup absolute fitness function parameters.

Parameter	**Symbol**	**Weight**
Field distribution	P_p	10^0
Distance to the ball	P_{bd}	10^3
Number of kicks	P_k	2×10^4
Number of false kicks (ball is kicked out of bounds)	P_{fk}	10^4
Ball distance to the opponent's goal	P_{bg}	10^5
Number of scores	P_s	4×10^6

Points are awarded every 5 seconds for field distribution, every 4 seconds for distance to the ball and every 2 seconds for distance to the opponent's ball. Field distribution is based on 64 control points: if a player is closest to a point, a point is awarded. A point rewards distance to the ball if a player in the team is closest to the ball. Kicking-related points are awarded a point per kick. Figure 5.1 presents a visual representation of how the evolutionary process is guided towards specific behaviour.

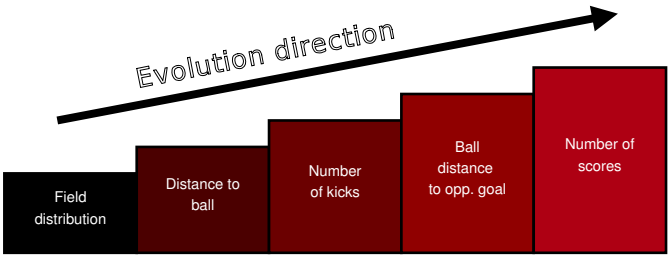


Figure 5.1: Rampup absolute fitness direction of evolution.

It is clear to see how the *rampup absolute fitness function* drives the evolutionary process towards a specific bias. Inspection of the reward function reveals that only offensive-type strategies are rewarded - a team playing more defensively will not be rewarded enough to survive the evolutionary process, given enough generations. The objective of the soccer game is indeed to score goals; however, drawing behaviour is also worth rewarding. Drawing a game against a very good team can be compared to winning against a mediocre team.

The reader should note that Fehervari and Elmenreich made use of a single population per team; thus, a single individual represented two game agents in contrast to the work done in this study, where game agents coevolve as individual populations (or particle swarms, in this case).

5.2.2 Unbiased Fitness

The previous section showed that it is possible to introduce bias into the training process through the relative fitness function. Developing a well-performing unbiased relative fitness function is extremely important: it has already been shown in the previous chapter how a weak-performing function influences the training process. Two unbiased relative fitness functions were developed for the simple soccer problem by the author of this study. The two relative fitness functions are discussed below.

Fixed Reward

The early experimental results reported in the previous chapter were obtained from experiments that made use of a fixed reward scheme aimed at encouraging victory, and discouraging defeat by penalising losses more excessively than rewarding victory. The results in section 4.5 proved promising, with well-optimised parameters leading to high scoring performance, as measured by the S measure. The experimental results did, however, indicate the presence of outliers. It is hypothesised that the outliers are caused due to the lack of information revealed by the fixed reward relative fitness function.

The reward scheme used in the previous chapter rewarded victories with one point, draws with zero points and defeats with -2 points [22, 23, 52]. In contrast to this point

structure international soccer leagues typically use a point system where three points are awarded for a victory, one point for a draw, and zero for a defeat.

Updating the point allocation to be inline with the soccer league scoring system will not address the primary weakness of the fixed reward scheme. Draws will become a less desirable outcome, but no additional information about the actual game performance will be revealed. A very good team winning 15 : 0 against a very bad team will be rewarded the same as a team winning 9 : 6 against a very good team. This leads to a low diversity between the relative fitness value of particles in a swarm in cases where a small number of games are played. Higher diversity in the relative fitness function reveals more information about the search space, which in turn, allows for faster convergence on well-performing areas of the search space. A relative fitness function that reveals more information about the search space is thus more desirable.

Goal Difference

In order to increase population diversity, more information must be gathered from the competitions being played. One approach is to reward points based on the score difference (and not just based on the competition outcome) between the two teams competing in a competition. For the simple soccer model, the score difference is simply the goal difference at the end of the game. A team winning 15 to zero will be rewarded with 15 points for their victory, whereas the losing team will receive a penalty of -15 points for their defeat. A draw will result in both teams receiving zero points. The relative fitness, $\mathcal{F}(t)$, for iteration t is calculated using the following formula:

$$\mathcal{F}(t) = \sum_{n=1}^{N} (\mathcal{G}_n - \mathcal{O}_n) \tag{5.5}$$

where $\mathcal{G}_n$ is the number of goals scored in the n'th competition, $\mathcal{O}_n$ is the number of goals the opponents scored in the n'th competition, and N is the number of games played per tournament.

The goal difference relative fitness function gathers more information from each game being played when compared to the fixed reward relative fitness function. It is, however, more influenced by the game rules than a fixed reward scheme. For instance, if the game is played only up to five goals it does not take the number of game steps (or iterations)

into account for a team to score the five goals. One team might score five goals very fast, whereas another team might take a very long time to score five goals. Limiting the game in question to a fixed number of steps (or iterations) addresses this problem to some extent. The fast scoring team will still be able to score five goals, whereas the slow scoring team might only score, say, two goals. This approach may not always be an option for all games as certain games dictate game-end conditions as part of the game rules and iteration limits cannot be applied. For simple soccer, an iteration limit can be added.

5.2.3 FIFA League Ranking

An alternative approach to increasing the phenotypic diversity is to make use of the Official FIFA World Ranking points system.[1] After the 2006 FIFA World Cup, a revised ranking calculation procedure was introduced to significantly simplify the procedure. The new ranking system makes use of the following formula to calculate the points awarded to a team per match:

$$P = M \times I \times T \times C \times 100 \tag{5.6}$$

where

- **M** indicates if the match ended in a victory (3 points), a draw (1 point), or a defeat (0 points). In cases where a penalty shoot-out is required, the winning team is awarded 2 points and the losing team 1 point. Should a team lose a match, no points are awarded.
- **I** indicates the importance of the match based on a weight scheme. For friendly matches $I = 1.0$, for World Cup qualifier and continental qualifier matches $I = 2.5$, for continental final competitions and FIFA confederation cup matches $I = 3.0$, and for World Cup final competitions $I = 4.0$.
- **T** indicates the strength of the opposition, calculated as

$$T = \frac{200 - (\texttt{opposition rank position})}{100} \tag{5.7}$$

[1] http://www.fifa.com/worldfootball/ranking/procedure/men.html

The top team is assigned $T = 2.0$ and teams ranked lower than 150 is assigned $T = 0.5$. Rank is based on the most recent FIFA ranking publication.

- **C** indicates the strength of the confederation. For intercontinental matches, the average of the confederations the two competing teams belong to, is used. The confederation strength is based on the number of victories by the confederation at the last three FIFA World Cup competitions. After the 2010 FIFA World Cup the strengths were *UEFA* 1.00, *CONMEBOL* 1.00, *CONCACAF* 0.88, *AFC* 0.85, *CAF* 0.86, and *OFC* 0.85.

- **P** indicates the points awarded to the team in this match.

Generally speaking, the greatest winners in the new point system are teams who win competitive matches against high-ranking opponents. Friendly matches and draws provide limited gains and losses provide none.

Rank is calculated for a team as the sum of all the games played in the last four years devalued over the period (100% for the first year, 50% for the second year, 30% for the third year, and 20% for the fourth year back).

The ranking system introduced by FIFA recognises the fact that a victory against a good team (or a high-ranking team) is more difficult to achieve - scoring goals against a good team is inherently more difficult, and should thus be rewarded more than a victory against a bad team (or a low-ranking team). The ranking system also takes into account past results, albeit at a reduced weight when compared to current results, thus avoiding a situation where a single team can dominate the ranks based on a single year's results.

Competitive coevolution makes use of tournaments to determine a relative fitness for each individual. The relative fitness can be seen as a rank. Similarly, the relative fitness function can be seen as a ranking system. Building on this idea, the experimental work in this study made use of a new relative fitness function based on the FIFA ranking system. Points are calculated using:

$$\mathcal{P}(t) = \sum_{n=1}^{N} (\mathcal{M}_n \times \frac{200 - \mathcal{T}_n}{100}) \tag{5.8}$$

where N is the number of games played in the competitive coevolution tournament, $\mathcal{M}_n$ is the n'th game outcome, defined as:

$$\mathcal{M}_n = \begin{cases} 3 & \text{for a victory} \\ 1 & \text{for a draw} \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

and $\mathcal{T}_n$ is the n'th opposition team rank. Each tournament team consists of randomly chosen team members. The use of randomly constructed teams creates a problem as no team rank can be maintained with consistency between iterations. Instead, the rank must be maintained per player, and then calculated for a team based on the members. The opposition team rank, $\mathcal{T}_n$, is calculated by averaging the rank of the members of the opposition team as follows:

$$\mathcal{T}_n = \frac{\sum_{p=1}^{P} r(p)}{P} \tag{5.10}$$

where function $r(p)$ represents the rank of player p in a team consisting of P members. A player's rank is determined based on the relative fitness for the particle in the previous iteration. The relative fitness, $\mathcal{F}(t)$, for iteration t is calculated using

$$\mathcal{F}(t) = \mathcal{P}(t) + 0.5 \times \mathcal{P}(t-1) + 0.3 \times \mathcal{P}(t-2) + 0.2 \times \mathcal{P}(t-3) \tag{5.11}$$

taking into account previous points awarded using the same weighting scheme as the FIFA ranking system.

Two concepts not incorporated into the newly developed relative fitness function are the addition of a game importance weight (I in the FIFA equation) and confederation strength (C in the FIFA equation). In theory it would be possible to incorporate these two components if the training algorithm made use of multiple populations for a single position. Currently, the algorithm uses a single population per position, rendering these components redundant.

As historic performance is indirectly maintained by the rank, $r(p)$, used when determining the new fitness value (which in turn determines the new rank) the function is further simplified for this study to $\mathcal{F}(t) = \mathcal{P}(t)$.

It should be noted that the FIFA league ranking system does not incorporate any problem domain information other than game outcome similar to the fixed reward func-

tions. This property makes it possible to apply the ranking system as a relative fitness function, as described here, to any competitive coevolutionary problem.

5.2.4 Relative Fitness Function Evaluation

Of the three unbiased relative fitness functions reviewed above, the FIFA league ranking function stands out. FIFA league ranking takes into account past player performance when calculating the relative fitness value. The past performance allows for teams to be ranked over a number of games, allowing teams to be rewarded according to their expected performance. The function also provides a more detailed indication of a players' performance than the other two relative fitness functions discussed. As stated in the previous section, functions that reveal more information about the search space is considered more desirable as they improve training performance.

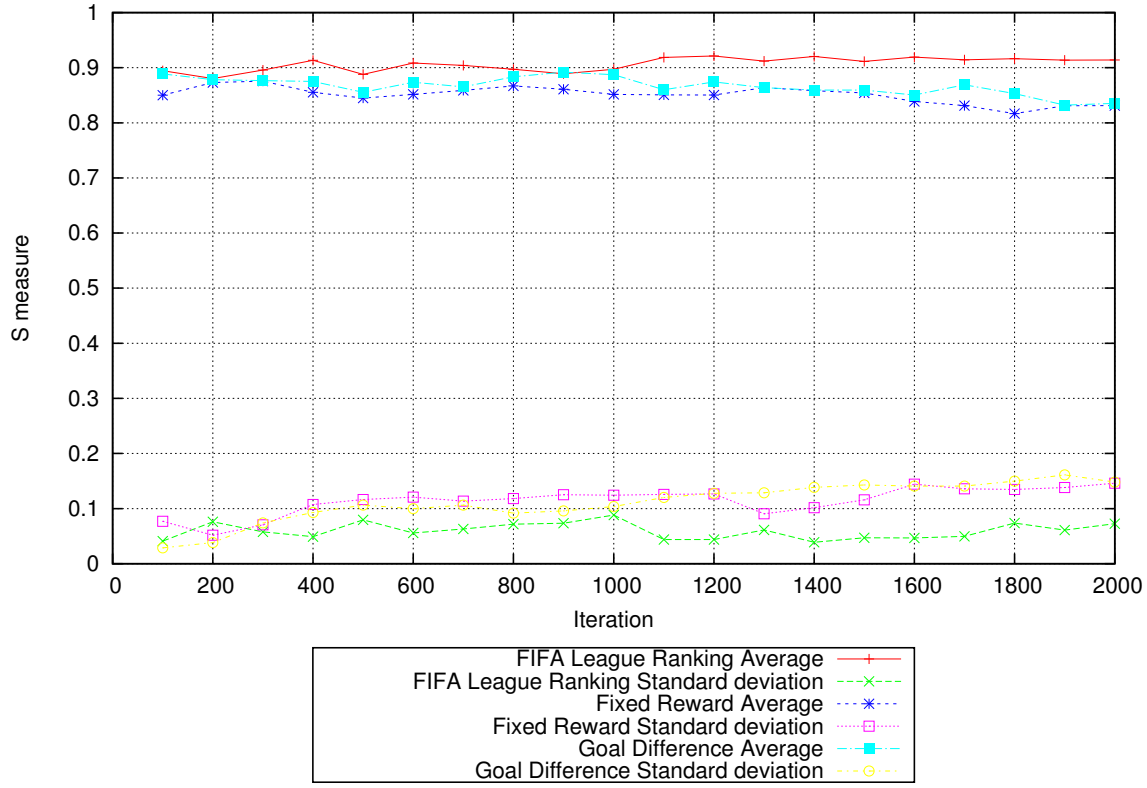
To objectively compare the performance of the three unbiased relative fitness functions a comparative study was carried out. Each function was tested using 30 simulations over 2000 iterations with the optimised parameter configuration.

Graph 5.1 provides the performance comparison of the three unbiased relative fitness functions. The average S measure and standard deviation for each of the three unbiased relative fitness functions is shown. The FIFA league ranking relative fitness function clearly outperformed the other two relative fitness functions. A higher S measure and a lower standard deviation were achieved. The lower standard deviation indicates a higher level of consistency in the algorithms' ability to produce well performing players.

Based on this empirical analysis the remainder of this study makes use of the FIFA league ranking relative fitness function.

5.3 Parameter Optimisation using FIFA League Ranking

Any change to the relative fitness function used by the competitive coevolutionary algorithm changes the shape of the hyper-dimensional search space. Such change to the search space in turn implies that any parameter optimisation done is effectively voided



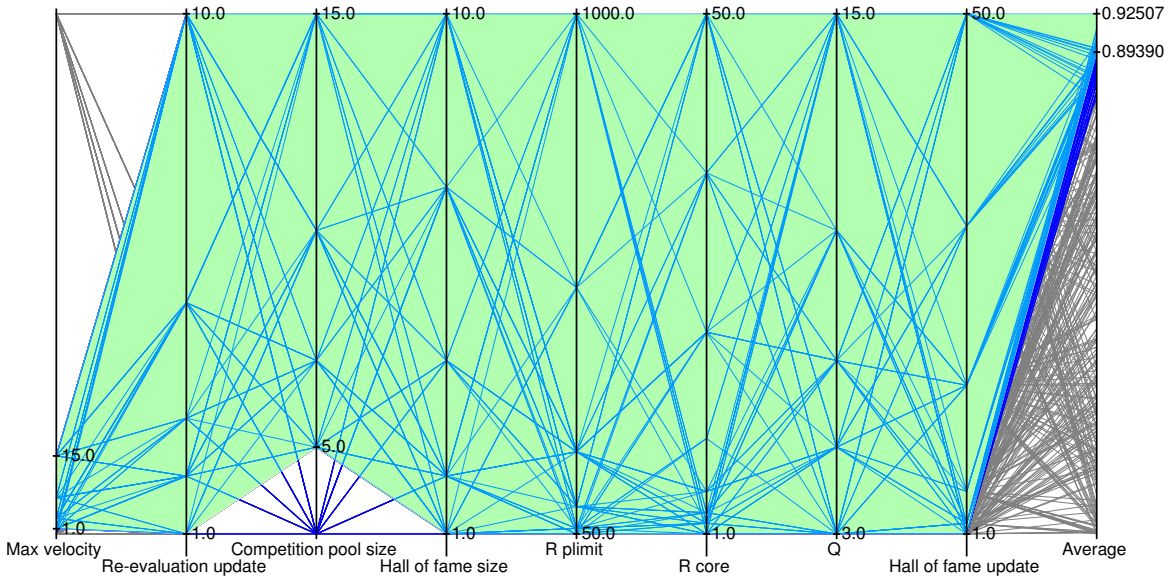
Graph 5.1: Relative fitness function comparison.

because the function being optimised has changed. All the parameters for the competitive coevolutionary algorithm need to be re-optimised for use with the new FIFA league ranking relative fitness function since it reveals the most information about the search space.

The same sample sets identified in table 4.1 were used to repeat the visual parameter optimisation. Graph 5.2 depicts the parallel coordinate visualisation with the top 5% best-performing parameter combinations in the highlighted area.

The performance result for the top 5% best-performing parameters lead to S measure scores in the area of $[0.8939, 0.92507]$, thus winning between 89% and 93% of the games when playing against random opponents. This is already a notable increase in performance when compared to previous best results of 87% to 90% wins shown in graph 4.2 when using the constant reward relative fitness function.

Graph 5.2 shows that, unlike the previous parameter optimisation runs, this time



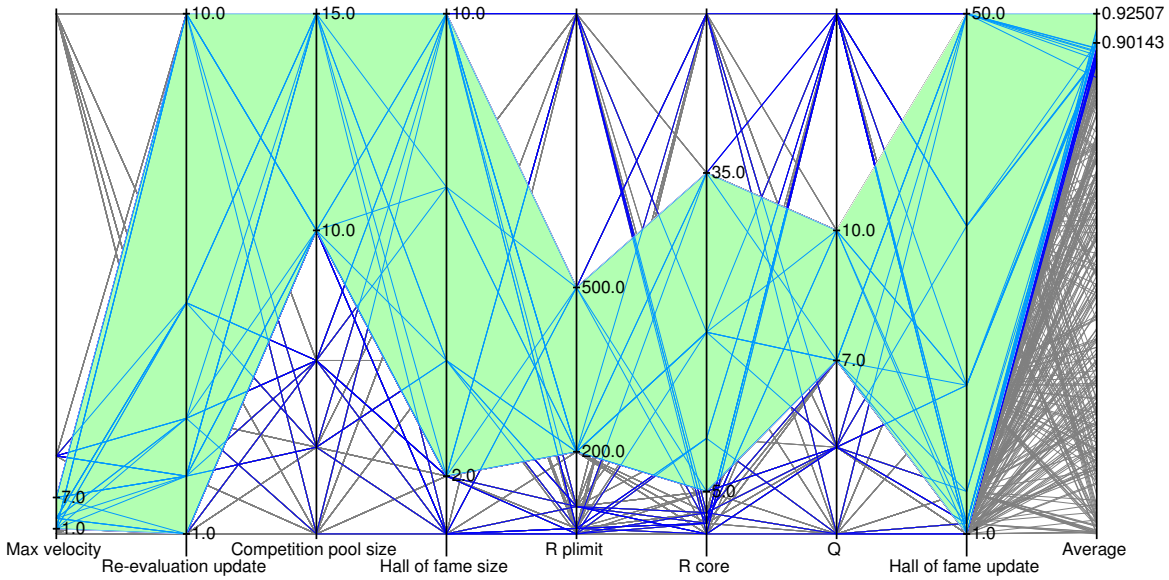
Graph 5.2: Average S measure sampled for parameter optimisation using FIFA league ranking (top 5% highlighted).

only the parameters V_{max} and *competition pool size* showed sensitivity to change with optimised values in the range of $[1, 15]$ and $[5, 15]$, respectively.

In order to identify any other parameters that might show sensitivity, the parallel coordinate visualisation was redrawn with only the top 2% best-performing parameter combinations highlighted in graph 5.3. All parameters except the *re-evaluation modules* and *hall of fame epoch* showed sensitivity. As previously indicated in section 4.5, it is intuitive that a larger competition pool size should lead to better performance. A large competition pool size increases the number of games being played for each evaluation, which in turn increases the resolution of the relative fitness function because more information is being "mined" from the player.

From graph 5.2 the optimised parameters for the algorithm using the FIFA league ranking relative fitness function can be established, as summarised in table 5.2.

The next section reviews if the new relative fitness function addresses the outliers problem that was encountered in the previous chapter.



Graph 5.3: Average S measure sampled for parameter optimisation using FIFA league ranking (top 2% highlighted).

5.4 Outliers analysis

Table 5.3 lists the top 10 best-performing parameter configurations along with the respective average S measure values and standard deviations.

It is immediately apparent from the results that the standard deviations decreased notably when compared with the earlier results, as given in table 4.4. The average standard deviation over the top 10 best-performing parameter configurations has decreased to only 0.051795468; this translate into an average expected performance variance of less than 5.2% over 30 simulations - much better than the previous 17%. For the best-performing parameter configuration, the performance is $92.5\% \pm 3\%$, significantly better than the previous best of $86\% \pm 11.8\%$, clearly much more preferred due to the lower variance and larger base value. A visual inspection of graph 5.5 confirms that there are no longer any outliers present in the collected data.

Graph 5.4 depicts the S measure average, median, team A average, team B average, and standard deviation for all samples over 30 simulations. It should be noted from the graph that there is no notable increase in performance for either team from around

Table 5.2: Summary of optimised parameter values. Computational inexpensive choices are listed as well as more accurate choices.

Parameter	Range	Inexpensive	Accurate
R_{plimit}	$[50.0, 1000.0]$		
R_{core}	$[1.0, 50.0]$		
Q	$[3.0, 15.0]$		
Hall of Fame size	$[1, 10]$	1	10
Hall of Fame epoch	$[1, 50]$	50	1
V_{max}	$[1.0, 15.0]$		
Competition pool size	$[5, 15]$	5	15
Re-evaluation iteration modulus	$[1, 10]$	10	1

iteration 500. The algorithm reaches maximum performance, as measured by the S measure, within the first 500 iterations, with only minor changes in performance thereafter. In other words, the training algorithm stagnates after 500 iterations. The stagnation is a sign that the performance of the trained players is no longer improving. In order to identify possible reasons for the stagnation, the playing strategies of the individual players must be analysed. A full analysis of the playing strategies is the subject of the next chapter.

Graph 5.5 depicts the performance of the individual simulation's for the results presented in graph 5.4. The graph confirms the already made analysis that there is indeed no notable increase in performance after iteration 500. The graph also confirms that outliers are not the reason for the stagnation.

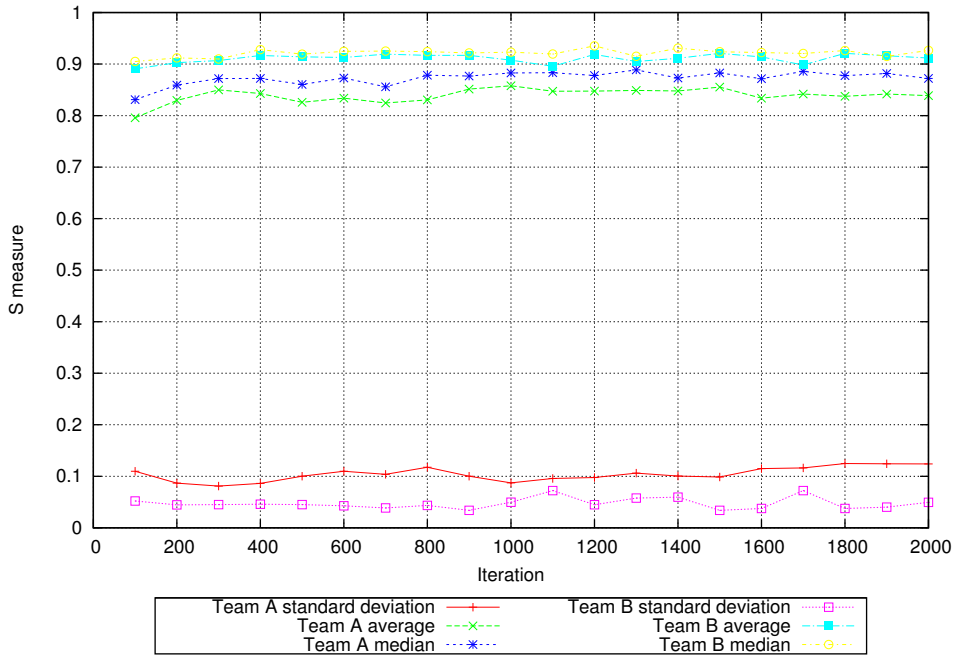
5.5 Summary

The objective of this chapter was to present and analyse various relative fitness functions with the goal of improving the training performance of the CCPSO algorithm proposed in chapter 4.

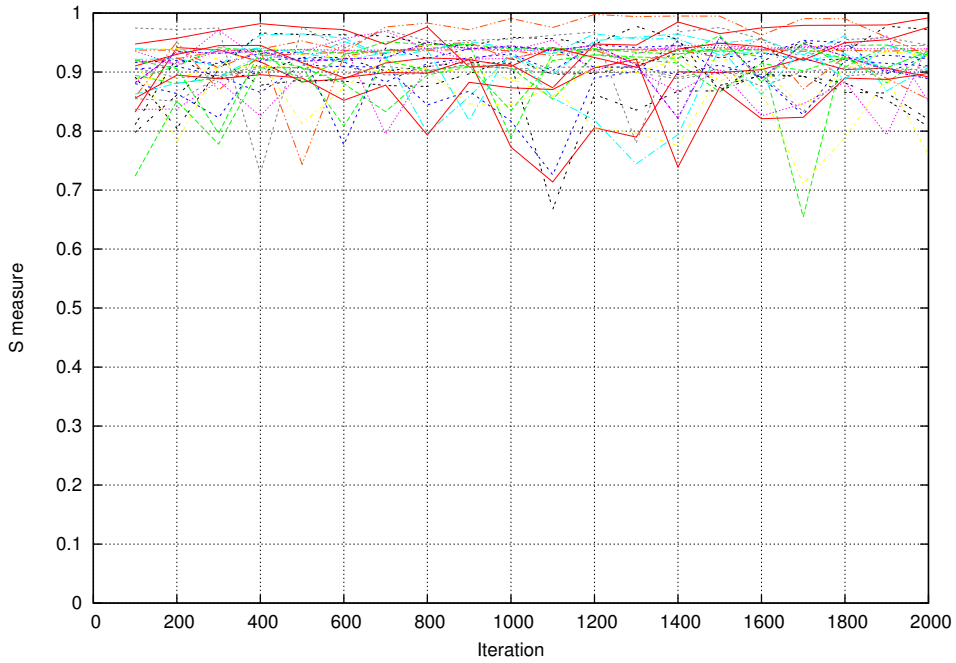
Various design aspects of the relative fitness function were presented and a new func-

Table 5.3: Best performing parameter configurations.

#	Average S measure	Standard deviation
1	0.925072222	0.030269119
2	0.901438889	0.067542552
3	0.907153333	0.045346403
4	0.913194444	0.040913234
5	0.901454444	0.054592311
6	0.894835556	0.079307801
7	0.895322222	0.079173374
8	0.911525556	0.02909921
9	0.898908889	0.054036707
10	0.904540000	0.037673969



Graph 5.4: Average, median, and team averaged S measure using the FIFA relative fitness function.



Graph 5.5: S measure values for 30 simulation over 2000 iterations using the FIFA relative fitness function.

tion based on FIFA's league ranking algorithm was presented. Parameter optimisation was repeated using the newly developed relative fitness function and the results reveal that the problem with outliers experienced with the initial algorithm have been addressed by the new fitness function. Based on the new results, performance was more consistent.

The next chapter expands the performance analysis conducted in chapter 4 to look at the actual gameplay strategies and player behaviours that evolved from the training done using the CCPSO algorithm presented in chapter 4, using the new *FIFA league ranking* relative fitness function presented in this chapter.

Chapter 6

Evolving Playing Strategies

"Strategy without tactics is the slowest route to victory. Tactics without strategy is the noise before defeat."

Sun Tzu (500 BC)

The training algorithm presented in the previous chapter was analysed using the S measure by playing 15000 games against random players. The objective of this chapter is to expand on the already performed analysis by inspecting the behaviour of the players visually. Gameplay strategies are identified and classified in the first part of this chapter, followed by a brief investigation into why the strategies were so simplistic.

6.1 Introduction

The objective of this study is the development of an algorithm capable of evolving gameplay strategies from zero knowledge. The previous chapter showed that the training algorithm is capable of training teams that perform well when playing against randomly behaving opponents. Unfortunately the discrete measurements presented in the previous chapter does not reveal any gameplay strategy information. It was also shown that the training process stagnated from around iteration 500. A visual analysis is needed to rate the effectiveness of the training algorithm as well as identify possible causes for the stagnation.

This chapter presents the gameplay strategies that were evolved as observed from visual analysis carried out on games played by the agents. Section 6.2 reviews each of the identified gameplay strategies in more detail and attempts to identify why each of these strategies emerged from the training process. The evolved strategies that were observed, were extremely simplistic and initial attempts to increase complexity by increasing training iterations proved futile. Section 6.3 investigates the cause of the emergence of such weak gameplay strategies, as well as why the additional training iterations did not improve the performance.

6.2 Gameplay Strategies

Visual analysis revealed that the trained players tended to converge on specific gameplay strategies. For each offensive strategy that developed, the opposing team developed a counter-move to avoid losing points. A number of the strategies identified from the visual analysis, i.e. dual ram, goalie and striker, kickaway and kick-pass goal, are discussed below in more detail.

6.2.1 Dual Ram

The first strategy that emerged is referred to as the *dual ram* strategy. For the dual ram strategy both players in a team literally ram together up to the opposing team's goal position. The two players stay exactly one block away from each other, one behind the other. Figure 6.1 depicts visually what happens during the dual ram offensive.

The defending team has an opportunity to take the ball away from the opposition team's front player (the striker) once it crosses the same block. In the event that the ball owner probability, as defined by the simple soccer rules in section 3.2.1, favours the defending player, the defending player will move onto the same square as the second opposition team player. This second player now has a chance to regain ball ownership and continue the run to the goal.

An alternative approach to the dual ram strategy would have been to move towards the defending team's goal while both players occupy the same block. Remember that two or more players may occupy the same block on the field according to the simple

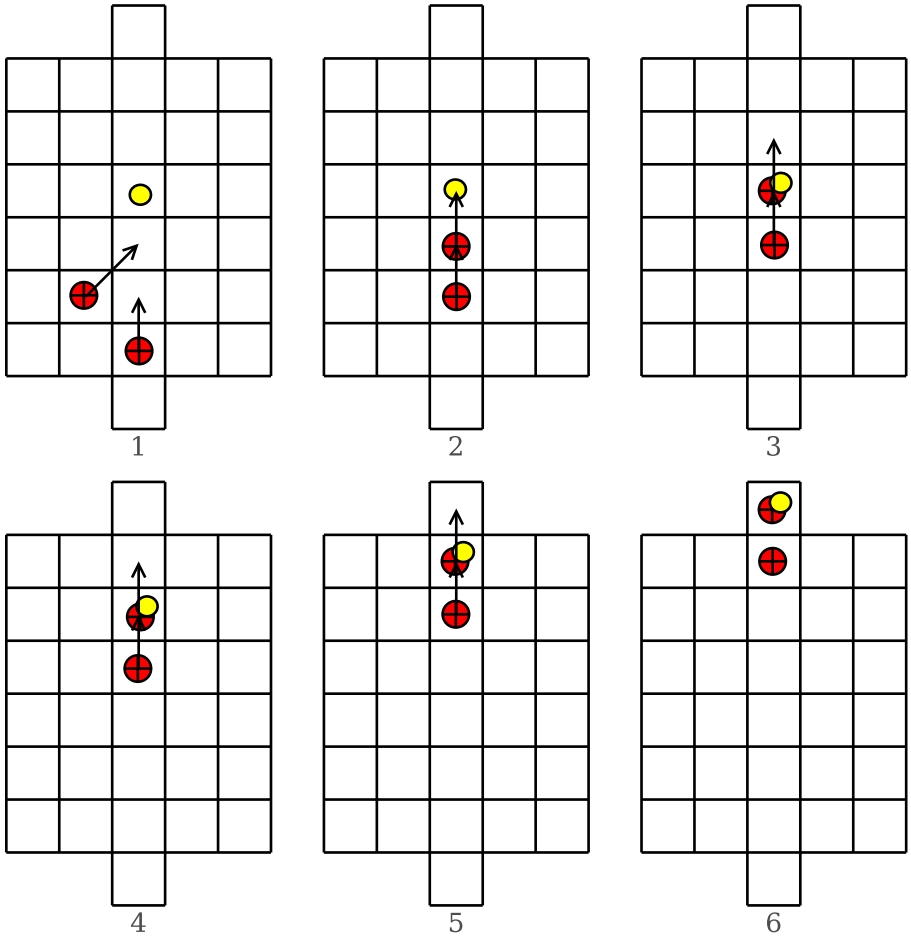


Figure 6.1: Dual ram strategy

soccer rules. It can be deduced from an evaluation of the two approaches presented here in terms of the ball possession probabilities that it is indeed more optimal not to occupy the same block. Consider the following: if both players stayed on the same block as the ball while a player from the opposing team entered the block, the probability to retain the ball would have been $\frac{2}{3}$ or 66.6%. That is two out of the three players are from the team that is attacking, while the defending player would have a probability of $\frac{1}{3}$ or 33.3% to capture the ball. If the players stayed one block away from each other, the first opposition player would have a probability of $\frac{1}{2}$ or 50% of retaining the ball whereas the opposition player would also have a probability of $\frac{1}{2}$ or 50% to capture the ball. Should the opposition player capture the ball, the opposition player will then move onto

the square where the second opposition player is located, leading again to a $\frac{1}{2}$ or 50% probability of the opposition team retaking the ball and a $\frac{1}{2}$ or 50% probability of the defence player keeping the ball. For the opponent team this leads to a total probability of $\frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{3}{4}$ or 75% of keeping the ball, while the defending team has a probability of 25% of capturing the ball. The scenario described here covers just one of many possible outcomes, as various counter-moves exist that will be able to increase the defending team's probability of capturing the ball when faced with an opponent team using the dual ram strategy.

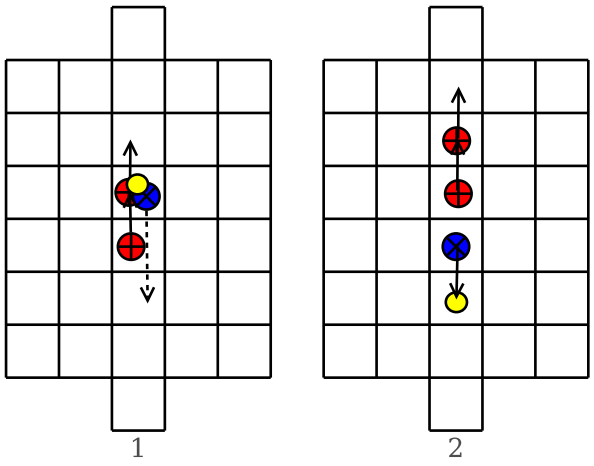


Figure 6.2: Dual ram counter strategy

One such counter strategy was visually observed, as depicted in figure 6.2. The observed counter strategy for the dual ram is fairly simple: once the defending player is on the same block as the first opposition player, the defending player has a 50% probability of capturing the ball. In the event that the defending player does capture the ball, the defending player simply kicks the ball towards the opposition team goal, effectively bypassing the second opposition player.

The dual ram strategy is a fairly straightforward offensive strategy with an easy counter-move that reduces its effectiveness. The players exhibit similar traits and no clear position or role assignment is clear for the dual ram strategy - both players are required to cooperate for the strategy to succeed, though. The required cooperation is straightforward: both players need to move while staying one square away from each

other. Fifty percent of the observed games showed a form of the dual ram strategy.

6.2.2 Goalie and Striker

The second strategy that emerged is referred to as the *goalie and striker* strategy. The name should already make it clear that this is one of the well known soccer strategies as it forms the basis of more real-life soccer games. For the goalie and striker strategy one player assumes the role of a goalie and the other player leads the striking effort against the opposing team. Figure 6.3 depicts the strategy where one player simply stays static one square in front of the goal post while the other player follows a straight path towards the opposing goal.

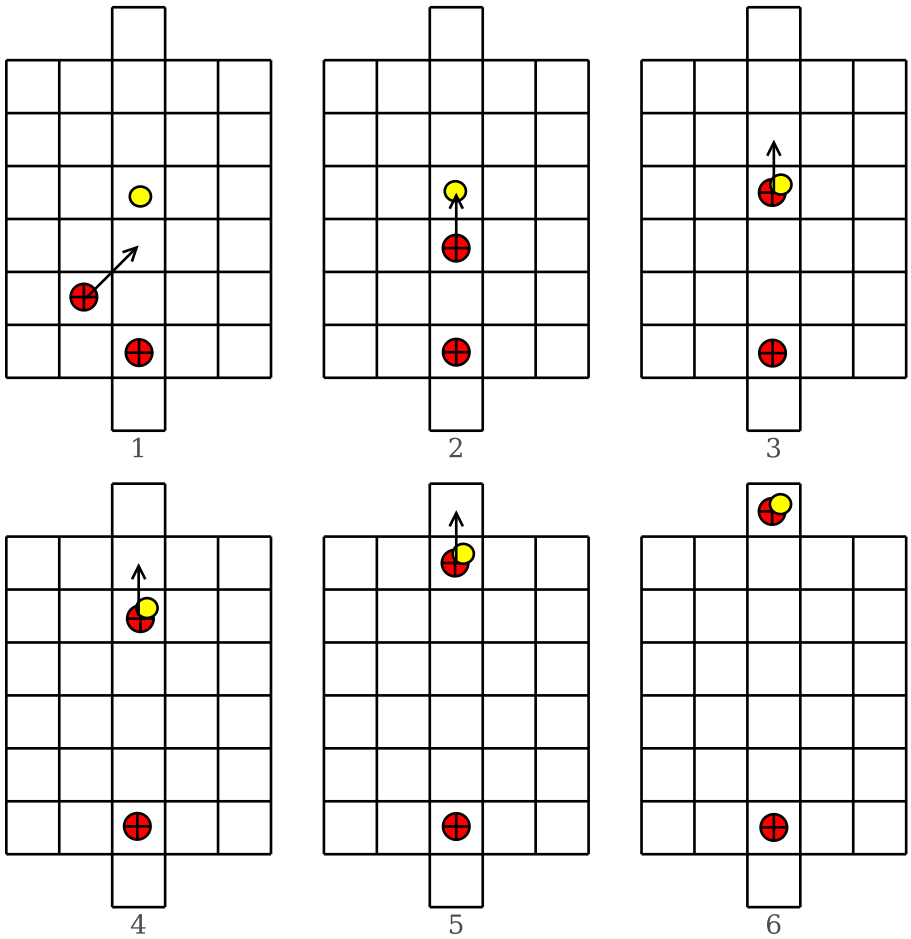


Figure 6.3: Goalie and striker offensive strategy

Should the offensive player (or striker) lose the ball possession during the offensive run, the defending player (or goalie) might still be able to prevent the opposition team from striking a goal. Figure 6.4 depicts the defensive part of the goalie and striker strategy as visually observed from the experimentation. Once an opposition player crossed the square where the goalie was located, the opposition player would have a 50% probability of losing the ball ownership to the goalie. In the case where the goalie did manage to get possession of the ball, the goalie simply kicked the ball in the direction of the opposition team's goal. At this stage the ball would be in the middle of the field and the game would typically run up to the iteration limit, leading to a draw for the match.

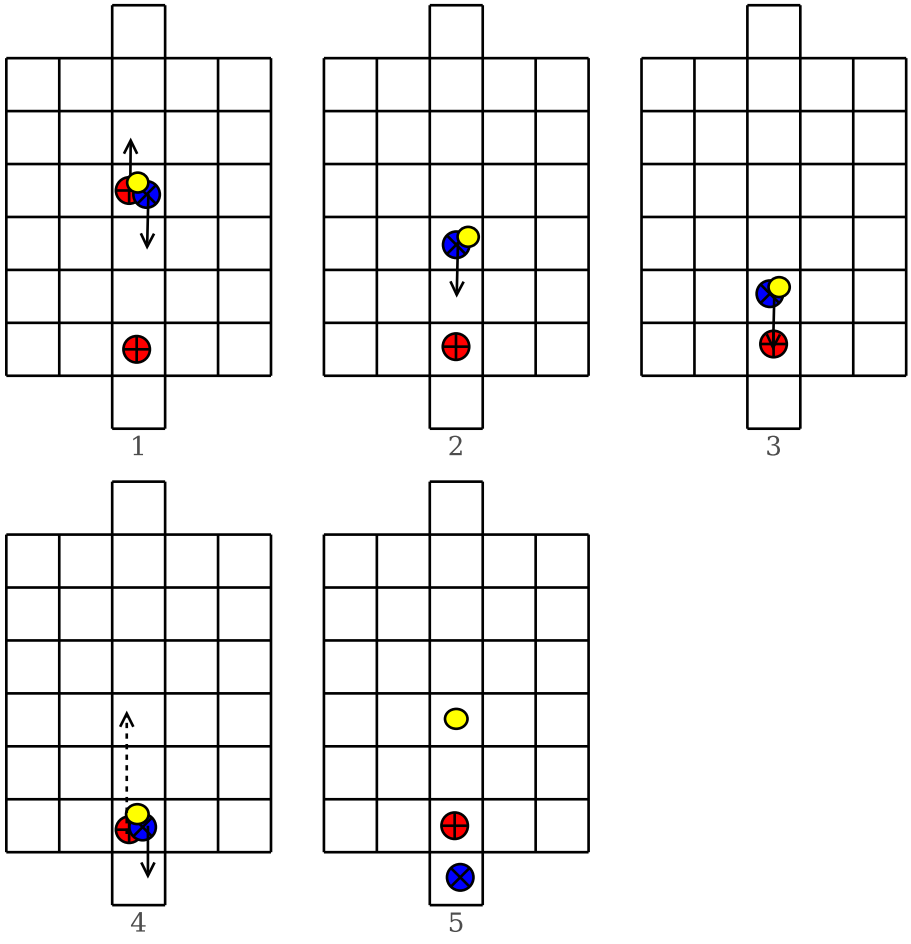


Figure 6.4: Goalie and striker defence strategy

It should be noted that, similar to the dual ram strategy described previously, the

striker and goalie strategy also attempts to reduce the opposing team's probability of scoring to less than 25% if a simple run for the goal strategy is used by the opposing team. In contrast to the dual ram strategy, the goalie and striker strategy does not make use of the goalie to continue a counter-offensive goal run in cases where ball possession was lost by the striker.

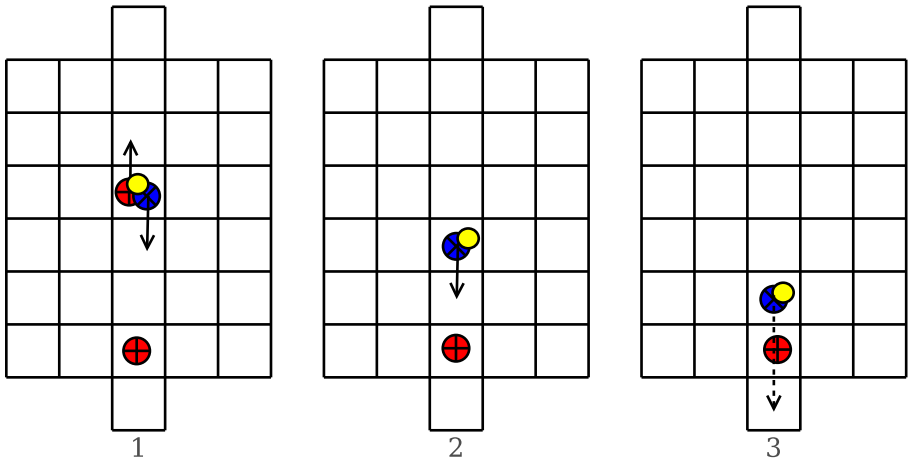


Figure 6.5: Goalie and striker counterstrategy

Similar to the dual ram strategy, the goalie and striker strategy can be outwitted by a simple counter-move. Figure 6.5 depicts one of the observed counter-moves. In the case where the opposition team manages to take possession of the ball, the opposition player will continue to move towards the goal up to the square in front of the goalie. Once this position is reached, the opposition striker kicks the ball over the goalie into the goal - effectively rendering the goalie useless as the opposition striker never crosses the square where the goalie is positioned.

The goalie and striker strategy proved effective against straightforward goal run strategies employed by the opposition team players. Players exhibited unique traits: one player took on the role of a dedicated goalie protecting the goal only, while the other player took the role of dedicated striker. Games where the goalie protected the goal typically ended in a draw as the striker did not return to take possession of the ball once the goalie kicked it away. Draw results reduced the effectiveness of this strategy as both teams taking part in the game is rewarded for a draw result. Twenty four percent of the

observed games showed a form of the goalie and striker strategy.

6.2.3 Kickaway

The third strategy that emerged is referred to as the *kickaway* strategy. For the kickaway strategy to work, a player must simply kick the ball to one of the field sides once the player has possession of the ball. Figure 6.6 depicts the basic strategy where a player effectively takes the ball out of play by kicking it away.

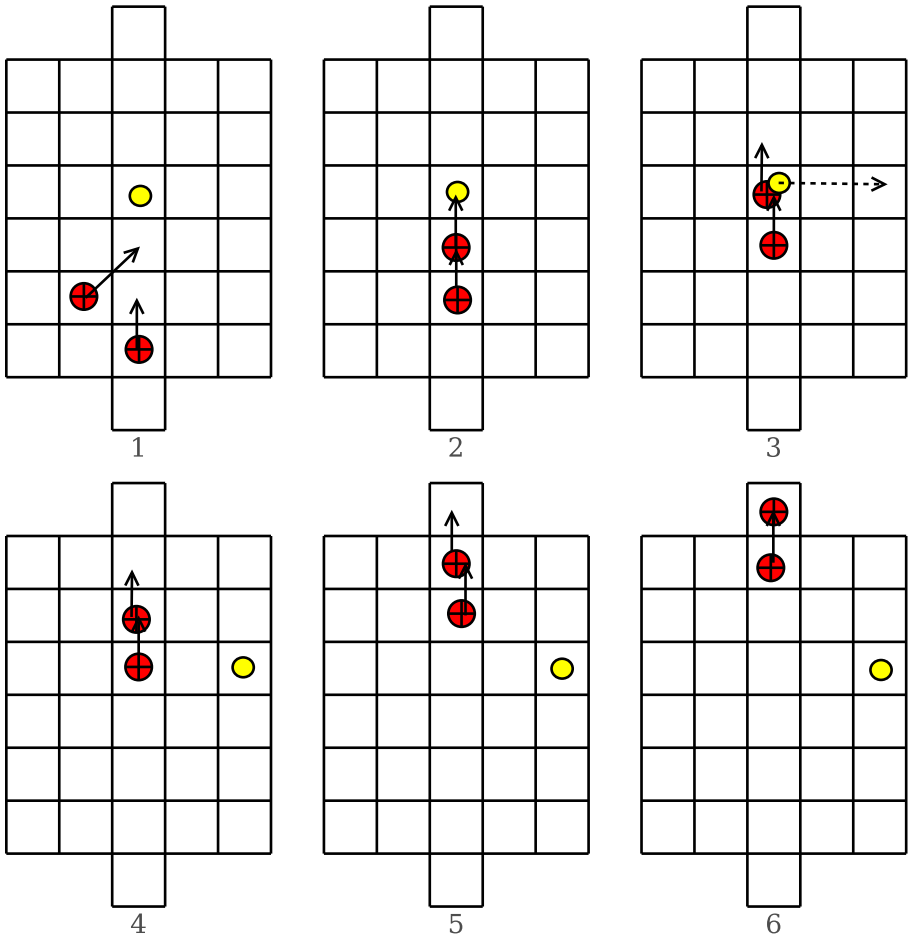


Figure 6.6: Kickaway strategy

Once the ball has been kicked away, there is typically no further noteworthy gameplay in the round and the game will run up to the iteration limit. The kickaway strategy

essentially forces a draw game result because the players do not typically find a way to return the ball into the gameplay before the iteration limit is reached.

To understand why the kickaway strategy exists, the competitive reward function was evaluated. In this case the FIFA league ranking relative fitness was used as the competitive reward function. The FIFA league ranking relative fitness function, as described in section 5.2.3, rewards players for both victory and draw results. Reward is allocated based on the ranking of the opposition players: a draw result against a high-ranking team can be rewarded more than a victory against a low-ranking team. In cases where a low ranking team competed against a high ranking team the evolutionary process found that draw results yielded a higher reward for the team being evolved than for the opposition team. Remember that a draw against a good team is rewarded more than a draw against a weak team. This property makes it possible for a weak team to overtake a stronger (or higher ranked) team in the rankings by simply drawing a number of games against a stronger team. The kickaway strategy effectively exploits this weakness in the ranking system.

A weak team can easily use the kickaway strategy to force a draw outcome against a strong team, making it an effective defensive strategy against strong opposition teams. No specific role assignment is done for the kickaway strategy and no cooperation is required between the players. The kickaway strategy can be successfully applied by a single player. Observations of hundreds of games showed that the kickaway strategy typically occurs early on in the training process whereafter a more efficient strategy is found. Even though the kickaway strategy exploits a potential weakness in the relative fitness function, observations showed that the higher reward for victory was enough to prevent the observed team from converging on this strategy. Sixteen percent of the observed games displayed the kickaway strategy after 500 iterations.

6.2.4 Kick-pass Goal

The fourth strategy that emerged is referred to as the *kick-pass goal* strategy. The kick-pass goal strategy was the most complex and rarest of all the strategies observed.

The kick-pass goal strategy is a fairly complex move involving both players cooperating by being on the right squares at precise times. The front player or striker moves

to take possession of the ball, followed closely (one square away) by the second player (referred to as the volleyer in this strategy). In the case where the striker manages to take possession of the ball, the striker immediately kicks the ball into the field corner on the opposition team's side while also moving towards the corner. The volleyer continues to move towards the opposition goal. Once the striker reaches the corner, it performs another kick: this kick places the ball on the same square as the volleyer. The volleyer simply dribbles or kicks one square forward to finish the move and to score a goal. Figure 6.7 depicts the kick-pass goal strategy as observed in the empirical analysis.

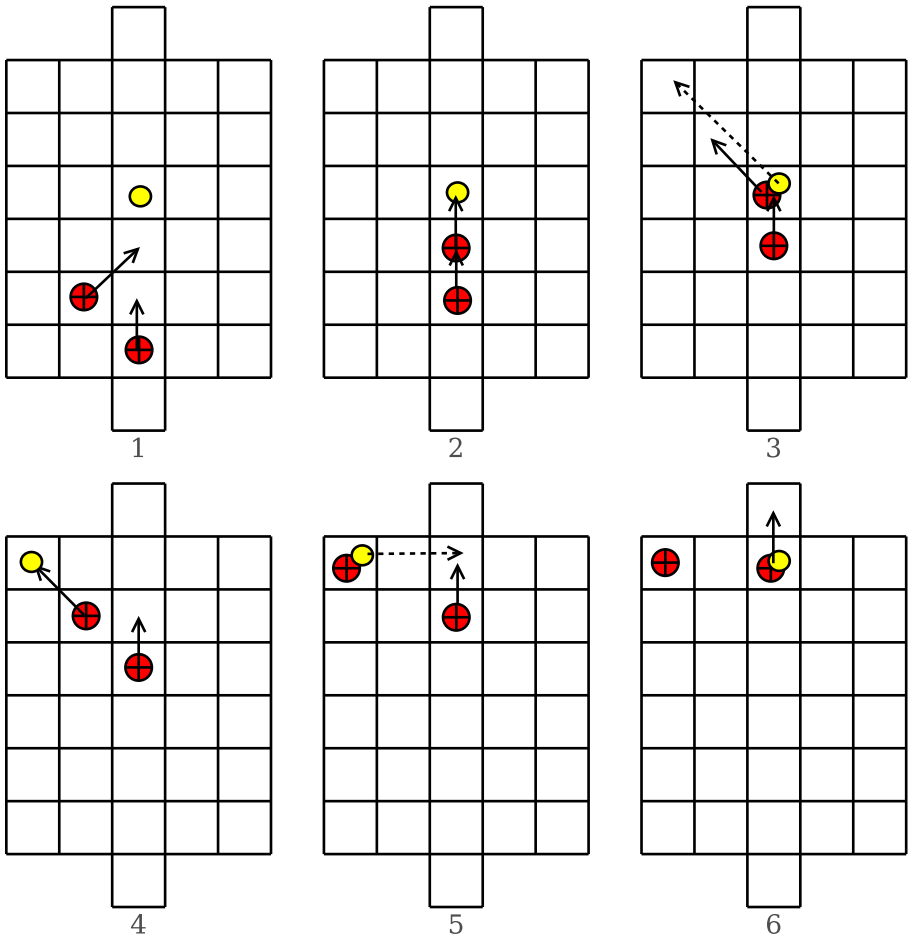


Figure 6.7: Kick pass goal strategy

It should be noted that the goalie and striker defensive strategy discussed earlier proved to be an efficient defence against the more complex kick-pass goal strategy. The

ball is kicked onto the same square as where the goalie is positioned in the goalie and striker strategy, giving the goalie an effective 50% probability of countering the move. The kick-pass goal strategy is extremely effective against the dual ram strategy, because not only is the ball prevented from being retaken by the second ramming player, but the ball is also repositioned for the final volley move to take the goal.

The kick-pass goal strategy requires precise teamwork between the two players: each player assumes a unique role with unique traits of striker and volleyer in order to succeed. Observations showed this strategy in only seven percent of the games, most likely due to the complexity of the moves required for this strategy to work and the effectiveness of the single goalie defence that can be used against this strategy. For the observed games, the kick-pass goal strategy proved to be extremely efficient against most of the counter-strategies.

6.2.5 Summary of Gameplay Strategies

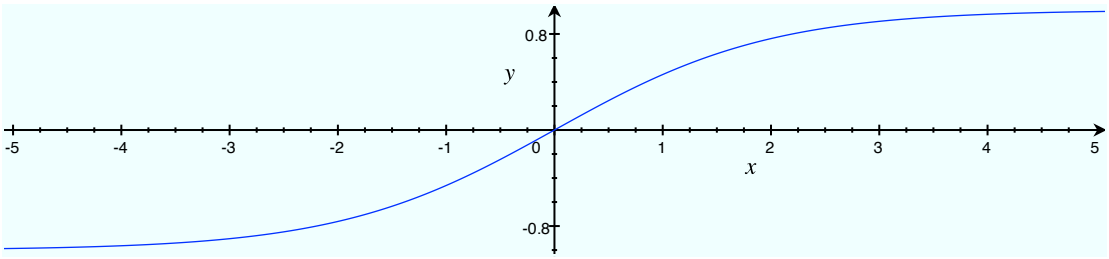
None of the strategies discussed above proved unbeatable. In each case the opposition had a number of counter-strategies that can be applied successfully to limit opposition goals being scored. The strategies that involved forms of cooperation proved more successful, leading to higher points over the course of the evolutionary process, than the strategies where no cooperation was visible. This shows that cooperation is key to higher reward even though there was no direct reward mechanism that evaluated the level of cooperation.

The next section attempts to identify why the evolved strategies seemed so simplistic and weak.

6.3 Gameplay Strategy Stagnation

Visual analysis confirmed that increasing the number of iterations to 2000 leads to no notable change in the evolved gameplay strategies. The visual analysis reinforces the hypothesis made from the discrete measurements in the previous chapter that the training stagnates. In order to identify possible reasons for the training stagnation, existing literature on PSO training of neural networks were consulted.

Van Wyk and Engelbrecht demonstrated that the use of a bounded activation function can lead to stagnation during the training of neural networks using particle swarm optimisation [158]. Van Wyk and Engelbrecht found that stagnation occurs once the weights (particle positions) move too far away from the active region of the neural network activation function. The further away from the active region the particle's position values move, the less significant any changes to the output of a neuron become, irrespective of how large the changes are to the net input signal of the neuron. Since the output of the neural net does not change the personal best positions of the particles, and subsequently, the global best position of the swarm no longer changes. Graph 6.1 demonstrates the problem visually. From the graph it is clear that the function value no longer increases a significant amount once the input exceeds $+3$ on the positive side and -3 on the negative side.



Graph 6.1: Hyperbolic tangent activation function.

In order to investigate if the stagnation occurs due to saturation of the neural network weight vector as postulated by Van Wyk and Engelbrecht, histograms of the neural network weights were plotted. Graphs 6.2 and 6.3 depict the histograms obtained from 30 independent samples using the optimised parameter configuration from the previous chapter.

Visual analysis of the histograms reveals that the weights move away from the more active region of the activation function, i.e. the area around 0. This happens in as few as two iterations from the start of the algorithm. As the number of iterations increases, the number of weights that leave the active region of the activation function also increases, which in turn further degrades the training performance. Also note that the histograms show a large number of weights in the area around zero; this is an indicator that the

neural network size is not undersized and thus not the reason for the bad-performing players.

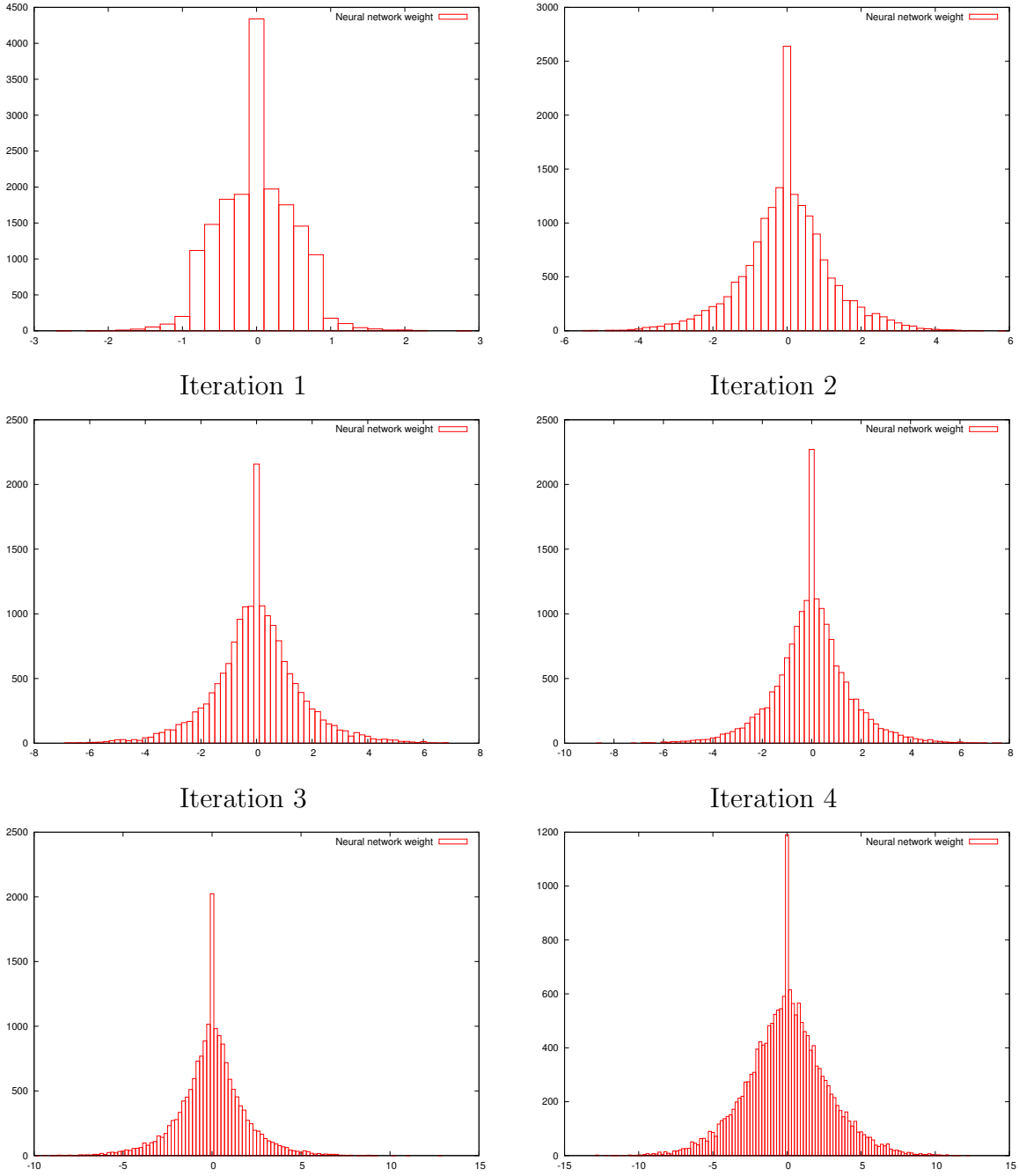
The histograms presented here strengthen the hypothesis that the training algorithm saturates the neural network weights causing the training process to stagnate. The next chapter investigates possible ways to address the saturation problem.

6.4 Summary

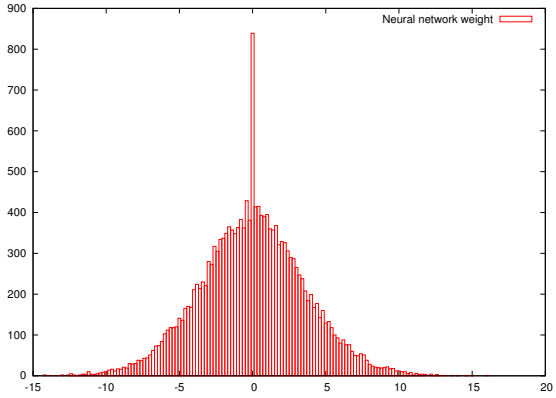
The objective of this chapter was to extend the analysis performed in the previous chapter by visually analysing the evolved gameplay strategies. Various gameplay strategies were identified and recurring strategies were classified accordingly. It was found that the strategies that did evolve were fairly weak and simplistic, even though basic forms of team cooperation could already be noted in the results.

It was discovered that the weak gameplay strategies were at least partly due to stagnation, with the training algorithm being unable to improve further upon the global best position. It was hypothesised that this is the result of the particles leaving the active regions of the neural network activation function. Histograms were drawn to analyse the neural network weight distribution, and confirmed that the weights indeed left the active region of the activation function.

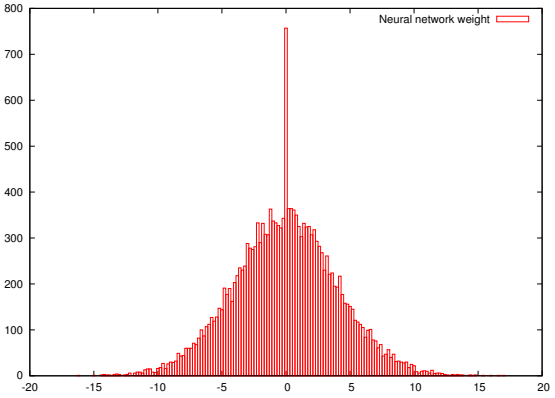
The next chapter continues on the work done in this chapter by further investigating the saturation hypothesis and identifying ways to improve the performance of the new algorithm.



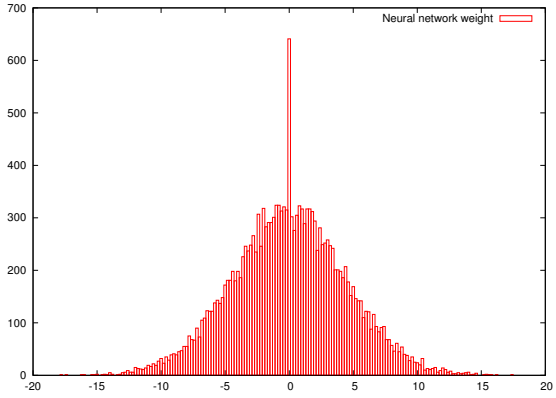
Graph 6.2: Neural network weight histograms for 30 independent samples using the optimised parameter configuration.



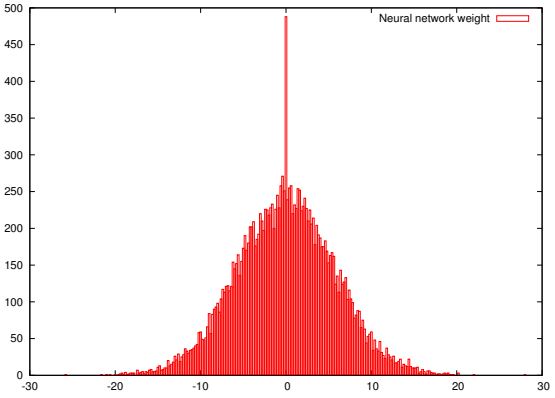
Iteration 50



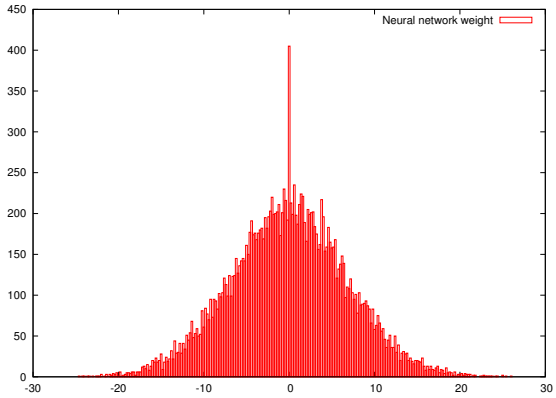
Iteration 75



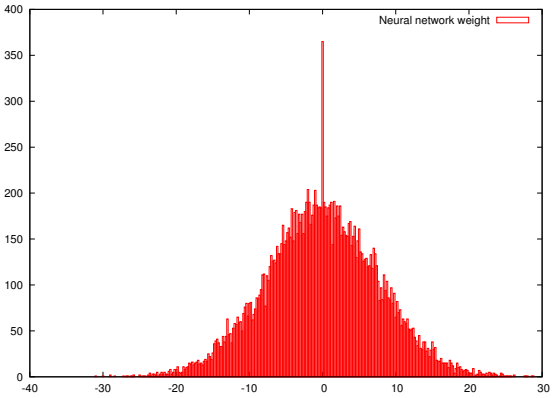
Iteration 100



Iteration 200



Iteration 300



Iteration 400

Graph 6.3: Neural network weight histograms for 30 independent samples using the optimised parameter configuration.

Chapter 7

Performance Improvements

"One must change one's tactics every ten years if one wishes to maintain one's superiority."

Napoleon Bonaparte (1769 - 1821)

The previous chapter showed that the CCPSO training algorithm as presented in chapter 4 led to weak gameplay strategies. Initial investigations as summarised in chapter 6 showed that the problem is likely be due to weights moving away from the more active region of the neural network activation function. The objectives of this chapter are:

- to lessen the weight saturation problem and thus improve the evolved strategies, and
- to perform an in-depth analysis of the trained player performance after applying the improvements to the algorithm.

An in-depth analysis of this stagnation problem is performed and new performance measurements are taken as part of this analysis to provide additional insight into why additional training iterations fail to improve the performance of the evolved agents. The resulting playing strategies are presented and player behaviours and gameplay strategies are analysed.

7.1 Introduction

The previous three chapters demonstrated that particle swarms could be used in a coevolutionary manner to train soccer-playing neuro-controllers. The results proved promising with the neuro-controlled soccer players learning basic gameplay strategies. However, the algorithm failed to evolve complex gameplay strategies. It was also shown that the training algorithm stagnates and no significant improvement can be achieved by increasing the number of training iterations. This chapter focusses on improving the training algorithm in order to evolve more complex gameplay strategies. A complete analysis of the improved performance is also presented.

Section 7.2 investigates the detail of how particle swarm-based training stagnates due to the neural network weights saturation and proposes solutions to the problem. Section 7.3 presents the bounded personal best particle swarm as a possible solution to the saturation problem. Section 7.4 presents a modification to the charged PSO to further improve convergence. Section 7.5 presents a behavioural analysis of the global best position for each swarm, followed by section 7.6 where the behaviour of each player is analysed. Section 7.7 presents an analysis of the new gameplay strategies. Finally, section 7.8 analyses the trained players using discrete measurements to gauge the actual gameplay performance of each team and player.

7.2 Neural Network Weight Saturation

In order to prevent the neural network weights from becoming saturated, the global best particle position must be prevented from leaving the active region of the activation function. For a hyperbolic tangent activation function, as shown in graph 6.1, the particle must not leave the region $(-5, 5)$ as any value beyond that will have no significant effect on the activation value.

Particle position changes in the charged PSO are driven by the velocity and position update equations. Four components guide the direction and magnitude of step sizes, namely the momentum $(\vec{v}_i(t-1))$, the cognitive component $(\vec{\rho}_1(\vec{x}_{pbest_i} - \vec{x}_i(t)))$, the social component $(\vec{\rho}_2(\vec{x}_{gbest} - \vec{x}_i(t)))$, and the charged acceleration $(\vec{a}_i(t))$. Only half the particles in the swarm will have a non-zero acceleration due to an atomic swarm being

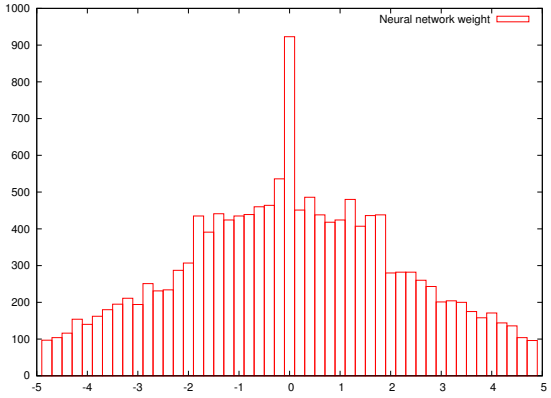
used, i.e. $\vec{a}_i(t) \neq 0$. Since the momentum is a factor of the previous iterations' velocity, the momentum does not dictate any new directional forces onto the particle, changes in the direction of position updates are thus only introduced through the cognitive, social, and acceleration components or, more specifically, the $\vec{x}_{pbest_i}$, $\vec{x}_{gbest_i}$ and $\vec{a_i}(t)$ vectors. Also take note that the global best particle, $\vec{x}_{gbest_i}$, is simply the highest-ranked personal best particle's position, $\vec{x}_{pbest_i}$.

In an attempt to remove saturation of the neural network weights, a clamping restriction is placed on the value of the $\vec{x}_{pbest_i}$. A particle may update the personal best position only if the current position is within the bounded area and if the current position represents a better fitness. The CCPSO algorithm with the $(-5, 5)$ clamping condition on $\vec{x}_{pbest_i}$ is referred to as the bounded CCPSO algorithm. The effectiveness of this new clamping operation is investigated in the next section.

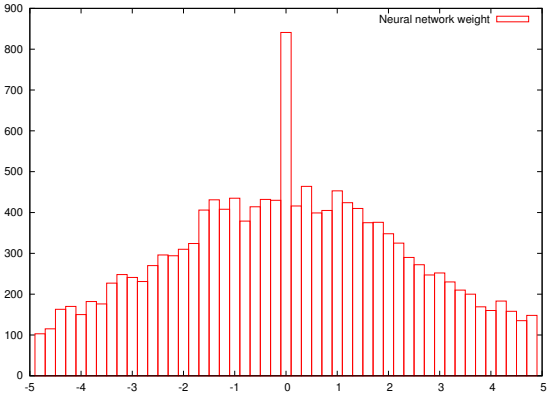
7.3 Bounded personal best performance

Histograms were again drawn for the neural network weights after repeating the training using the bounded CCPSO algorithm. The training was done using the previously determined optimised parameter configuration. Graphs 7.1 and 7.2 depict the obtained results. The histograms clearly show that the newly added personal best position clamping prevents the weights from leaving the more active region of the activation function. Based on these results, the clamping should prevent the training algorithm from stagnating as shown by Van Wyk and Engelbrecht [158].

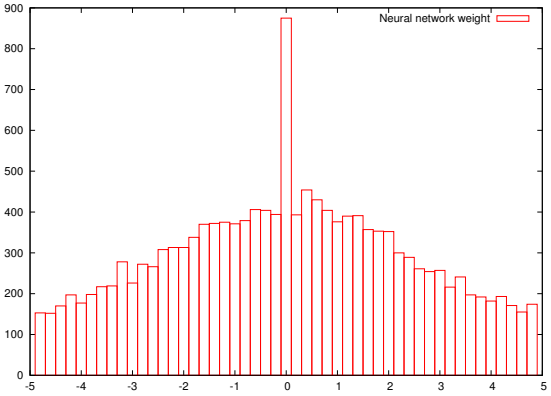
The effect of the personal best position clamping can also be seen on the swarm diversity. Graph 7.3 provides a comparison of the swarm diversity between the CCPSO algorithm presented in the previous chapter and the bounded CCPSO algorithm described in the previous section. Swarm diversity is calculated as the degree of dispersion of particles and is formally defined by Krink *et al.* in [96]. The swarm diversity for the bounded CCPSO algorithm decreases steadily up to iteration 300 from where it stabilises. Both algorithms stabilise on high swarm diversities. A high swarm diversity is expected as the algorithm makes use of the charged PSO. As discussed in section 2.4.5, the charged swarm inherently maintains a high diversity to facilitate exploration of the



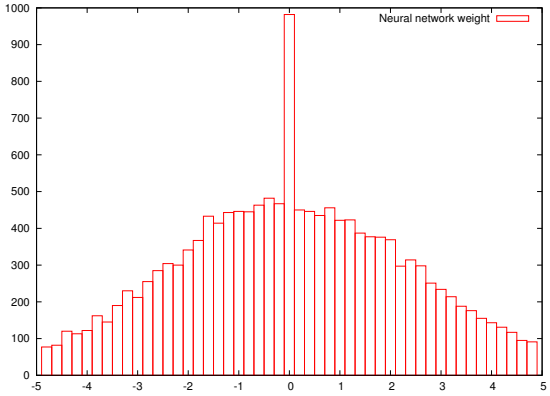
Iteration 1



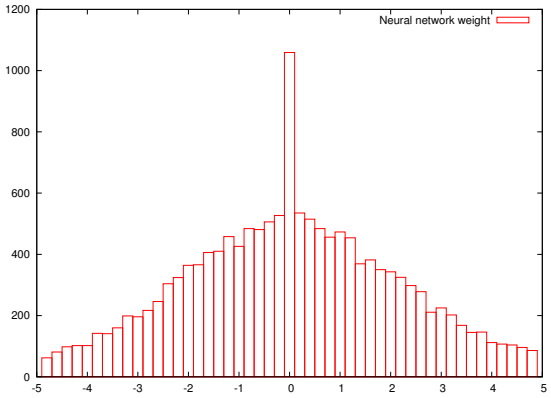
Iteration 2



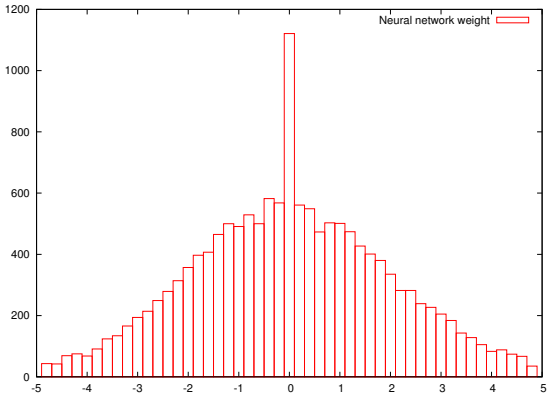
Iteration 3



Iteration 4

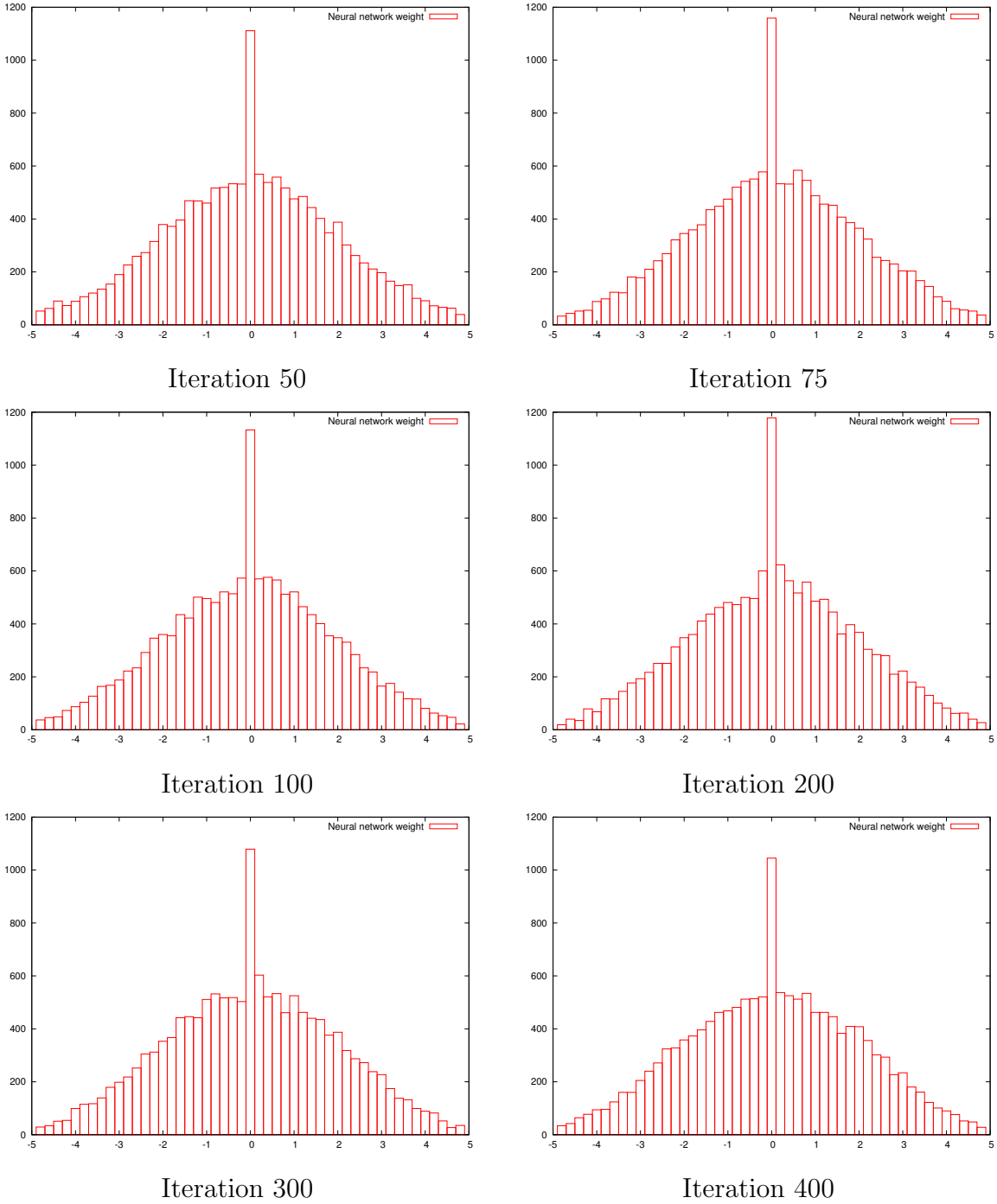


Iteration 10

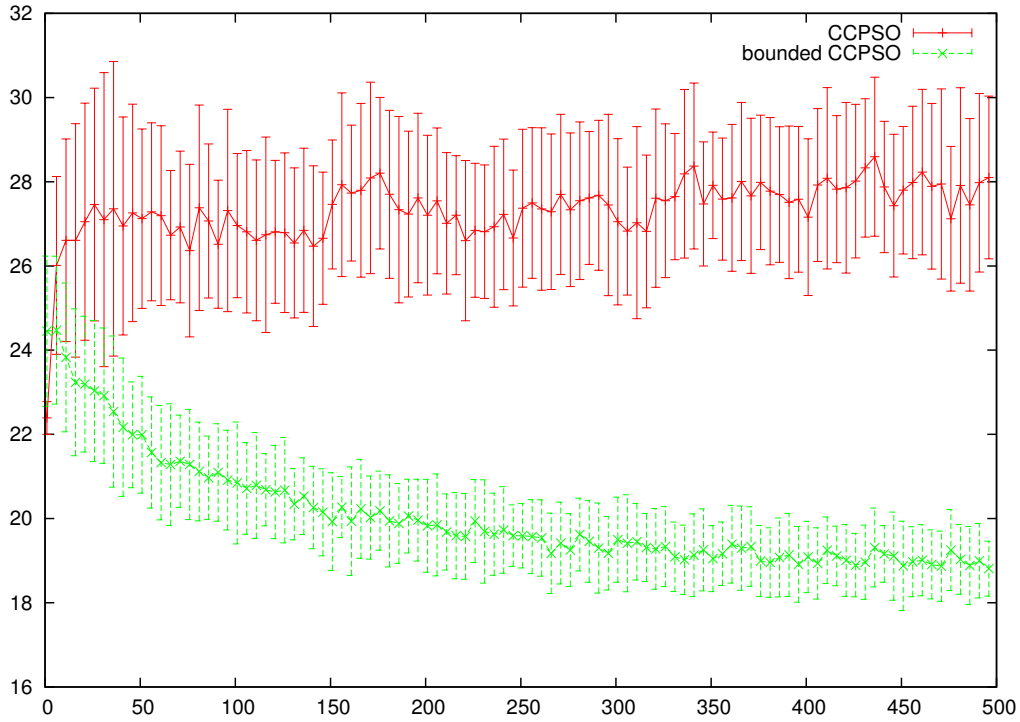


Iteration 25

Graph 7.1: Neural network weight histograms for 30 independent samples using the bounded CCPSO with the optimised parameter configuration.



Graph 7.2: Neural network weight histograms for 30 independent samples using the bounded CCPSO with the optimised parameter configuration.



Graph 7.3: Swarm diversity using the CCPSO algorithm in comparison with the bounded CCPSO algorithm

search space, allowing new better solutions to be found.

While exploration of the search space is desirable early on in the training process, the importance of exploitation increases towards the end of the training process. The next section describes how the added exploration of the charged PSO can be removed to focus only on exploitation.

7.4 Improving Convergence onto a Gameplay Strategy

The previous section showed that high diversity is maintained throughout the training process. This high diversity is due to the charged particles in the swarm maintaining high diversity in order to facilitate continued exploration of the search space. Towards

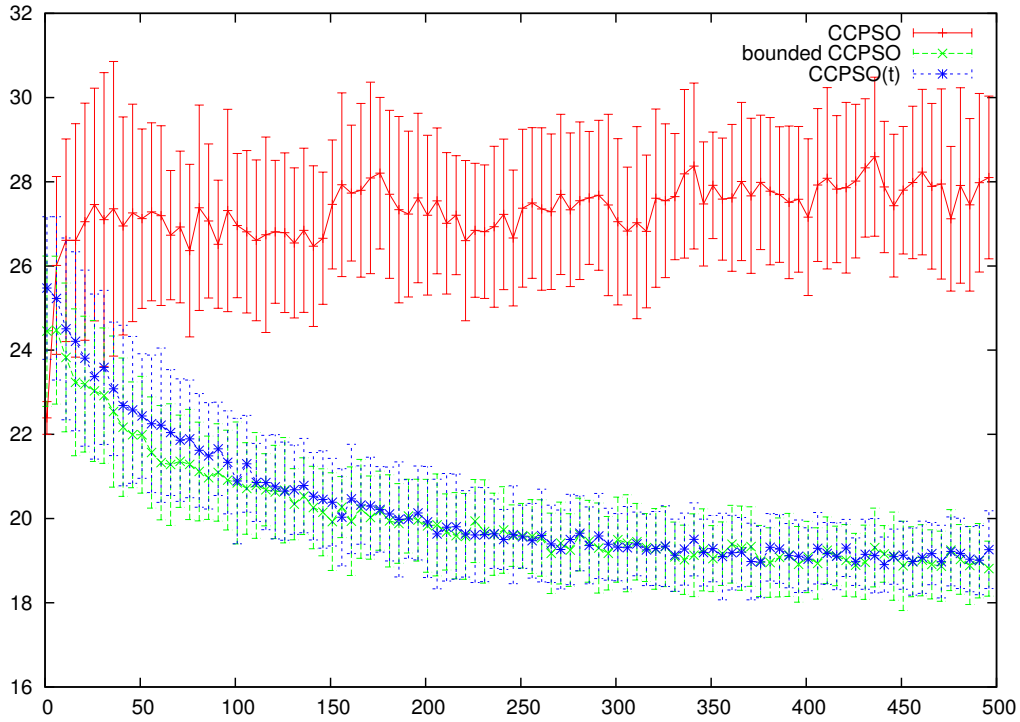
the end of the training process it becomes desirable to remove the extra exploration introduced by the charged particles to focus only on exploitation of the already found solution(s). A solution where the effect of the charged component of the charged PSO is reduced to zero after a specified number of iterations is needed.

It was shown in section 2.4.5 that the charged PSO makes use of a modified velocity update equation as given in equation (2.12). The new acceleration term introduced a repulsion force between two particles based on the perception limit, R_{plimit}. If the distance between two particles are greater than R_{plimit} the new acceleration term, $\vec{a}_i(t)$ becomes 0. If $\vec{a}_i(t) = 0$, then the velocity update is the standard velocity update of equation (2.7).

By introducing a time-varying perception limit, $R_{plimit}(t)$, that decreases to zero over a given number of iterations, the charged PSO moves towards a standard PSO. This modification to the algorithm was implemented as a linear decreasing function where the value of $R_{plimit}(t)$ was decreased linearly to zero over 250 iterations. The choice of 250 iterations is based on the fact that the algorithm showed stagnation from around 300 iterations.

Graph 7.4 depicts the diversity of the swarm for the new variant of the algorithm, referred to as *CCPSO(t)*, compared to the CCPSO algorithm and the bounded CCPSO algorithm. The graph shows that no notable decrease in swarm diversity have been obtained compared to the bounded CCPSO. The bounded CCPSO and *CCPSO(t)* algorithms follow almost identical swarm diversity trends, displaying similar variances in performance as well. An initial drop in diversity is notably visible over the first 300 iterations after which the swarm diversity stabilised.

The presented results show that the high swarm diversity, after iteration 250, can not be attributed to the use of the charged PSO algorithm, as the algorithm effectively changed into a standard PSO and still maintained the high diversity. The next section investigates the behaviour of the global best position to gain more insight into why the algorithm maintains a high diversity.



Graph 7.4: Swarm diversity using CCPSO(t) in comparison with the original and bounded personal best algorithms

7.5 Behavioural Analysis of the Global Best Position

The initial high diversity in a swarm is indicative of the exploration phase of the search algorithm. As better and better solutions are found, the swarm of non-charged particles converges onto these better areas of the search space. The convergence is a process driven by the shared $\vec{x}_{gbest}$ component of the PSO velocity update equation [153]. The search process leads to a behaviour where the global best position jumps around in large steps right after the swarm is initialised. As the search process progress and a smaller area of the search space is exploited the jumps in the global best position becomes smaller. Obviously, should a faraway solution be found that presents a better solution, the global best position moves in a single large jump to that new location.

As previously noted, the problem investigated by this work is not a static, predictable one, but is rather classified as a dynamic environment problem in which the search space

is constantly changing. For a dynamic environment, it is expected that the global best position changes more often than for a static problem. However, in the case of the simple soccer problem investigated in this work, convergence on specific gameplay strategies is hoped for. It is expected that changes in the global best position decrease to extremely small jumps as a player/swarm exploits a gameplay strategy.

In order to investigate the behaviour of the global best position, a new measurement was implemented to track the magnitude of the jumps in the position of the global best between consecutive iterations. The global best movement measure, $\Phi(t)$, is defined as

$$\Phi(t) = \sum_{j=1}^{J} |\vec{x}_{gbest,j}(t) - \vec{x}_{gbest,j}(t-1)| \tag{7.1}$$

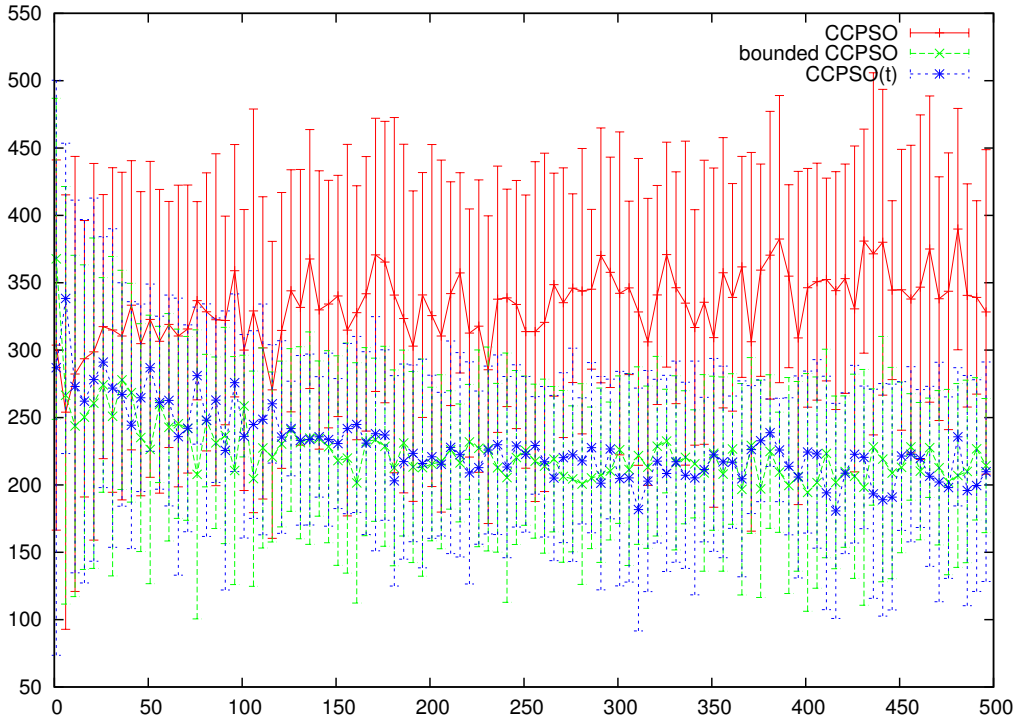
This new measure simply adds the magnitude of the change in each dimension together. Small values indicate a small change in position, whereas large values indicate a larger change in position. The measurement will give an indication of the magnitude of a change in position.

Graph 7.5 depicts the results obtained when measuring $\Phi(t)$ for each of the three algorithms previously presented. The graph indicates fairly chaotic behaviour in the swarms, with the global best position changing continuously by making fairly large jumps. A high standard deviation among the 30 evaluated simulations is also noted.

For the *CCPSO* algorithm, the jumps in global best position increase slightly with increase in the number of iterations. The *CCPSO* algorithm also maintained the largest jumps in global best positions when compared to the other two algorithms. This can be ascribed to the fact that the two enhanced algorithms only search in a bounded search space.

Both the bounded CCPSO and the *CCPSO(t)* algorithms exhibited similar behaviour with reference to $\Phi(t)$. A slight decrease in the size of the jumps can be noted up to around iteration 200, after which the jump size stabilised. The average jump size still exceeded 200 after 500 iterations, with a noticeably high standard deviation among the 30 simulations evaluated, but less than that of the *CCPSO*.

The observed behaviour contradicts the expected behaviour that the global best position jumps would become smaller as the search progresses. The observer behaviour clearly indicates that none of the three algorithms lead to any form of convergence. The



Graph 7.5: Measured Φ using CCPSO(t) in comparison with the CCPSO and bounded CCPSO algorithm

two enhanced versions of the algorithm had slightly better performance when evaluating only $\Phi(t)$.

From the behaviour analysis it is clear that the global best position still changes position quite dramatically, and no single solution is converged onto.

The observed jumping behaviour might be explained by evaluating the following hypothesis: If multiple gameplay strategies are found by each swarm, the optimal strategy, represented by the global best position, will move between these strategies as no one strategy proves superior to other gameplay strategies. An offensive strategy might prove superior to a defensive strategy against an extremely aggressive opposition team. In other cases, however, the defensive strategy might prove superior or an alternative attack strategy might exploit a weakness against an opposing team leading to a win. The optimal and best performing strategy will thus vary between iterations with no single optimal strategy being superior.

Multiple gameplay strategies are present in the search space as different optimal areas (or peaks in the hyper-dimensional search space). If multiple strategies (solutions) exist, particles will "group" around these optimal areas forming clusters of particles, with each cluster effectively converging on its own optimal position. The behaviour observed in the previous section can then be explained as the global best position changing position between different clusters. Because each cluster is spaced noticeably far away from the others, the jumps between the clusters are clearly visible in the graphs.

In order to evaluate the clustering hypothesis, a clustering algorithm was applied to cluster the global best positions found for each iteration that the algorithm ran. To prevent previously identified neural network weight saturation and stagnation problems from influencing the clustering results, only the *CCPSO(t)* training algorithm is considered.

The X-means clustering algorithm extends the K-means clustering algorithm with efficient estimation of the optimal number of clusters [121]. X-means allows for data to be clustered without knowing beforehand how many clusters exist. X-means also attempts to address the computational scalability shortcoming of the K-means algorithm when dealing with larger datasets. The X-means clustering algorithm was applied to the global best positions for each of the swarms separately. This resulted in four sets of clusters (one set for each player position).

The X-means algorithm revealed that clusters were indeed formed in the swarms. The number of clusters found per swarm varied between two and five. In effect, this indicates that each swarm, or game agent learned between two to five different gameplay strategies. The next section explores the different strategies that were learned. It should be noted that two clusters may represent exactly the same behaviour, because equivalent neural networks can be obtained by simply swapping hidden layer neurons. The resulting neural networks, after swapping hidden neurons, have exactly the same behaviour.

To confirm the proposed hypothesis, each cluster is analysed and the gameplay strategy represented by each cluster is identified. To achieve this goal, agents were trained with the *CCPSO(t)* training algorithm and clusters were computed for each of the four swarms that represented the trained players. Each cluster centroid was taken as the playing strategy of the player that corresponds to that swarm. Each of these candidate

players was then visually analysed when playing against every other candidate player.

7.6 Player Strategy Analysis

Each centroid was analysed independently against every other centroid. Figure 7.1 labels player starting positions for each analysis. The observed behaviour of each cluster centroid is discussed in this section. Analysis of the combined effect of the observed behaviours is provided in section 7.7. It should be noted that when the player behaviours are viewed in isolation, the behaviours described below can be seen as undesirable or even counter-productive, though the combined effect and interaction of these behaviours sheds light on how these behaviours evolved. The notation used, $X_y(z)$, indicates team X, position y, and centroid number z.

7.6.1 Player A_1

Four centroids were identified for player A_1. Each centroid's behaviour is discussed below.

Centroid 1: The first complex gameplay behaviour observed was ball chasing behaviour. When the ball is kicked over the block player A_1 occupies, player A_1 moves in the direction of the ball to gain ball ownership. Once player A_1 has the ball, the ball is kicked in the direction of the opposition team's goal. This chasing behaviour is repeated until a goal is scored. Figure 7.2 depicts a recorded simulation where player player A_1 recovers the ball after the ball is kicked over player A_1. For illustrative purposes, only the applicable players are shown in the figure. Chasing behaviour was limited to forward and backward movement. For simulations where the ball was kicked sideways, player A_1 did not chase the ball. One of the recorded simulations showed player A_1 missing the opportunity to kick a goal while positioned perfectly for a goal kick, two blocks away from the opponents goal. Instead, player A_1 kicked the ball in a left forward direction into the corner. No other players were able to recover the ball after the missed kick, resulting in a draw.

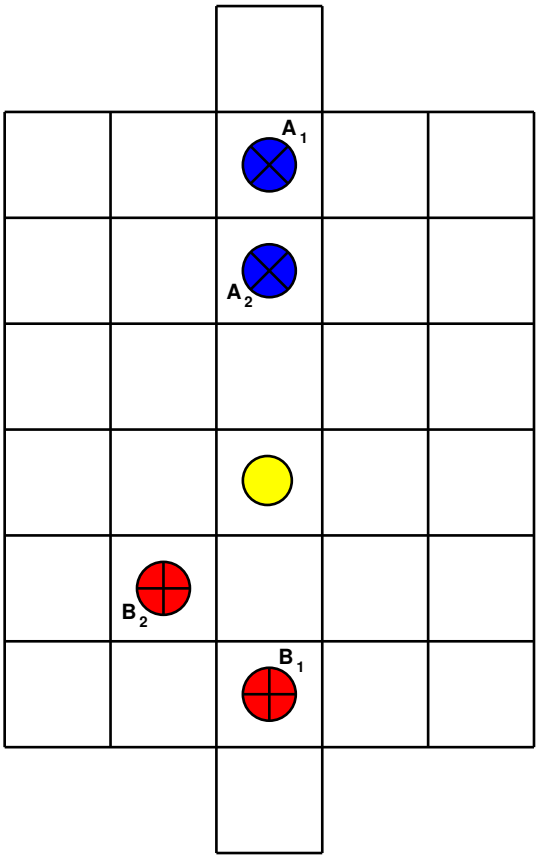


Figure 7.1: Simple soccer player positions

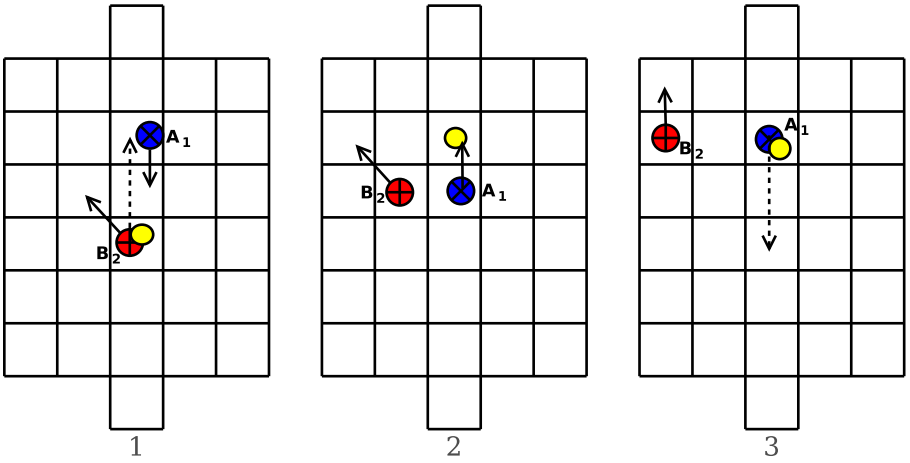


Figure 7.2: Player $A_1(1)$ demonstrating ball fetching behaviour

Centroid 2: Basic forward and backward ball chasing behaviour was observed in some games. However, it was not as successful as that of centroid 1. Sideways kicking behaviour was also observed, where player A_1 kicked the ball to the left and continued to move forward. In another simulation, as depicted in figure 7.3, player A_1 moved sideways one block and continued on to the opposing goal without ball ownership. The opponent managed to successfully score a goal in this example.

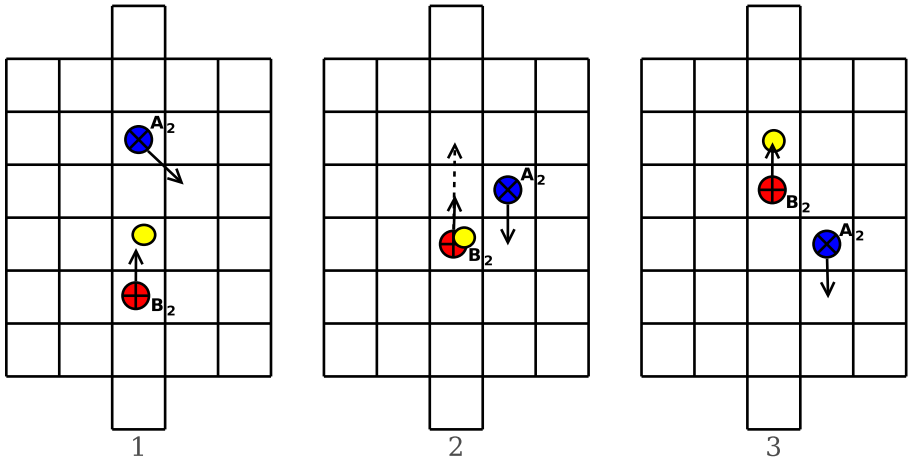


Figure 7.3: Player $A_1(2)$ demonstrating ball-evasion behaviour

Centroids 3 to 4: Basic forward goal-running behaviour was observed. In cases where an opponent player obstructed the forward path a kick was used to pass the ball over the opponent player. No forward, backward, or sideways ball-chasing behaviour was observed. This was the most simplistic playing behaviour observed.

7.6.2 Player A_2

Two centroids were identified for player A_2, each centroid's behaviour is discussed below.

Centroid 1: The player moves, without the ball, to the centre line of the field and then proceeded in the direction of the opposition goal. In cases where the ball ownership was gained a constant rule applied: If the opposition goal is three blocks away, kick to the left, or if the opposition goal is four blocks away, kick the the right. When viewed

in isolation, this kicking behaviour does not look sensible. The next section shows that this behaviour forms part of a more complex strategy. Once the sideways kick has been completed, the player continues to move forward towards the opposition goal. Figures 7.4 and 7.5 depict this behaviour.

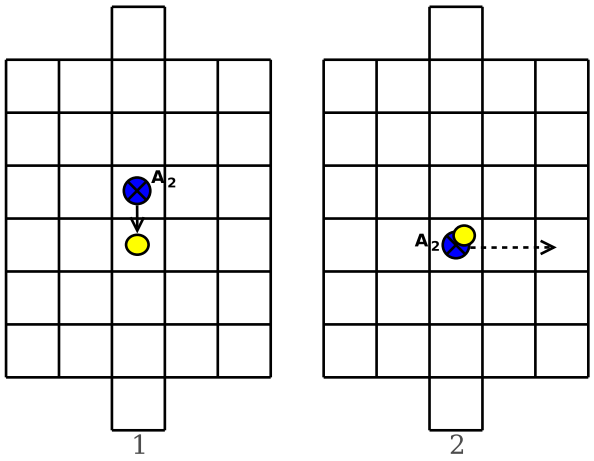


Figure 7.4: Sideways kick (scenario 1)

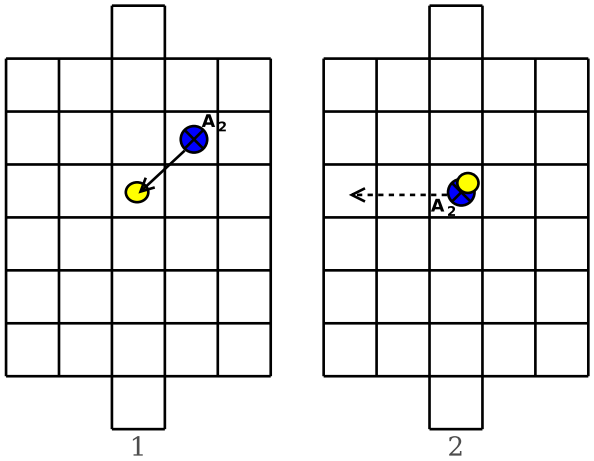


Figure 7.5: Sideways kick (scenario 2)

Centroid 2: A simplistic behaviour was observed for centroid 2, where player A_2 simply moved to the right of the field, irrespective of the ball position. No interaction with any

other game element was observed. Once the side of the field was reached, the player remained there for the remainder of the round.

7.6.3 Player B_1

Four centroids were identified for player B_1, and all the centroids for player B_1 exhibited similar behaviour.

Centroid 1 to 4: Player B_1 exhibited a forward-moving ball-kicking behaviour. In cases where opposition players blocked the way forward, player B_1 kicked the ball over them. Once player B_1 recovered the ball, a kick into the goal is performed. Even though this is a fairly simplistic gameplay tactic, visual analysis showed it to be quite a successful strategy where the outcome is often determined by the ball's random starting position. Figure 7.6 depicts a typical game where this simplistic tactic leads to a goal: first the ball ownership is acquired, then the ball is kicked over the opposition player. Finally, once player B_1 is in position, a kick is performed to score the goal.

7.6.4 Player B_2

Four centroids were identified for player B_2, each centroid's behaviour is discussed below.

Centroid 1: Player B_2 simply moves sideways to the left of the board. Once the corner was reached, no further actions were taken and player B_2 was effectively no longer taking part in the game. No ball chasing behaviour or forward movement was observed.

Centroid 2: An interesting looping behaviour was observed for player B_2, where the player moves one block forward to the left, and then backwards one block to the right. This looping behaviour led to an interesting catch of the ball, as depicted in figure 7.7, where the ball was caught after the opposition player kicked the ball to a empty block. Once the player has ball ownership, a kick is performed in the direction of the opposition team's goal. In one recorded simulation this resulted in a pass where the ball was effectively placed in position for player B_1 to perform a goal kick. Although the looping behaviour overshadowed most of this players' other movements, the actual

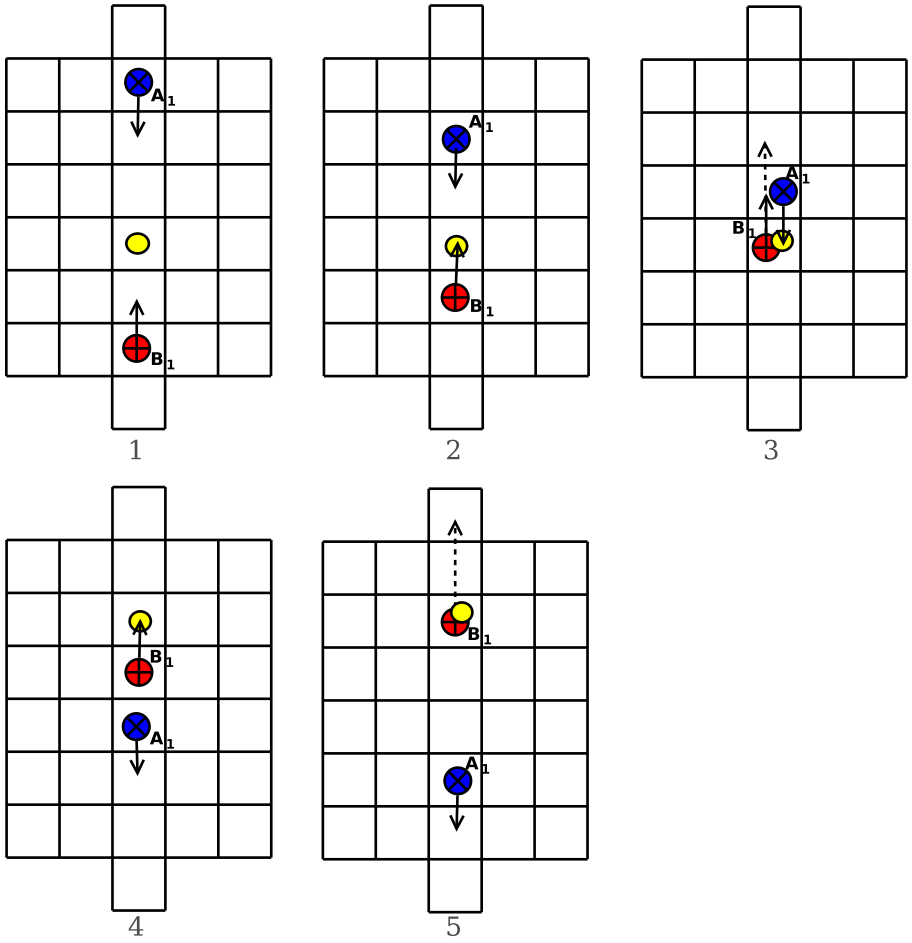


Figure 7.6: Player $B_1(1)$ scores a goal

position of where on the field the player looped is at least partially linked to the position of player B_1. In another recorded simulation the player looped between two blocks right in front of the opposition goal. In this case the ball was lost to all other players and the game effectively stalled with no other players moving on the board. The final sideways movement depicted in figure 7.7 is due to player B_1 (not shown in the figure) moving closer to the opposition goal.

Centroid 3: The player only moves in a forward and left direction. Once the edge of the board is reached, the player continues to move forward. Ball-kicking behaviour is not observed, except in a very special case: if the player has ball ownership in the top

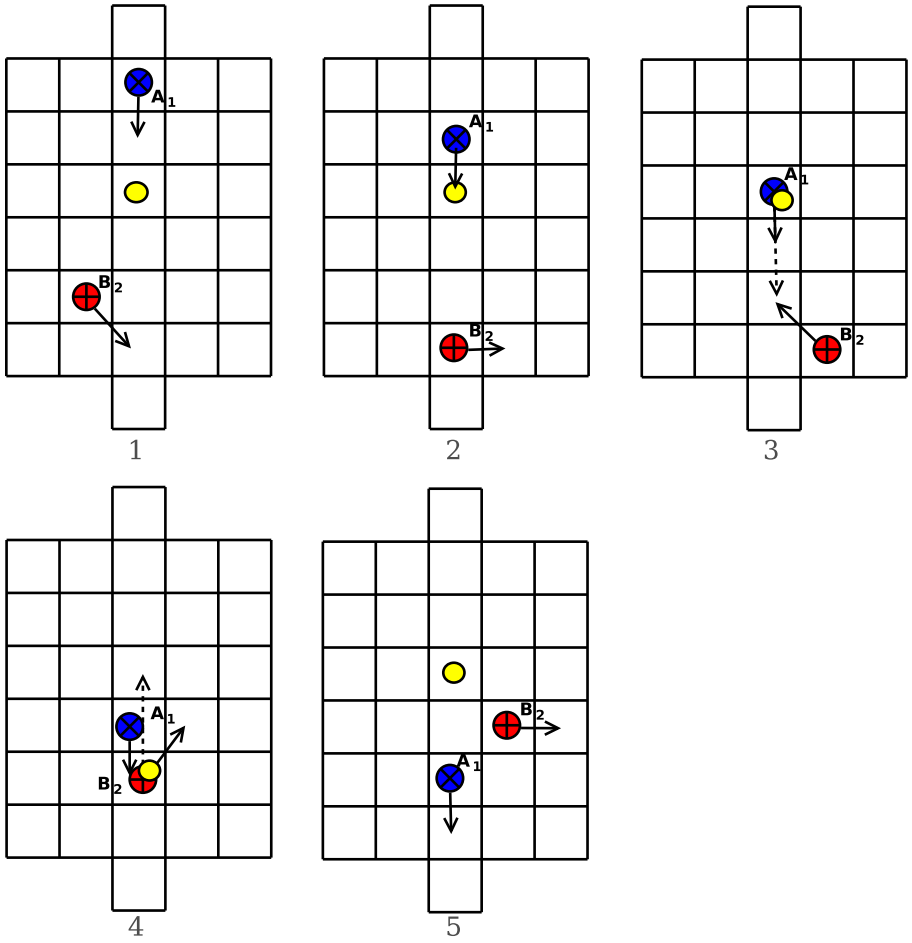


Figure 7.7: Player $B_2(2)$ catches the ball

left corner of the board, a right forward kick is performed that results in a goal being scored. This behaviour is shown in figure 7.8. It should be noted that player B_2 is not able to fetch the ball. Another player is responsible for passing the ball to player B_2 in order for this move to be performed successfully. In simulations where the ball was not passed to player B_2, player B_2 does not form part of the game.

Centroid 4: This player exhibited the most complex behaviour of all the behaviours analysed: sideway movement, looped movement, chasing behaviour, and opponent avoidance behaviour were observed. Figure 7.9 depicts a simulation where player B_2 moved away from the goal only to return just in time to prevent the opposition team from

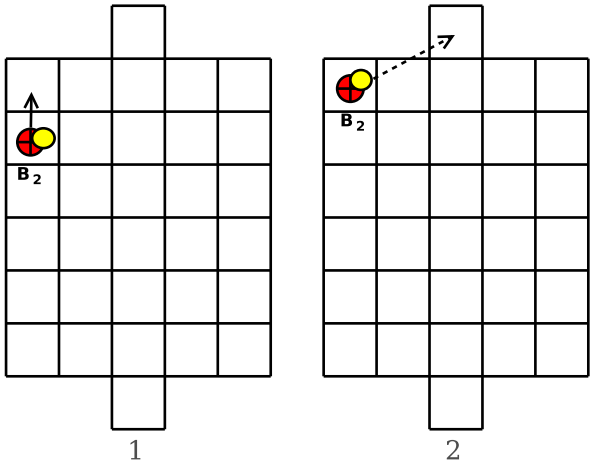


Figure 7.8: Player $B_2(3)$ kick sideways

scoring a goal. In this case the player was able to kick the ball away from the goal before the opposition team could score. The game resulted in a draw as no players were able to recover the ball from the middle of the field after the recovery move. A number of games showed ball-chasing behaviour. Successful ball recoveries such as the one demonstrated in figure 7.9 was observed in a number of other games. Looping behaviour where the player moved one block left, followed by one block right, and so forth, was noted in other games. These games typically resulted in draws, demonstrating that the fetching behaviour still needs much improvement.

7.7 Game Strategy Analysis

The previous section analysed the behaviour of each cluster centroid in isolation. Analysing each of these behaviours showed that the centroids do indeed represent different behaviours, and the proposed hypothesis that each centroid represents a different optimum is true. This section explores the more complex behaviours that emerged from different permutations of these behaviours in a team context. Interaction between players clearly demonstrates scenarios where some of the previously shown to be *bad* behaviours can result in good gameplay strategies, scoring more goals and preventing the opponents from scoring goals more often. For each of the games discussed below it is shown which

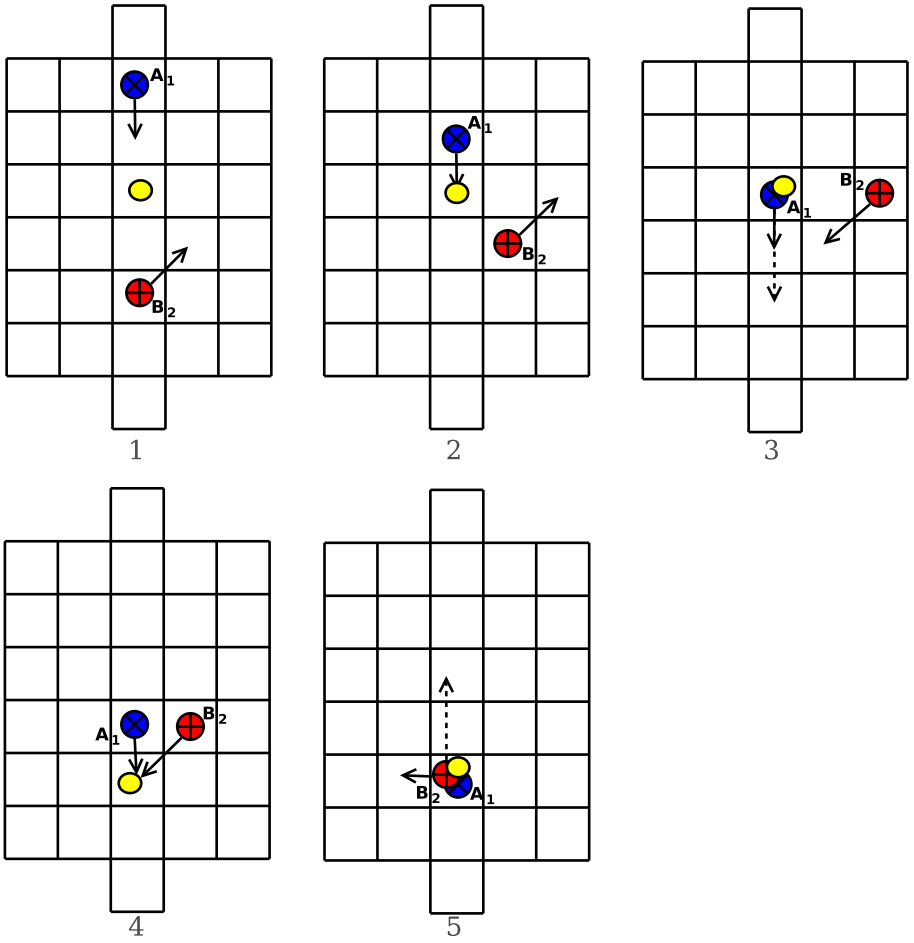


Figure 7.9: Player $B_2(4)$ moves over the field and returns to protect the goal

player (centroid) behaviour permutations form part the game. The notation used, $X_y(z)$, indicates team X, position y, and centroid number z. The centroid numbers correspond to the behaviours analysed in the previous section.

Four gameplay strategies were identified. The next four subsections discuss each of these strategies in more detail.

7.7.1 Ball Ownership Exchange

The first complex gameplay strategy that was observed is illustrated in figure 7.10. The players that made up two teams were $A_1(1)$, $A_2(1)$, $B_1(3)$ and $B_2(2)$. In this scenario the

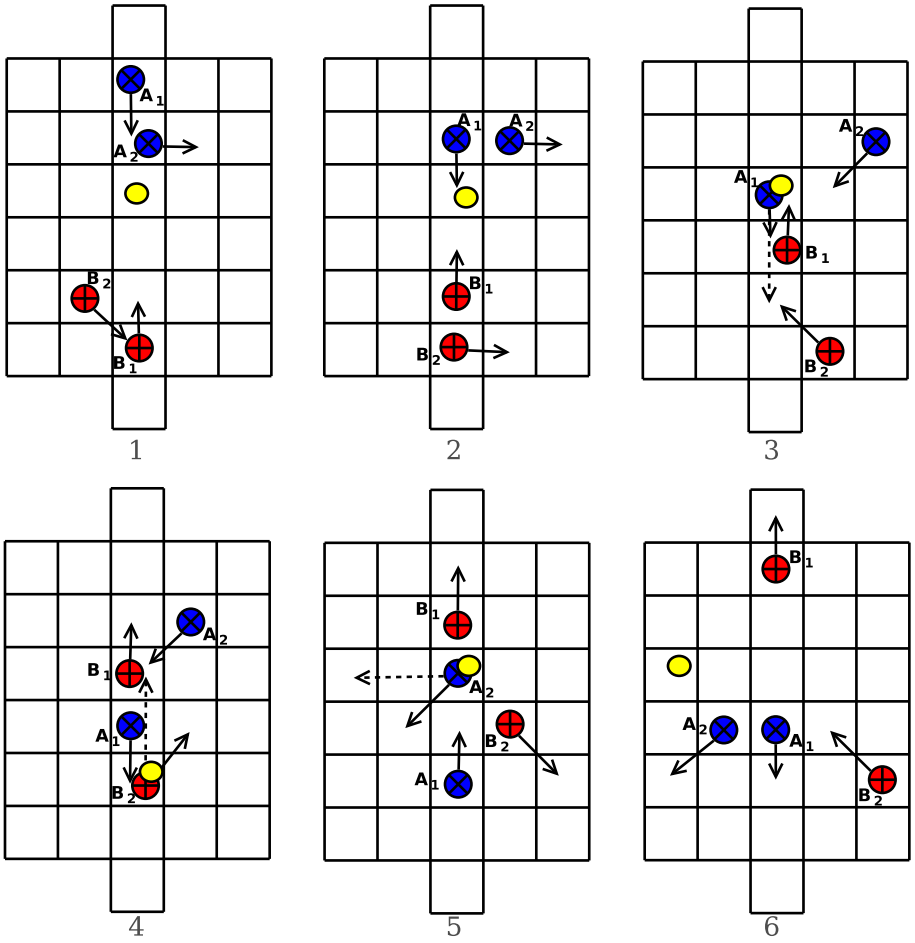


Figure 7.10: Multiple ball ownership exchanges

ball is kicked towards the team B goal by player A_1. The kick is, however, intercepted by player B_2, which in turn kicks the ball back towards the team A goal. This kick is intercepted by player A_2, and from here a sidewards kick places the ball on the left side of the playing field. The final kick performed by player A_2 matches the individual player behaviour analysed in section 7.6.2. In this case, however, this final kick effectively takes the ball out of play. The ball is considered out of play when no other players are able to recover the ball, causing the game to draw. In contrast to the general behaviour that was previously seen in section 7.6.2, player A_2 moved sideways visiting more areas on the field in this game, returning to the middle just in time to intercept the ball as it was heading towards the team A goal.

Player B_2 intercepted the first kick by player A_1, the behaviour observed here follows the individual behaviour analysed in section 7.6.4. The position of player B_2 is loosely linked to the position of B_1 and this is also the reason why player B_2 moved in to intercept the ball. Player B_2 ended up looping between the last two positions that player B_2 occupied on the field, as illustrated in figure 7.10. Another noteworthy observation is the application of the previously observed ball fetching behaviour of player A_1. In the second to last board state presented in figure 7.10, player A_1 changes direction moving back towards the ball. The fetching behaviour does not, however, extend to following the ball sideways and the player turns back towards the team B goal once the ball leaves the middle line.

Even though the game ended in a draw due to none of the players being able to find the ball in the end, the gameplay strategy visible here is more complex than what was previously seen in section 6.2.

7.7.2 Anticipatory Counter-move

As with the previous game analysed, the second game also revealed how the previously individually analysed behaviours led to more complex gameplay strategies if combined and not studied in isolation. Figure 7.11 depicts the second game to be analysed with players $A_1(4)$, $A_2(1)$, $B_1(4)$ and $B_2(3)$. Section 7.7.1 showed how player B_2 had the potential to score a goal by moving on the left edge of the playing field with the ball, all the way to the team A side. It was also shown that player A_2 kicked the ball to the left once it had the ball and was still four blocks away from the team B goal.

In the second game it becomes clear that player B_2 learned how to exploit the predictable sideways kick behaviour of player A_2. The previous game showed that the sideways kick was still effective in other games to force the game into a draw - which would be a more desirable result than a loss. This time, however, with player B_2 intercepting the ball, the attempt to force a draw is turned around to a nearly unstoppable offensive strategy. This strategy relies on, and exploits, the sideways kick performed by player A_2. It is unclear why player A_2 changes direction moving towards the side the ball was kicked.

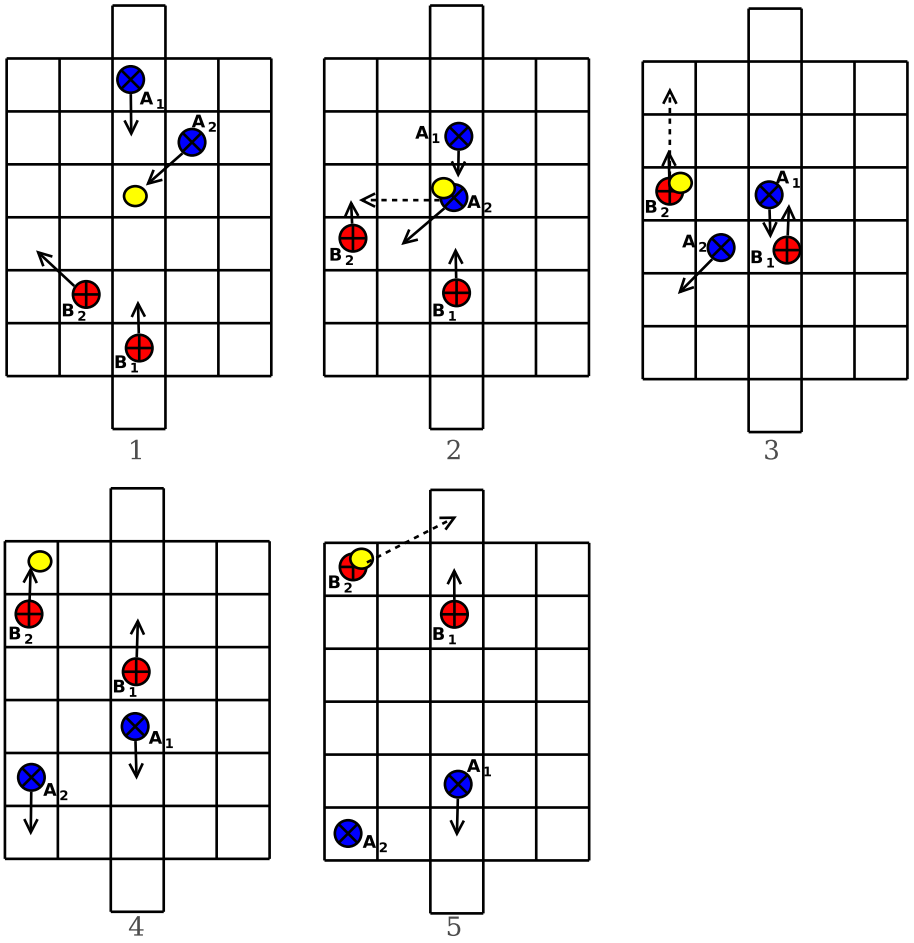


Figure 7.11: Example of the anticipatory counter-move gameplay strategy

7.7.3 Runaround Movement

Both of the previous games demonstrated how a kicked ball can be recovered, allowing the opposing team to take ownership of the ball and prevent a goal being scored against them. Basic ball-chasing behaviour was seen where a player turned around to move back towards the ball. What was not previously seen was dribbling behaviour. Figure 7.12 depicts a game where players $A_1(4)$, $A_2(2)$, $B_1(1)$ and $B_2(4)$ participated in a game where dribbling behaviour played a role in the goal-scoring strategy.

Player B_2 moves onto the block where the ball is located and continues to move towards the team A goal by moving sideways and forward. This is in contrast to the

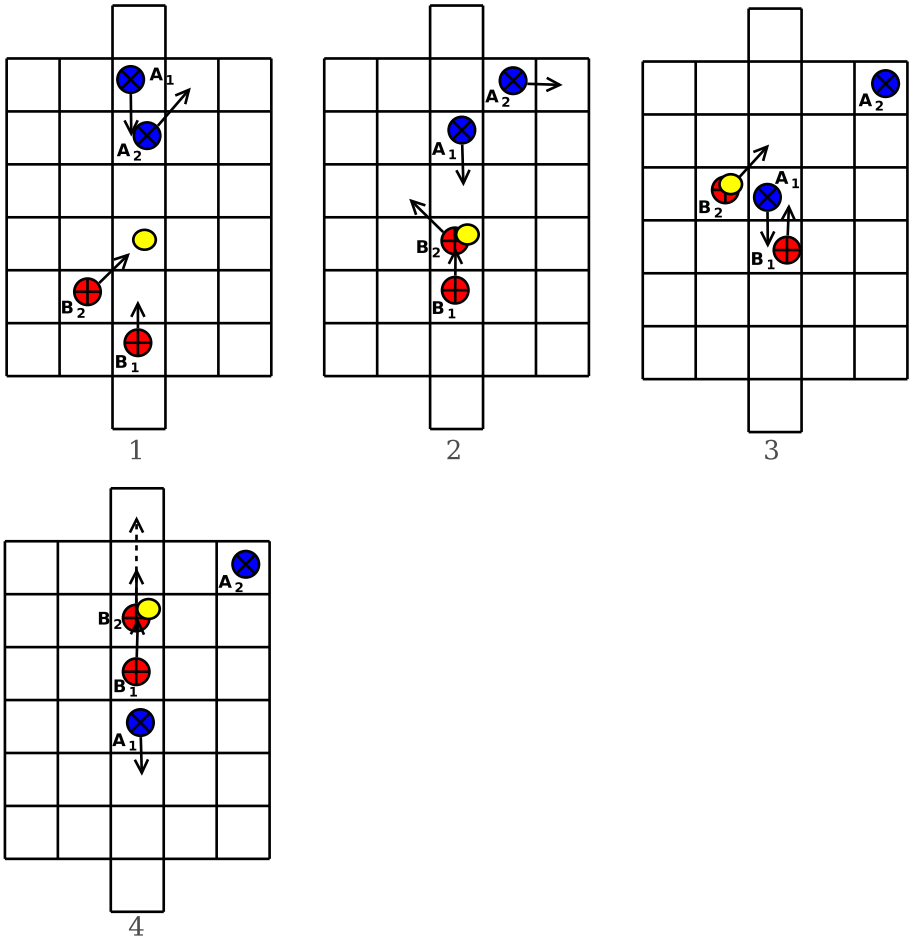


Figure 7.12: Example of the runaround movement gameplay strategy

typical behaviour that was seen in previous games where the ball was immediately kicked away as soon as a player gained ball ownership. The sidewards movement allowed the player effectively to bypass player A_1, which if it were not for the sideways movement, player B_2 would have been on the same block as player A_1. This would in turn have presented an opportunity to assume ownership of the ball. The move is finished off by moving back onto the middle line in a sideways forward movement, allowing for a direct goal kick.

Another of the previously identified behaviours is clearly visible in this example: Player A_2 takes itself out of the game by simply moving towards the upper right side of the field.

7.7.4 Complex Comeback

The final game that demonstrated a unique complex playing strategy demonstrates how an offensive strategy is turned around to defend against a goal being scored. Players $A_1(4)$, $A_2(2)$, $B_1(2)$ and $B_2(4)$ played in the game depicted in figure 7.13, showing player B_2 moving aggressively sideways and forward until team A assumes ownership of the ball.

Player A_1 follows a straightforward attack strategy where the ball is kicked towards the team B goal, while the player also moves towards the goal. The ball is kicked over player B_1 to prevent losing the ball to team B. Player A_1 keeps on moving towards the

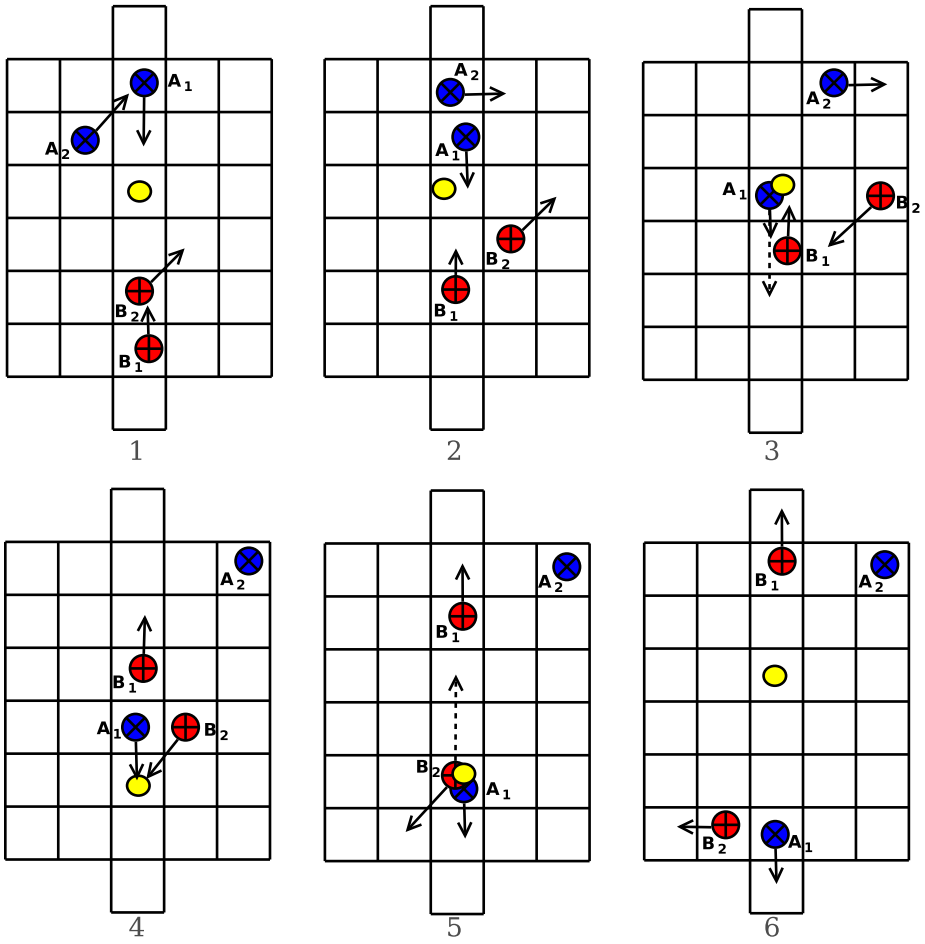


Figure 7.13: Example of the complex comeback gameplay strategy

block with the ball for the final goal kick. However, player B_2 at this point is the same distance away from the ball, also moving towards the ball. The two players arrive on the block with the ball at the same time, in which case ball ownership is determined randomly. In this case, B_2 managed to attain ownership. The ball is subsequently kicked towards the middle of the board, where no other players manage to retrieve it. The game ends as a draw.

It is clear that the final result depended heavily on the randomisation result that determined the ball ownership. Had player A_1 received ownership, this would certainly have been a goal kick against team B. Just as critical as the randomisation result towards stopping the goal was, is the speed at which player B_2 returned towards the goal to prevent the goal kick. Player B_2 changed direction from moving towards the team A side back to the team B side at the same step that player A_1 assumed ball ownership. At this point the strategy changed from offensive to defensive, a behaviour not seen in the earlier games.

The inability of the players to recover the ball from the middle of the board is an area of weakness for the majority of the evolved players. In this specific case the final kick does prevent the opposition from scoring a goal and forces a draw, a result that is more desirable than a loss.

7.8 Performance analysis

The previous two sections conducted a detailed visual analysis of the games played, identifying player behaviours and gameplay strategies. The visual analysis presented a limited-scope review of the player behaviours, although it did show behavioural traits that evolved which would otherwise not be visible in a quantitative analysis. This section provides the reader with a quantitative analysis of the trained players.

Each player, representing a cluster centroid in the particle swarm, was evaluated playing 15000 games in each team combination. The results are presented in tables 7.1 and 7.2. Table 7.1 presents the results for the different team permutations averaged over all the opposition team permutations. Four measurements are reported in the table: The average number of goals scored in each game, the average iterations it took for a goal to

be scored, the percentage of games that ended in a draw, and the percentage of games the team won. It should be noted that the percentage of games ending in a draw were reported in a separate column in the tables and not calculated as part of the win/loss percentage. For example, the team $\{A_1(1), A_2(2)\}$ won 75% of the rounds not ending in a draw. Table 7.2 presents the same results at player level.

A number of observations can be made by reviewing the team performance results:

- Team B teams never lost more than 50% of the games.
- Three team B teams won 100% of the games that did not end in a draw. A minimum of 30% of the games played by these teams ended in draws.
- Two team A teams lost 100% of the games. A minimum of 37% of the games played by these teams ended in draws.
- Team $\{A_1(2), A_2(2)\}$ did not score a single goal in all the games.
- 20.41% of the total number of games played resulted in draws.
- With the exception of three team A teams, $\{A_1(1), A_2(1)\}$, $\{A_1(1), A_2(2)\}$ and $\{A_1(2), A_2(1)\}$, it took an average of 5 iterations per goal. For the three team A teams it took an average of 6 iterations per goal.

The results for the two team A teams, $\{A_1(2), A_2(1)\}$ and $\{A_1(2), A_2(2)\}$, that lost all of their games can be traced back to the player performance figures in table 7.2 for player $A_1(2)$, revealing that this player did not win a single game in any team combination. In addition, this player also had the highest game draw percentage of 38.58%.

The average number of iterations per goal can also be related to the player performance figures. Player $A_1(1)$ and $A_1(2)$ are the only players that had an average of 6 iterations per goal, corresponding to the three team A teams, $\{A_1(1), A_2(1)\}$, $\{A_1(1), A_2(2)\}$ and $\{A_1(2), A_2(1)\}$, in table 7.2. The fourth team A combination, $\{A_1(2), A_2(2)\}$, did not score any goals so no average was recorded. From the Simple Soccer rules it can be deduced that the minimum number of iterations required to score a goal is 4 in the case where player A_1 or B_1 scores the goal and 5 in the case where player A_2 or B_2 scores the goal. Only player $B_2(4)$ had an average number of iterations per goal less

Table 7.1: Team performance

Team		Average goals scored	Iterations per goal	% draws	% wins
A_1	A_2				
1	1	2.482	5.952	22.10	37.50
1	2	3.286	6.479	16.23	75.00
2	1	0.533	6.000	39.86	0.00
2	2	0.000		37.30	0.00
3	1	2.268	5.000	16.25	12.50
3	2	3.403	5.000	7.72	37.50
4	1	2.272	5.000	16.17	12.50
4	2	3.403	5.000	7.67	25.00
B_1	B_2				
1	1	3.540	5.023	12.66	75.00
1	2	1.844	5.032	36.95	50.00
1	3	4.037	5.022	7.53	62.50
1	4	1.761	4.968	39.35	62.50
2	1	3.538	5.023	12.60	75.00
2	2	2.439	5.031	25.46	87.50
2	3	4.056	5.022	7.65	62.50
2	4	2.198	4.967	31.10	100.00
3	1	3.536	5.023	12.55	75.00
3	2	2.434	5.031	25.63	87.50
3	3	4.048	5.023	7.50	50.00
3	4	2.196	4.967	31.35	100.00
4	1	3.539	5.023	12.35	75.00
4	2	2.434	5.032	25.52	87.50
4	3	4.036	5.023	7.64	50.00
4	4	2.219	4.994	30.57	100.00

Table 7.2: Player performance

Player	Average goals scored	Iterations per goal	% draws	% wins
A_1				
1	2.884	6.215	19.17	56.25
2	0.267	6.000	38.58	0.00
3	2.836	5.000	11.98	25.00
4	2.837	5.000	11.92	18.75
A_2				
1	1.889	5.454	23.59	15.63
2	2.553	5.493	17.23	34.38
B_1				
1	2.796	5.011	24.12	62.50
2	3.057	5.011	19.20	81.25
3	3.053	5.011	19.26	78.13
4	3.057	5.018	19.07	78.13
B_2				
1	3.538	5.023	12.59	75.00
2	2.288	5.031	28.39	78.13
3	4.044	5.023	7.58	56.25
4	2.093	4.974	33.09	90.63

than 5. Based on the minimum number of iterations needed to score a goal, it can be deduced that player $B_2(4)$ had to have scored the goals in those cases. Player $B_2(4)$ was also previously shown, in section 7.6.4, to have exhibited the most complex behaviour in the visual analysis, a trait that is also clearly visible in the discrete measurements. The team performance results shown indicate that $B_2(4)$ won all the games in all its team combinations, except when paired with player $B_1(1)$ - leading to an average of 90.63% games won. Player $B_1(1)$ is shown to be the weakest of the team B players, winning only 62.50% of the games.

Note that there is not a very strong correlation between the average number of

goals scored and the outcome of a game. A high average number of goals does not automatically lead to a win. Player $B_2(4)$, the best performing player, maintained an average of only 2.093 goals scored per game while winning 90.63% of the games. The only player with a lower number of iterations per goal was $A_2(2)$, which also lost all its games.

Also note the influence that player $B_2(3)$ had on the teams in which it played. All these games had the lowest draw percentages, but not the best performance. These team compositions resulted in quite poor performance when evaluating the percentage of games won. As previously noted in the visual analysis, this player had learned a very specific move - to intercept a sideways kick and perform an angled goal kick. In cases where it did not have the ball, it simply moved towards the opposition goal on the side of the field. The measured bad performance of this player emphasises the importance of both players in a team staying in the game for the best possible result - once one player is taken out of the game, the combined performance decreases visibly. This also demonstrates the importance of defensive play as the low draw percentage is simply due to the opposition team's scoring goals more easily. Remember that a draw is more desirable than a loss result. It is also illustrated that the trained players are able to exploit even a small weakness in their opposition. In this case, instead of drawing more games, they succeed in scoring more goals against their weak opposition.

7.9 Summary

The first objective of this chapter was to address the training saturation problem identified in the previous chapter. It was determined that the saturation occurred due to the neural network weights leaving the active regions of the activation function. To counter this problem the personal best position of each particle in the swarm was clamped to the region $(-5, 5)$. The swarm diversity reflected the effect of the clamping by dropping considerably. However, it was shown that this was not enough to improve the gameplay strategies. A second change was made to the algorithm to linearly decrease the perception limit and core of the charged particle swarm algorithm, effectively morphing the CCPSO, over 250 iterations, into the standard particle swarm optimisation algorithm.

The swarm diversity did not decrease as much as expected. In fact, the swarm diversity remained notably high.

A new measurement was introduced to analyse the behaviour of the global best particle in order to better understand why the particle swarm diversity was so high. The new measurement revealed that the global best position was in effect jumping over large distances between iterations. Owing to the clamping on particle velocities, it was hypothesised that the algorithm converged on multiple optimal positions in the hyper-dimensional search space. The reasoning behind the hypothesis was that each optimal point represented a different behaviour for the player, and there is no clear winner when evaluating different behaviours against each other. This hypothesis was confirmed by applying a clustering algorithm to the evolved swarms. The X-means clustering algorithm was used to perform the clustering. A number of well-formed clusters were found for each swarm, and each cluster centroid was calculated for further analysis.

The second objective of this chapter was to perform an in-depth analysis of the trained player behaviours. Visual analysis confirmed that each cluster centroid indeed represented different player behaviours. The training algorithm evolved a number of candidate optimal players for each position and the swarm indeed tried to find multiple optimal positions instead of simply converging on a single player behaviour. Visual and quantitative analysis was performed on a set of trained players, demonstrating that the training algorithm succeeded in training players from zero knowledge to play the game of Simple Soccer. The visual and quantitative analysis further showed that trained players were indeed capable of exploiting weaknesses in the opposition teams, that is, scoring many more goals and drawing fewer games when facing weak opponents. Various intelligent playing strategies were also visible in the visual analysis, backed up by the quantitative results.

Overall, it was shown that the new *CCPSO(t)* training algorithm, along with the new *FIFA league ranking* relative fitness function, is capable of training players from zero knowledge to successfully playing a complex non-deterministic game.

The next chapter summarises the findings of this study. Future research areas that flow out of this study are also identified and briefly discussed.

Chapter 8

Findings and Conclusions

"History is the version of past events that people have decided to agree upon."

Napoleon Bonaparte (1769-1821)

This chapter summarises the major findings of the work done for this study, summarises the conclusions drawn from the study, and provides a number of suggestions for future work that can be pursued as a result of this work.

8.1 Summary of Findings and Conclusions

This study aimed to develop a coevolutionary algorithm to train neuro-controlled multi-agent teams from zero knowledge. Various computational intelligence techniques led to the development of exactly such an algorithm. A model was developed with the specific purpose of providing a re-usable benchmark to evaluate the performance of this newly developed algorithm.

Background was provided into the classical robotics soccer problem along with the introduction of the simple soccer model. Enhancements were made to the original model based on previous research done in coevolutionary soccer models.

After reviewing an existing competitive PSO training algorithm, the design of the new multi-population team-based coevolutionary training algorithm based on the charged

particle swarm optimiser was presented. All aspects, from the particle and population representation and the neural network architecture to the particle swarm optimiser, were discussed and described in detail. Player performance benchmarking techniques were briefly discussed and parameter optimisation of the training algorithms' parameters was performed. The early results obtained from the parameter optimisation of the training algorithms' parameters were worrisome: the new algorithm did not train well-performing simple soccer players in a consistent manner; it was revealed that there was a high variance in the performance of some of the trained players, duely noted as outliers in the data.

The relative fitness function used by the CCPSO training algorithm was shown to be the cause of the outliers: the relative fitness function did not provide enough information about the players' performance for consistent training to take place. Various alternative fitness functions were discussed. A number of relative fitness functions proved to bias the training into specific behaviours, thus violating the zero-knowledge goal of this study. Owing to the fact that the non-biasing functions did not reveal enough about the players' performance, a new non-biasing function based on the *FIFA league ranking* was developed. The newly developed *FIFA league ranking* relative fitness function takes player skill level and past performance into account to improve the accuracy of the relative fitness value. The *FIFA league ranking* relative fitness function provided the required level of detail to avoid the previously encountered outlier problem and proved successful in consistently training players. Parameter optimisation was repeated using the *FIFA league ranking* relative fitness function.

A visual analysis was conducted of the playing strategies of the teams evolved by the CCPSO training algorithm. A number of noteworthy strategies that emerged from the training were discussed in detail, all of which proved to be fairly simplistic in nature. The reason for these weak and simplistic strategies was investigated further.

It was revealed that the CCPSO training algorithm stagnated after only a few iterations, the cause of which was identified as the particles leaving the region of the hyper-dimensional search space that mapped to the active region of the neural network activation function. Once the active region was left, there was no further influence on the neural networks' behaviour, saturating the resulting gameplay strategies.

Various ways of improving the CCPSO training algorithm's performance were investigated. The first was to provide a solution to the saturation problem, by preventing the particles from leaving the active region of the activation function. Bounding the personal best position of the particles in the swarms solved that problem. Swarm diversity measurements, however, showed that the swarm diversity did not decrease as expected.

The charged particle swarm's behaviour was analysed and investigated further as a possible cause for the high diversity observed in the swarms. The algorithm was enhanced with a decreasing perception and core limit, effectively morphing the charged particle swarm into a standard particle swarm algorithm. This did not have a noticeable effect on the measured swarm diversity, even though it would have an effect on a theoretical level, as the particles are now *allowed* to converge on a solution.

A behavioural analysis of the global best position revealed that the global best position jumped considerable distances between iterations. This was seen as a sign that the particle swarm might be converging on multiple optimal areas in the search space. In order to verify this hypothesis, the X-means clustering algorithm was applied to the particles. It was revealed that the particles were indeed clustering around certain areas in the search space. The cluster centroids were calculated as candidate players for each position for further investigation.

Each centroid was evaluated as it played in each team permutation against each opposition team permutation. The resulting games were visually inspected to identify the behavioural traits of each player and team. The resulting strategies were discussed in detail. To counter the possible subjectivity of the visual analysis, a quantitative analysis of the performance was also conducted and the results of that analysis were discussed in detail. The results revealed that strong gameplay strategies, based on at least a manner of teamwork, emerged. In addition, it was shown that each centroid represents a different player behaviour that should be treated, and thus scored, individually as a candidate player for the team. The results also revealed that weaknesses in the players are easily exploited by the opposition team, indicating that a good team required not only a good offensive behaviour, but also a good defensive behaviour.

Overall it was shown that the new CCPSO(t) training algorithm, along with the new *FIFA league ranking* relative fitness function, was capable of training players from zero

knowledge to successfully play a complex, non-deterministic game.

A large number of extensions and future work can be derived from the results of this study. The next section discusses some recommended areas for future work.

8.2 Future Work

Throughout this study several new ideas for future research presented themselves. Below is a summary of each of these.

Improving convergence

Experimental work in this study demonstrated that the particle swarms had limited success in converging on a single solution. Swarm diversity measurements indicated that the swarm diversity remained high throughout the training process. Even though each of the optimal points being converged on represented a different playing strategy, the algorithm did not manage to fully optimise any of the found optimal points.

Improving convergence will allow the particle swarm to optimise individual playing strategies. Further research is required to find a way of optimising each of the positions identified by the particle swarms.

Adaptive strategies

Experimental work in this study demonstrated that each particle swarm found multiple *optimal* regions in the hyper-dimensional search space. Observations showed that each optimal point represented different player behaviours.

In the real world a good player will choose a playing strategy dependent on the opponent he faces. Switching between strategies gives the player an edge above his opponent, allowing the player to win. When a player does not switch between strategies, he gives the opponent the opportunity to predict responses to moves he makes, giving the opponent the edge.

A suitable mechanism for switching between strategies should be developed. This mechanism should allow for the optimal strategy to be selected during gameplay, and

so allow a player to respond to environmental conditions and counter specific strategies used by the opposition team.

Different neural network architectures

The algorithm proposed in this study does not dictate the architecture of the neuro-controllers used to control the individual players. By optimising the neural network architecture further it might be possible to develop better playing strategies. The feed forward neural network used in this study does not allow for players to recall past moves. This prevents the player from learning and adapting the players' own moves based on how the opponent plays. Switching to a recurrent neural network architecture may allow for a form of temporal information to be incorporated into each game.

Optimisation of the neural network architecture might serve to increase player performance and help prevent overfitting, as the feed forward neural network used in this study is typically seen as quite simplistic. It should be noted that the neural network weight analysis showed that the architecture as used in this study was not overly simplistic for the problem being solved. Additional research is required to better understand the impact of different architectures on the training algorithm's performance.

PSO algorithm variations

This study made use of the charged particle swarm optimisation algorithm. This algorithm was chosen as it addresses problems typically encountered in dynamic environments. Many other particle swarm variations exist that also attempt to solve the problems associated with dynamic environments. The quantum PSO, for instance, should be investigated as a candidate PSO algorithm to be used with the training algorithm presented in this study.

In addition to the PSO algorithm itself, the effect of different neighbourhood structures and swarm size, should be investigated more thoroughly. The design of the PSO algorithm used in this study was inspired by past research done in the field, although, this work has shown a number of new problems with using the PSO to train neuro-controlled players.

Specialised players

The simple soccer model used in this study treats all players as equal. However, in real world soccer, this is not the case. The goalie, for instance, is allowed to pick up the ball with his hands in certain areas of the field, giving him an advantage over other players. This advantage strengthens a team's defensive ability, and when a player attempts to score, it can almost be seen as an unfair comparison as the striker is prevented from using his hands.

The possibility of extending the simple soccer model further to introduce the concept of specialised players should be investigated. Such an extension will also result in the training algorithm exploiting these special abilities of players.

Comparative study

Analysis of the training algorithm was done in order to investigate whether the algorithm succeeded in training players from zero knowledge. The results, both visually and quantitatively, demonstrated that the training algorithm did succeed in training game-playing agents. No performance comparison was made with existing training techniques. Although many of the existing techniques do not train from zero knowledge and thus do not provide a level playing field for comparisons, they will still be useful in identifying areas of improvement and shortcomings in the proposed training algorithm.

The fact that the proposed algorithm trains players from zero knowledge makes it extremely valuable. When investigating how a robot can solve an unknown task it is important to measure the quality of the solutions found by the algorithm against previously optimised solutions.

Different problems

Even though simple soccer was selected for testing the proposed training algorithm, the algorithm itself is not bound to the problem. The only requirement is that the relative fitness function is informed about who won a round in the game. It is not even required to know by how much, as that already would introduce domain specific information into the training process.

An investigation into how the algorithm performs for different problems should be carried out. Problems with varying degrees of complexity should be considered, such as team based construction or navigation problems. The focus of the proposed algorithm is training teams of players from zero knowledge. That makes a number of problems not previously considered available as candidates for future research work.

Investigating robustness

After the parameter optimisation was conducted, the performance of the algorithm increased tremendously. The question that this opens is: How robust is the algorithm to deal with changes in the rules? A certain amount of robustness has already been shown in handling the incomplete information and randomness aspects of the simple soccer problem, and it was shown that these are handled extremely well.

A more conclusive study of the robustness of the algorithm is required. The algorithms' performance should be examined when the rules are changed or, alternatively, when the field size is changed. The results of such a study could be used to improve the performance of the algorithm further.

Bibliography

[1] Aldebaran Robotics. The NAO league. `http://www.aldebaran-robotics.com/en/Solutions/For-Robocup/the-NAO-league.html`, August 2012.

[2] P.J. Angeline and J.B. Pollack. Competitive Environments Evolve Better Solutions for Complex Tasks. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, volume 270, pages 264–270, 1993.

[3] M. Asada, H. Kitano, I. Noda, and M. Veloso. RoboCup: Today and tomorrow-What we have learned. *Artificial Intelligence*, 110(2):193–214, 1999.

[4] R. Axelrod and W.D. Hamilton. The Evolution of Cooperation. *Science*, 211(4489):1390–1396, 1981.

[5] T. Back, D.B. Fogel, and T. Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. Taylor & Francis, 2000.

[6] T. Back, D.B. Fogel, and T. Michalewicz. *Evolutionary Computation 2: Advanced Algorithms and Operations*. Taylor & Francis, 2000.

[7] R. Battiti. First-and Second-Order Methods for Learning: between Steepest Descent and Newton's method. *Neural Computation*, 4:141–166, 1992.

[8] T.M. Blackwell. Particle Swarms and Population Diversity I: Analysis. In *Genetic and Evolutionary Computation Conference Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 9–13, 2003.

[9] T.M. Blackwell. Particle Swarms and Population Diversity II: Experiments. In *Genetic and Evolutionary Computation Conference Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, 2003.

[10] T.M. Blackwell. Swarms in Dynamic Environments. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, pages 1–12. Springer-Verlag, 2003.

[11] T.M. Blackwell and P.J. Bentley. Dont push me! Collision-avoiding swarms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1691–1696, 2002.

[12] T.M. Blackwell and P.J. Bentley. Dynamic Search with Charged Swarms. In *Proceedings of the 2002 Genetic and Evolutionary Computation Conference*, pages 19–26. Morgan Kaufmann Publishers, 2002.

[13] T.M. Blackwell and J. Branke. Multi-swarm Optimization in Dynamic Environments. *Applications of Evolutionary Computing*, (3005):489–500, 2004.

[14] T.M. Blackwell and J. Branke. Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.

[15] 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.

[16] A.J. Carlisle and G. Dozier. Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the 5th Biannual World Automation Congress*, volume 13, pages 265–270, 2002.

[17] A.J. Carlisle and G.V. Dozier. Adapting Particle Swarm Optimization to Dynamic Environments. In *Proceedings of the International Conference on Artificial Intelligence*, volume 1, pages 429–434, 2000.

[18] A.J. Carlisle and G.V. Dozier. Tracking Changing Extrema with Particle Swarm Optimizer. Technical report, CSSE01-08, Auburn University, 2001.

[19] A.J. Carlisle and G.V. Dozier. *Applying the Particle Swarm Optimizer to Non-Stationary Environments*. Phd thesis, Auburn University, 2002.

[20] S.K. Chalup, C.L. Murch, and M.J. Quinlan. Machine Learning With AIBO Robots in the Four-Legged League of RoboCup. *IEEE Transactions on Systems, Man, and Cybernetics Part C*, 37(3):297–310, 2007.

[21] K. Chellapilla and D.B. Fogel. Evolution, Neural Networks, Games, and Intelligence. *Proceedings of the IEEE*, 89(9):1471–1496, 1999.

[22] 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.

[23] 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 the 2000 Congress on Evolutionary Computation*, volume 2, pages 857–863, 2002.

[24] S.H. Clearwater, T. Hogg, and B.A. Huberman. Cooperative Problem Solving. In *Computation: The Micro and the Macro View*, volume 275, pages 33–70. World Scientific, June 1992.

[25] M. Clerc. The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1951–1957, 1999.

[26] M. Clerc. Think Locally, Act Locally: The Way of Life of Cheap-PSO, an Adaptive PSO. Technical report, http://clercmaurice.free.fr/pso/, 2001.

[27] 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, 2002.

[28] J.P. Coelho, P.B. De Moura Oliviera, and J. Boa Ventura Cunha. Non-Linear Concentration Control System Design using a New Adaptive PSO. In *Proceedings of the 5th Portuguese Conference on Automatic Control*, 2002.

[29] J. Conradie and A.P. Engelbrecht. Training Bao Game-Playing Agents using Co-evolutionary Particle Swarm Optimization. In *Proceedings of the 2006 IEEE Symposium on Neural Networks*, pages 67–74, 2006.

[30] N.L. Cramer. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, volume 183, pages 183–187, 1985.

[31] C. Darwin. *The Origin of Species: By Means of Natural Selection- or the Preservation of Favoured Races in the Struggle for Life.* 1859.

[32] C. Darwin. *On the Various Contrivances by which British and Foreign Orchids are Fertilised by Insects, and on the Good Effects of Intercrossing.* John Murray, 1862.

[33] Oxford English Dictionary. Oxford Online Dictionary (Co-evolution). `http://oxforddictionaries.com/definition/english/co-evolution`, August 2012.

[34] D. Dong, J. Jie, J. Zeng, and M. Wang. Chaos-Mutation-based Particle Swarm Optimizer for Dynamic Environment. In *Third International Conference on Intelligent System and Knowledge Engineering*, pages 1032–1037, 2008.

[35] H.H.L.M. Donkers, H.J. van den Herik, and J.W.H.M. Uiterwijk. Opponent-Model Search in Bao: Conditions for a Successful Application. *10th International Conference on Advances in Computer Games, Many Games, Many Challenges*, 263:309–324, 2003.

[36] J.G.O.L. Duhain and A.P. Engelbrecht. *Particle Swarm Optimization in Dynamically Changing Environment - An Empirical Study.* Msc thesis, University of Pretoria, 2011.

[37] W.H. Durham. *Coevolution: Genes, Culture and Human Diversity.* Stanford University Press, Stanford, CA, USA, 1992.

[38] R.C. Eberhart and J. Kennedy. A New Optimizer using Particle Swarm Theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[39] R.C. Eberhart and Y. Shi. Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, pages 84–88. Piscataway, NJ, USA: IEEE, 2000.

[40] R.C. Eberhart and Y. Shi. Particle Swarm Optimization: Developments, Applications and Resources. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 81–86. Piscataway, NJ, USA: IEEE, 2001.

[41] R.C. Eberhart and Y. Shi. Tracking and Optimizing Dynamic Systems with Particle Swarms. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 94–100, 2001.

[42] R.C. Eberhart, P. Simpson, and R. Dobbins. *Computational Intelligence PC Tools*. Academic Press Professional, Inc., San Diego, CA, USA, 1996.

[43] J.L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

[44] A.P. Engelbrecht. *Sensitivity Analysis of Multi-Layer Feedforward Neural Networks*. Phd thesis, University of Stellenbosch, 1999.

[45] A.P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[46] A.P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2007.

[47] A.P. Engelbrecht and A. Ismail. Training Product Unit Neural Networks. *Stability and Control: Theory and Applications*, 2(1/2):59–74, 1999.

[48] A.P. Engelbrecht and F. van den Bergh. Particle Swarm Weight Initialization In Multi-layer Perceptron Artificial Neural Networks. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, pages 365–370. Citeseer, 1999.

[49] H. Fan. A Modification to Particle Swarm Optimization Algorithm. *Engineering Computations*, 19(8):970–989, 2002.

[50] I. Fehérvári and W. Elmenreich. Evolving Neural Network Controllers for a Team of Self-organizing Robots Self-organizing Systems. *Journal of Robotics*, 2010.

[51] FIRA. Federation of International Robot-soccer Association. http://www.fira.net.

[52] D.B. Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 2002.

[53] L.J. Fogel. Autonomous Automata. *Industrial Research*, 4(2):14–19, 1962.

[54] L.J. Fogel. *On the Organization of Intellect*. Phd thesis, University of California, Los Angeles, 1964.

[55] L.J. Fogel and D.B. Fogel. Artificial Intelligence through Evolutionary Programming. Technical report, Contract PO-9-X56-1102C-1, US Army, 1986.

[56] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[57] C.J. Franken. Visual Exploration of Algorithm Parameter Space. *Proceedings of IEEE Congress on Evolutionary Computation*, pages 389–398, 2009.

[58] C.J. Franken and A.P. Engelbrecht. Comparing PSO Structures to Learn the Game of Checkers from Zero Knowledge. In *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 234–241, 2003.

[59] C.J. Franken and A.P. Engelbrecht. Evolving intelligent game-playing agents. In *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology*, pages 102–110. South African Institute for Computer Scientists and Information Technologists, 2003.

[60] C.J. Franken and A.P. Engelbrecht. PSO Approaches to Coevolve IPD Strategies. In *Proceedings of the 2004 Congress on Evolutionary Computation*, volume 1, pages 356–363, 2004.

[61] C.J. Franken and A.P. Engelbrecht. *PSO-based Coevolutionary Game Learning*. Msc thesis, University of Pretoria, 2004.

[62] M. Friedman and L.J. Savage. *Planning experiments seeking maxima*. McGraw-Hill, 1947.

[63] C. Fujiko and J. Dickinson. Using the Genetic Algorithm to Generate LISP Source Code to Solve the Prisoner's Dilemma. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their Application*, pages 236–240, Cambridge, Massachusetts, United States, 1987. L. Erlbaum Associates Inc.

[64] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

[65] S. Grossberg. Contour Enhancement, Short Term Memory, and Constancies in Reverberating Neural Networks. *Studies in Applied Mathematics*, 52(3):213–257, 1973.

[66] V.G. Gudise and G.K. Venayagamoorthy. Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 110–117, 2003.

[67] G.R. Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Phd thesis, University of Michigan, Ann Arbor, MI, USA, 1997.

[68] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1998.

[69] MR Hestenes and E Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[70] J.F. Hicklin. *Application of the Genetic Algorithm to Automatic Program Generation*. PhD thesis, University of Idaho, 1986.

[71] N. Hirata, A. Ishigame, and H. Nishigaito. Neuro Stabilizing Control based on Lyapunov Method for Power System. In *Proceedings of the 41st SICE Annual Conference*, volume 5, pages 3169 – 3171, 2002.

[72] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992.

[73] X. Hu and R. Eberhart. Multiobjective Optimization using Dynamic Neighborhood Particle Swarm Optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1677–1681, 2002.

[74] X. Hu and R.C. Eberhart. Tracking Dynamic Systems with PSO: Where's the Cheese? In *Proceedings of the Workshop on Particle Swarm Optimization*, pages 80–83, 2001.

[75] X. Hu and R.C. Eberhart. Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1666–1670, 2002.

[76] A. Ismail and A.P. Engelbrecht. *Training and Optimization of Product Unit Neural Networks*. Msc thesis, University of Pretoria, 2001.

[77] I.K. Jeong and J.J. Lee. Evolving Multi-agents using a Self-organizing Genetic Algorithm. *Applied Mathematics and Computation*, 88(2-3):293–303, 1997.

[78] M.I. Jordan. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In *IEEE Computer Society Neural Networks Technology Series*, pages 112–127, 1990.

[79] KAIST. KAIST Robot Intelligence Technology Lab. `http://rit.kaist.ac.kr/home/jhkim/Biography_en`, August 2012.

[80] KAIST. Korea Advanced Institute of Science and Technology. `http://www.kaist.edu`, August 2012.

[81] H. Kargupta. SEARCH, computational processes in evolution, and preliminary development of the gene expression messy genetic algorithm. *Complex Systems*, 11(4):233–287, 1997.

[82] S.A. Kauffman and S. Johnsen. Coevolution to the Edge of Chaos: Coupled Fitness Landscapes, Poised States, and Coevolutionary Avalanches. *Journal of Theoretical Biology*, 149(4):467–505, 1991.

[83] J. Kennedy. The Particle Swarm: Social Adaptation of Knowledge. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[84] J. Kennedy. Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. In *Proceedings of the 1999 Congress of Evolutionary Computation*, volume 3, pages 1931–1938, 1999.

[85] J. Kennedy and R.C. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, April 1995.

[86] J. Kennedy and R.C. Eberhart. The Particle Swarm: Social Adaptation in Information-Processing Systems. In *New Ideas in Optimization*, pages 379–388. McGraw-Hill, 1999.

[87] J. Kennedy and R.C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[88] J. Kennedy and R. Mendes. Population Structure and Particle Swarm Performance. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1671–1676, 2002.

[89] J.H. Kim, D.H. Kim, Y.J. Kim, and K.T. Seow. *Soccer robotics*. Number 11 in Springer Tracts in Advanced Robotics. Springer Verlag, 2004.

[90] S. Kiranyaz, J. Pulkkinen, and M. Gabbouj. Multi-dimensional Particle Swarm Optimization in Dynamic Environments. *Expert Systems with Applications*, 38(3):2212–2223, 2011.

[91] H Kitano. Massively Parallel Artificial Intelligence and Grand Challenge AI Applications. Technical report, SS-93-04, Association for the Advancement of Artificial Intelligence, 1993.

[92] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. RoboCup: A Challenge Problem for AI and Robotics. In *RoboCup-97: Robot Soccer World Cup I*, volume 18, pages 1–19, 1998.

[93] T. Kohonen. *Self-Organizing Maps*. Springer, 3rd edition, 2000.

[94] J.R. Koza. Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In *Proceedings of the 11th international joint conference on Artificial intelligence*, volume 1, pages 768–774. Morgan Kaufmann Publishers Inc., 1989.

[95] J.R. Koza and R. Poli. Genetic Programming. In *Search Methodologies*, pages 127–164. Springer US, 2005.

[96] T. Krink, J.S. Vesterstrøm, and J. Riget. Particle swarm optimisation with spatial particle extension. *Proceedings of the Fourth Congress on Evolutionary Computation*, 2:1474–1479, 2002.

[97] G. Lakemeyer, E. Sklar, D.G. Sorrenti, and T. Takahashi. *RoboCup 2006: Robot Soccer World Cup X*. Springer-Verlag, 2007.

[98] S.L. Laubach, J. Burdick, and L. Matthies. An Autonomous Path Planner Implemented on the Rocky 7 Prototype Microrover. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 1, pages 292–297, 1998.

[99] B.J. Leonard, A.P. Engelbrecht, and A.B. van Wyk. Heterogeneous Particle Swarms in Dynamic Environments. In *Proceedings of the 2011 IEEE Symposium on Swarm Intelligence*, pages 1–8, 2011.

[100] X. Li and K.H. Dam. Comparing Particle Swarms for Tracking Extrema in Dynamic Environments. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 3, pages 1772–1779, 2003.

[101] L. Liu, S. Yang, and D. Wang. Particle Swarm Optimization With Composite Particles in Dynamic Environments. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 40(6):1634–1648, 2010.

[102] J.D. Lohn and J.A. Reggiat. Discovery of Self-Replicating Structures using a Genetic Algorithm. In *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 678–683, 1995.

[103] M. Løvbjerg and T. Krink. Extending Particle Swarm Optimisers with Self-organized Criticality. *Proceedings of the 2002 Congress on Evolutionary Computation*, 2:1588–1593, 2002.

[104] W.S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biology*, 5(4):115–133, 1943.

[105] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle Swarms for Feedforward Neural Network Training. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 2, pages 1895–1899, 2002.

[106] L. Messerschmidt and A.P. Engelbrecht. Learning to Play Games using a PSO-based Competitive Learning Approach. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pages 444–448, 2002.

[107] L. Messerschmidt and A.P. Engelbrecht. Learning to Play Games using a PSO-based Competitive Learning Approach. *IEEE transactions on evolutionary computation*, 8(3):280–288, 2004.

[108] L. Messerschmidt and A.P. Engelbrecht. *Using Particle Swarm Optimization to Evolve Two-player Game Agents*. Msc thesis, University of Pretoria, 2005.

[109] S. Milgram. The Small World Problem. *Psychology Today*, 2(1):60–67, 1967.

[110] M.L. Minsky and S.A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[111] S. Naka, T. Genji, T. Yura, and Y. Fukuyama. Practical Distribution State Estimation using Hybrid Particle Swarm Optimization. In *Proceedings of the IEEE Power Engineering Society Transmission and Distribution Conference*, volume 2, pages 815–820, 2001.

[112] I.A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum. CLARAty: Challenges and Steps Toward Reusable Robotic Software. *International Journal of Advanced Robotic Systems*, 3(1):23–30, 2006.

[113] I. Noda, S. Suzuki, H. Matsubara, M. Asada, and H. Kitano. Overview of RoboCup-97. In *RoboCup-97: Robot Soccer World Cup I*, pages 20–41. Springer-Verlag, 1998.

[114] M.G. Omran, A. Salman, and A.P. Engelbrecht. Image Classification using Particle Swarm Optimization. In *Proceedings of the 4th Asia-pacific Conference on Simulated Evolution and Learning*, volume 2002, pages 370–374, 2002.

[115] E. Ozcan and C.K. Mohan. Analysis of a Simple Particle Swarm Optimization System. In *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 253–258, 1998.

[116] E. Ozcan and C.K. Mohan. Particle Swarm Optimization: Surfing the Waves. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 6–9, 1999.

[117] G. Pampara, A.P. Engelbrecht, and T. Cloete. CIlib: A collaborative framework for Computational Intelligence algorithms-Part I. In *Proceedings of the 2008 IEEE World Congress on Computational Intelligence*, pages 1750–1757, 2008.

[118] G. Pampara, A.P. Engelbrecht, and T. Cloete. CIlib: A collaborative framework for Computational Intelligence algorithms-Part II. In *Proceedings of the 2008 IEEE World Congress on Computational Intelligence*, pages 1764–1773, 2008.

[119] D.B. Parker. *Learning-logic: Casting the Cortex of the Human Brain in Silicon.* Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1985.

[120] E.S. Peer and A.P. Engelbrecht. *A Serendipitous software framework for facilitating collaboration in computational intelligence.* Msc thesis, University of Pretoria, 2004.

[121] D. Pelleg and A.W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, 2000.

[122] T. Peram, K. Veeramachaneni, and C.K. Mohan. Fitness-Distance-Ratio based Particle Swarm Optimization. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. SIS'03*, pages 174–181, 2003.

[123] W.R. Plant, G. Schaefer, and T. Nakashima. An Overview of Genetic Algorithms in Simulation Soccer. In *Proceedings of the 2008 IEEE World Congress on Computational Intelligence*, pages 3897–3904, 2008.

[124] M.A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution.* Phd thesis, George Mason University, 1997.

[125] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Particle Swarm Optimization with Self-adaptive Acceleration Coefficients. In *Proceedings of the 1st International Conference on Fuzzy Systems and Knowledge Discovery*, pages 264–268, 2002.

[126] I. Rechenberg. Cybernetic Solution Path of an Experimental Problem. In *Royal Aircraft Establishment Translation No. 1122, B. F. Toms, Trans. (August 1965).* Ministry of Aviation, Royal Aircraft Establishment, 1965.

[127] I. Rechenberg. *Evolutionsstrategie - Optimierung technisher Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Technical University of Berlin, 1971.

[128] RoboCup. The robocup standard platform league. `http://www.robocup.org/robocup-soccer/standard-platform/`, August 2012.

[129] RoboCup. Robot soccer world cup. `http://www.robocup.org`, August 2012.

[130] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Book, New York, 1962.

[131] C.D. Rosin and R.K. Belew. Methods for Competitive Co-evolution: Finding Opponents Worth Beating. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 373–380. Morgan Kaufmann Publishers, Inc., 1995.

[132] C.D. Rosin and R.K. Belew. New Methods for Competitive Coevolution. *Evolutionary Computation*, 5(1):1–29, January 1997.

[133] D.E. Rumelhart, G.E. Hintont, and R.J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[134] A.L. Samuel. Some Studies in Machine Learning using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[135] J.F. Schutte and A.A. Groenwold. Sizing Design of Truss Structures using Particle Swarms. *Structural and Multidisciplinary Optimization*, 25(4):261–269, 2003.

[136] H.P. Schwefel. *Cybernetic Evolution as Experimental Research Strategy in Fluid Mechanics*. Diploma in engineering (aero- and space technology), Technical University of Berlin, 1965.

[137] M. Settles and B. Rylander. Neural Network Learning using Particle Swarm Optimizers. In *Advances in Information Science and Soft Computing*, pages 224–226, 2002.

[138] Y. Shi and R.C. Eberhart. A Modified Particle Swarm Optimization. In *Proceedings of the 1998 IEEE World Conference on Computational Intelligence*, volume 25, pages 69–73, 1998.

[139] Y. Shi and R.C. Eberhart. Parameter Selection in Particle Swarm Optimization. In *Evolutionary Programming VII*, pages 591–600. Springer, 1998.

[140] Y. Shi and R.C. Eberhart. Empirical Study of Particle Swarm Optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1948–1950, 1999.

[141] Y. Shi and R.C. Eberhart. Fuzzy Adaptive Particle Swarm Optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 101–106, 2001.

[142] A. Silva, A. Neves, and E. Costa. An Empirical Comparison of Particle Swarm and Predator Prey Optimisation. In *Artificial Intelligence and Cognitive Science*, volume 2464, pages 1–45. Springer Berlin / Heidelberg, 2002.

[143] J.A. Snyman. A New and Dynamic Method for Unconstrained Minimization. *Applied mathematical Modelling*, 6(6):449–462, 1982.

[144] J.A. Snyman. An Improved Version of the Original Leap-frog Dynamic Method for Unconstrained Minimization: LFOP1 (b). *Applied mathematical modelling*, 7(3):216–218, 1983.

[145] JA Snyman. An Improved Version of the Original LeapFrog Dynamic Method for Unconstrained Minimization: LFOP1 (b). *Applied Mathematical Modelling*, 7(3):216–218, 1983.

[146] R.V. Southwell. *Relaxation Methods in Theoretical Physics*. Clarendon Press, 1946.

[147] R. Storn and K. Price. Differential Evolution A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[148] P.N. Suganthan. Particle Swarm Optimiser with Neighbourhood Operator. In *Proceedings of the 1999 Congress of Evolutionary Computation*, volume 3, pages 1958–1962, 1999.

[149] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[150] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. MIT Press, A Bradford Book, Cambridge, Massachusetts, United States; London, England, 1998.

[151] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[152] I.C. Trelea. The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection. *Information Processing Letters*, 85(6):317–325, 2003.

[153] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. Phd thesis, University of Pretoria, 2002.

[154] F. van den Bergh. A Study of Particle Swarm Optimization Particle Trajectories. *Information Sciences*, 176(8):937 – 971, 2006.

[155] F. van den Bergh and A.P. Engelbrecht. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. *South African Computer Journal*, 26:84–90, 2000.

[156] F. van den Bergh and A.P. Engelbrecht. Training Product Unit Networks Using Cooperative Particle Swarm Optimizers. In *Joint Conference on Neural Networks*, pages 126 –131, 2001.

[157] F. van den Bergh, A.P. Engelbrecht, and E.S. Peer. Using Neighborhood with the Guaranteed Convergence PSO. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 235–242, 2003.

[158] A.B. van Wyk and A.P. Engelbrecht. Overfitting by PSO Trained Feedforward Neural Networks. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[159] G. Venter and J. Sobieszczanski-Sobieski. Particle Swarm Optimization. *Journal for American Institute of Aeronautics and Astronautics*, 41(8):1583–1589, 2003.

[160] G. Venter and J. Sobieszczanski-Sobieski. Multidisciplinary Optimization of a Transport Aircraft Wing using Particle Swarm Optimization. *Structural and Multidisciplinary Optimization*, 26(1/2):121–131, 2004.

[161] J.S. Vesterstrøm and R. Riget. *Particle Swarms: Extensions for Improved Local, Multi-modal, and Dynamic Search in Numerical Optimization.* Msc thesis, University of Aarhus, 2002.

[162] Alexander Waibel, Toshiuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Speech and Signal*, 37(3):328–339, 1989.

[163] D.J. Watts and S.H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 393(6684):440–442, 1998.

[164] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Phd thesis, Harvard University, 1974.

[165] 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.

[166] U. Witkowski, J. Sitte, S. Herbrechtsmeier, and U. Rückert. AMiRESotA New Robot Soccer League with Autonomous Miniature Robots. *Progress in Robotics*, 44:332–345, 2009.

[167] H. Yoshida, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A Particle Swarm Optimization for Reactive Power and Voltage Control in Electric Power Systems Considering Voltage Security Assessment. In *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, pages 497–502, 1999.

[168] C. Zhang and H. Shao. An ANN's Evolved by a New Evolutionary System and its Application. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 4, pages 3562–3563, 2000.

[169] C. Zhang, H. Shao, and Y. Li. Particle Swarm Optimisation for Evolving Artificial Neural Network. In *Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2487–2490, 2002.

[170] Y. Zheng, L. Ma, L. Zhang, and J. Qian. Empirical Study of Particle Swarm Optimizer with an Increasing Inertia Weight. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 1, pages 221–226, 2003.

[171] Y. Zheng, L. Ma, L. Zhang, and J. Qian. On the Convergence Analysis and Parameter Selection in Particle Swarm Optimization. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1802–1807, 2003.

Appendix A

Acronyms

AN	Artificial Neuron
ANN	Artificial Neural Network
ARPSO	Attractive and Repulsive Particle Swarm Optimisation
CCPSO	Competitive Coevolving Team-based PSO
CCPSO(t)	CCPSO with bounded personal best particle positions
CPSO	Charged Particle Swarm Optimisation
CPSO-S	Cooperative Particle Swarm Optimisation Split
DE	Differential Evolution
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EP	Evolutionary Programming
ES	Evolution Strategies
FIRA	Federation of International Robot-soccer Association
GA	Genetic Algorithm

GCPSO	Guaranteed Convergence Particle Swarm Optimisation
GP	Genetic Programming
HOF	Hall of Fame
IPD	Iterated Prisoner's Dilemma
LVQ	Learning Vector Quantizer
PSO	Particle Swarm Optimisation
QSO	Quantum Swarm Optimisation
SOM	Self-Organizing Feature Map

Appendix B

Symbols

This appendix lists symbols used throughout this study. A separate list of symbols is given for the different chapters of the study. For each chapter, only new symbols are defined.

B.1 Chapter 2: Background

$\vec{x}$	particle position in hyper-dimensional space
$\mathcal{F}(\vec{x})$	fitness function for particle position $\vec{x}$
$\vec{x}_i(t)$	particle i'th position at iteration t
$\vec{x}_{gbest}$	global best particle position
$\vec{x}_{pbest}$	personal best particle position
$\vec{v}_i(t)$	particle i'th velocity at iteration i
ρ_1, ρ_2	constants in velocity update equation
r_1, r_2	uniform random numbers between 0 and 1
c_1, c_2	acceleration constants
t	current iteration
m	neighborhood size
$P(t)$	particle swarm at iteration t
P_i	particle i in particle swarm P

i	particle index
$V_{max,l}$	maximum velocity for dimension l
δ	positive constant, typically between 0 and 1
$\mathcal{K}$	constriction coefficient
ρ	summation of ρ_1 and ρ_2
w	inertia weight
n_t	number of iterations until inertia weight is recalculated
α	positive constant
$\vec{v}_{social(i)}$	social swarm velocity for particle i
$\vec{v}_{cognative(i)}$	cognative swarm velocity for particle i
d_{swarm}	swarm diversity
d_{low}, d_{high}	swarm diversity lower and upper thresholds
$\vec{a}_i$	acceleration constant for particle i
$\vec{a}_{il}$	repulsion force between particle i and l
Q_i	charge magnitude of particle i
R_{core}	core radius
R_{plimit}	perception limit
f	competitive fitness sharing function
K	split factor to split the CPSO-S$_k$ solution vector into K parts
n	number of dimensions
$\vec{i}$	neural network input vector
i_k	k'th input unit
$\vec{o}$	output vector
o_k	k'th output unit
$\vec{h}$	hidden vector
h_k	k'th hidden unit
J	number of input units
K	number of hidden units

f_{AN}	activation function
β, β_1, β_2	positive constants
λ	steepness factor of the function
σ^2	variance of the gaussian distribution
E	sum squared error
$\vec{t}_p$	p'th target vector
$\vec{o}_p$	p'th output vector
K	dimension of target and output vectors
P_T	number of target patterns
$f_{sigmoid}$	sigmoid function
e	Euler's number

B.2 Chapter 3: Simulated Soccer

A_1, A_2	team A player positions 1 and 2
B_1, B_2	team B player positions 1 and 2
$f(north)$	input function for direction north
$f(south)$	input function for direction south
$f(west)$	input function for direction west
$f(east)$	input function for direction east
Δx	distance along the east-west direction
Δy	distance along the north-south direction
d	distance

B.3 Chapter 4: Cooperative Competitive Coevolution with Charged PSO

s	player position, population number or particle swarm number
$O_s(t)$	particle swarm for player position s at iteration t

$c_{re-evaluate}$	particle fitness re-evaluation modulus constant
M	Messerschmidt performance measure
F	Franken performance measure
S	modified Franken performance measure
f	outcome probability as measured against N random opponents
N	positive constant
n_{won}	number of games won
n_{total}	number of games played
R	random function
$fanin$	number of incoming connections for a neuron
V_{max}	maximum velocity
P_{core}	perception core radius
P	perception limit

B.4 Chapter 5: Relative Fitness

$L'(i)$	points scored in iteration i
$f_{goal-A}(t)$	reward function for team A scoring at iteration t
$f_{goal-B}(t)$	reward function for team B scoring at iteration t
$f_{goal-A}(t)$	reward function for team A possessing the ball at iteration t
$f_{goal-B}(t)$	reward function for team B possessing the ball at iteration t
t_{final}	number of iterations until a goal is scored or 30 otherwise
F_{ramp}	rampup absolute fitness
W_pP_p	weighted points awarded for field distribution
$W_{bd}P_{bd}$	weighted points awarded for distance to the ball
W_kP_k	weighted points awarded for number of kicks
$W_{fk}P_{fk}$	weighted points awarded for number of out of bound kicks
$W_{bg}P_{bg}$	weighted points awarded for ball's distance to the goal
W_sP_s	weighted points awarded for number of goals scored

$\mathcal{G}_n$	number of goals scored in the n'th competition
$\mathcal{O}_n$	number of goals scored by the opposition in the n'th competition
$\mathcal{F}$	relative fitness function
$\mathcal{F}(t)$	relative fitness function for iteration t
P	points awarded per match
M	match points
I	importance of match indicator
T	opposition team strength based on their rank
C	strength of the confederation
$\mathcal{P}(t)$	points awarded at iteration t
$\mathcal{T}_n$	n'th opposition team rank
$\mathcal{M}_n$	match outcome points
$r(p)$	rank of player p

B.5 Chapter 7: Improving performance

$\Phi(t)$	global best particle position movement from iteration $t-1$ to t
$X_y(z)$	team X, position y, cluster centroid number z

Appendix C

CILib Simulation Definitions

This appendix lists the CILib XML simulation definitions. Each simulation consists of four sections. A problem definition representing the simple soccer game, the neural network controllers, and the scoring mechanism. An algorithm definition defining the training algorithm along with its control parameters. A measurement definition specifying all the measurements. Finally, the simulation definition brings the first three sections together. Variations of the different sections are also listed.

C.1 Problem

```
<problem id="SimpleSoccer" class="problem.coevolution.
CompetitiveCoevolutionGameLearningOptimizationProblem"
numberOfEvaluations="1">
 <game class="games.game.simplesoccer.RealTimeSimpleSoccerGame">
  <agent class="games.agent.NeuralAgent"
  AgentToken="SimpleSoccer.PLAYER_A1">
   <neuralNetwork class="nn.NeuralNetwork">
    <architecture class="nn.architecture.Architecture">
     <architectureBuilder class="nn.architecture.builder.
     FeedForwardArchitectureBuilder">
      <addLayer class="nn.architecture.builder.LayerConfiguration"/>
```

```
      <addLayer class="nn.architecture.builder.LayerConfiguration"
      size="5">
       <ActivationFunction class="functions.activation.TanH"/>
      </addLayer>
      <addLayer class="nn.architecture.builder.LayerConfiguration">
       <ActivationFunction class="functions.activation.TanH"/>
      </addLayer>
      <layerBuilder class="nn.architecture.builder.
      PrototypeFullyConnectedLayerBuilder" domain="R(-1,1)" />
     </architectureBuilder>
    </architecture>
   </neuralNetwork>
   <stateInputStrategy class="games.game.simplesoccer.
   BallOwnerDistanceNeuralSimpleSoccerInputStrategy"/>
   <outputInterpretationStrategy class="games.game.simplesoccer.
   NeuralSimpleSoccerOutputStrategy">
   </outputInterpretationStrategy>
  </agent>
  <agent class="games.agent.NeuralAgent"
  AgentToken="SimpleSoccer.PLAYER_A2">
   <neuralNetwork class="nn.NeuralNetwork">
    <architecture class="nn.architecture.Architecture">
     <architectureBuilder class="nn.architecture.builder.
     FeedForwardArchitectureBuilder">
      <addLayer class="nn.architecture.builder.LayerConfiguration"/>
      <addLayer class="nn.architecture.builder.LayerConfiguration"
      size="5">
       <ActivationFunction class="functions.activation.TanH"/>
      </addLayer>
      <addLayer class="nn.architecture.builder.LayerConfiguration">
       <ActivationFunction class="functions.activation.TanH"/>
```

```
      </addLayer>
      <layerBuilder class="nn.architecture.builder.
      PrototypeFullyConnectedLayerBuilder" domain="R(-1,1)" />
     </architectureBuilder>
    </architecture>
   </neuralNetwork>
   <stateInputStrategy class="games.game.simplesoccer.
   BallOwnerDistanceNeuralSimpleSoccerInputStrategy"/>
   <outputInterpretationStrategy class="games.game.simplesoccer.
   NeuralSimpleSoccerOutputStrategy">
   </outputInterpretationStrategy>
  </agent>
  <agent class="games.agent.NeuralAgent"
  AgentToken="SimpleSoccer.PLAYER_B1">
   <neuralNetwork class="nn.NeuralNetwork">
    <architecture class="nn.architecture.Architecture">
     <architectureBuilder class="nn.architecture.builder.
     FeedForwardArchitectureBuilder">
      <addLayer class="nn.architecture.builder.LayerConfiguration"/>
      <addLayer class="nn.architecture.builder.LayerConfiguration"
      size="5">
       <ActivationFunction class="functions.activation.TanH"/>
      </addLayer>
      <addLayer class="nn.architecture.builder.LayerConfiguration">
       <ActivationFunction class="functions.activation.TanH"/>
      </addLayer>
      <layerBuilder class="nn.architecture.builder.
      PrototypeFullyConnectedLayerBuilder" domain="R(-1,1)" />
     </architectureBuilder>
    </architecture>
   </neuralNetwork>
```

```
 <stateInputStrategy class="games.game.simplesoccer.
 BallOwnerDistanceNeuralSimpleSoccerInputStrategy"/>
 <outputInterpretationStrategy class="games.game.simplesoccer.
 NeuralSimpleSoccerOutputStrategy">
 </outputInterpretationStrategy>
</agent>
<agent class="games.agent.NeuralAgent"
AgentToken="SimpleSoccer.PLAYER_B2">
 <neuralNetwork class="nn.NeuralNetwork">
  <architecture class="nn.architecture.Architecture">
   <architectureBuilder class="nn.architecture.builder.
   FeedForwardArchitectureBuilder">
    <addLayer class="nn.architecture.builder.LayerConfiguration"/>
    <addLayer class="nn.architecture.builder.LayerConfiguration"
    size="5">
     <ActivationFunction class="functions.activation.TanH"/>
    </addLayer>
    <addLayer class="nn.architecture.builder.LayerConfiguration">
     <ActivationFunction class="functions.activation.TanH"/>
    </addLayer>
    <layerBuilder class="nn.architecture.builder.
    PrototypeFullyConnectedLayerBuilder" domain="R(-1,1)" />
   </architectureBuilder>
  </architecture>
 </neuralNetwork>
 <stateInputStrategy class="games.game.simplesoccer.
 BallOwnerDistanceNeuralSimpleSoccerInputStrategy"/>
 <outputInterpretationStrategy class="games.game.simplesoccer.
 NeuralSimpleSoccerOutputStrategy">
 </outputInterpretationStrategy>
</agent>
```

C.1.1 Fixed Reward

```
  <scoringStrategy class="games.game.simplesoccer.score.
  SimpleSoccerGameScoringStrategy"/>
  <gameEndCondition class="games.game.simplesoccer.score.
  SimpleSoccerGameMaxIterationsEndCondition" MaxIterations="50"/>
 </game>
</problem>
```

C.1.2 Goal Difference

```
  <scoringStrategy class="games.game.simplesoccer.score.
  SimpleSoccerGoalDifferenceScoringStrategy"/>
  <gameEndCondition class="games.game.simplesoccer.score.
  SimpleSoccerGameMaxIterationsEndCondition" MaxIterations="50"/>
 </game>
</problem>
```

C.1.3 FIFA League Ranking

```
  <scoringStrategy class="games.game.simplesoccer.score.
  SimpleSoccerFIFALeagueRankingScoringStrategy"/>
  <gameEndCondition class="games.game.simplesoccer.score.
  SimpleSoccerGameMaxIterationsEndCondition" MaxIterations="50"/>
 </game>
 <fitnessCalculation class="games.game.simplesoccer.score.
 FIFALeagueRankingFitnessCalculationStrategy"/>
</problem>
```

C.2 Algorithm

C.2.1 Original

```
<algorithms>
 <algorithm id="pso" class="pso.PSO">
  <iterationStrategy class="pso.dynamic.DynamicIterationStrategy">
  <IterationStrategy class="pso.iterationstrategies.
  ASynchronousIterationStrategy"/>
   <DetectionStrategy class="pso.dynamic.detectionstrategies.
   PeriodicDetectionStrategy" IterationsModulus="3"/>
   <ResponseStrategy class="pso.dynamic.responsestrategies.
   CompetitiveCoevolutionParticleReevaluationResponseStrategy"/>
  </iterationStrategy>
  <initialisationStrategy class="algorithm.initialisation.
  ClonedPopulationInitialisationStrategy" entityNumber="20">
   <entityType class="pso.dynamic.ChargedParticle">
    <neighbourhoodBestUpdateStrategy class="pso.positionupdatestrategies.
    IterationNeighbourhoodBestUpdateStrategy"/>
    <velocityUpdateStrategy class="pso.dynamic.
    ChargedVelocityUpdateStrategy">
     <vMax class="controlparameter.ConstantControlParameter"
     parameter="15.0"/>
     <pCore class="controlparameter.ConstantControlParameter"
     parameter="2"/>
     <p class="controlparameter.ConstantControlParameter"
     parameter="500"/>
    </velocityUpdateStrategy>
    <chargedParticleInitialisationStrategy class="pso.dynamic.
    StandardChargedParticleInitialisationStrategy"
    ChargeMagnitude = "15.0"/>
   </entityType>
```

```
  </initialisationStrategy>
  <topology class="entity.topologies.VonNeumannTopology"/>
 </algorithm>
 <algorithm id="compcoevol" class="coevolution.CoevolutionAlgorithm">
  <addStoppingCondition class="stoppingcondition.MaximumIterations"
  maximumIterations="2000"/>
  <coevolutionIterationStrategy class="coevolution.
  CompetitiveCoevolutionIterationStrategy">
   <opponentSelectionStrategy class="coevolution.selection.
   SelectNOpponentSelectionStrategy" numberOfOpponents="15">
    <addPoolSelectionStrategy class="coevolution.selection.
    SelectpBestSolutionsPoolSelectionStrategy"/>
    <addPoolSelectionStrategy class="coevolution.selection.
    SelectHOFPoolSelectionStrategy" HOFSize="4" AddToHOFEpoch="30"/>
   </opponentSelectionStrategy>
  </coevolutionIterationStrategy>
 </algorithm>
</algorithms>
```

C.2.2 Bounded Personal Best

```
<algorithms>
 <algorithm id="pso" class="pso.PSO">
  <iterationStrategy class="pso.dynamic.DynamicIterationStrategy">
  <IterationStrategy class="pso.iterationstrategies.
  ASynchronousIterationStrategy"/>
   <DetectionStrategy class="pso.dynamic.detectionstrategies.
   PeriodicDetectionStrategy" IterationsModulus="3"/>
   <ResponseStrategy class="pso.dynamic.responsestrategies.
   CompetitiveCoevolutionParticleReevaluationResponseStrategy"/>
  </iterationStrategy>
  <initialisationStrategy class="algorithm.initialisation.
```

```
ClonedPopulationInitialisationStrategy" entityNumber="20">
 <entityType class="pso.dynamic.ChargedParticle">
  <personalBestUpdateStrategy class="pso.positionupdatestrategies.
  BoundedPersonalBestUpdateStrategy"/>
  <neighbourhoodBestUpdateStrategy class="pso.positionupdatestrategies.
  IterationNeighbourhoodBestUpdateStrategy"/>
  <velocityUpdateStrategy class="pso.dynamic.
  ChargedVelocityUpdateStrategy">
   <vMax class="controlparameter.ConstantControlParameter"
   parameter="15.0"/>
   <pCore class="controlparameter.ConstantControlParameter"
   parameter="2"/>
   <p class="controlparameter.ConstantControlParameter"
   parameter="500"/>
  </velocityUpdateStrategy>
  <chargedParticleInitialisationStrategy class="pso.dynamic.
  StandardChargedParticleInitialisationStrategy"
  ChargeMagnitude = "15.0"/>
 </entityType>
</initialisationStrategy>
<topology class="entity.topologies.VonNeumannTopology"/>
</algorithm>
<algorithm id="compcoevol" class="coevolution.CoevolutionAlgorithm">
<addStoppingCondition class="stoppingcondition.MaximumIterations"
maximumIterations="2000"/>
<coevolutionIterationStrategy class="coevolution.
CompetitiveCoevolutionIterationStrategy">
 <opponentSelectionStrategy class="coevolution.selection.
 SelectNOpponentSelectionStrategy" numberOfOpponents="15">
  <addPoolSelectionStrategy class="coevolution.selection.
  SelectpBestSolutionsPoolSelectionStrategy"/>
```

```
    <addPoolSelectionStrategy class="coevolution.selection.
    SelectHOFPoolSelectionStrategy" HOFSize="4" AddToHOFEpoch="30"/>
   </opponentSelectionStrategy>
  </coevolutionIterationStrategy>
 </algorithm>
</algorithms>
```

C.2.3 Linear Decreasing R_{core} and Bounded Personal Best

```
<algorithms>
 <algorithm id="pso" class="pso.PSO">
  <iterationStrategy class="pso.dynamic.DynamicIterationStrategy">
  <IterationStrategy class="pso.iterationstrategies.
  ASynchronousIterationStrategy"/>
   <DetectionStrategy class="pso.dynamic.detectionstrategies.
   PeriodicDetectionStrategy" IterationsModulus="3"/>
   <ResponseStrategy class="pso.dynamic.responsestrategies.
   CompetitiveCoevolutionParticleReevaluationResponseStrategy"/>
  </iterationStrategy>
  <initialisationStrategy class="algorithm.initialisation.
  ClonedPopulationInitialisationStrategy" entityNumber="20">
   <entityType class="pso.dynamic.ChargedParticle">
    <personalBestUpdateStrategy class="pso.positionupdatestrategies.
    BoundedPersonalBestUpdateStrategy"/>
    <neighbourhoodBestUpdateStrategy class="pso.positionupdatestrategies.
    IterationNeighbourhoodBestUpdateStrategy"/>
    <velocityUpdateStrategy class="pso.dynamic.
    ChargedVelocityUpdateStrategy">
     <vMax class="controlparameter.ConstantControlParameter"
     parameter="15.0"/>
     <pCore class="controlparameter.LinearDecreasingControlParameter"
     upperBound="10.0" decreaseByIteration="250"/>
```

```
      <p class="controlparameter.LinearDecreasingControlParameter"
      upperBound="1000.0" decreaseByIteration="250"/>
     </velocityUpdateStrategy>
     <chargedParticleInitialisationStrategy class="pso.dynamic.
     StandardChargedParticleInitialisationStrategy"
     ChargeMagnitude = "15.0"/>
    </entityType>
   </initialisationStrategy>
   <topology class="entity.topologies.VonNeumannTopology"/>
  </algorithm>
  <algorithm id="compcoevol" class="coevolution.CoevolutionAlgorithm">
   <addStoppingCondition class="stoppingcondition.MaximumIterations"
   maximumIterations="2000"/>
   <coevolutionIterationStrategy class="coevolution.
   CompetitiveCoevolutionIterationStrategy">
    <opponentSelectionStrategy class="coevolution.selection.
    SelectNOpponentSelectionStrategy" numberOfOpponents="15">
     <addPoolSelectionStrategy class="coevolution.selection.
     SelectpBestSolutionsPoolSelectionStrategy"/>
     <addPoolSelectionStrategy class="coevolution.selection.
     SelectHOFPoolSelectionStrategy" HOFSize="4" AddToHOFEpoch="30"/>
    </opponentSelectionStrategy>
   </coevolutionIterationStrategy>
  </algorithm>
 </algorithms>
```

C.3 Measurement

```
<measurements id="fitness" class="simulator.MeasurementSuite"
resolution="100">
 <addMeasurement class="measurement.generic.Time"/>
 <addMeasurement class="measurement.generic.DateTime"/>
 <addMeasurement class="measurement.multiple.
 MultiPopulationFitness"/>
 <addMeasurement class="measurement.multiple.
 LeeJeongSimpleSoccerTeamFitness"/>
 <addMeasurement class="measurement.multiple.
 TemplateOpponentGameMeasurement" numberOfEvaluations="15000">
  <agent class="games.game.simplesoccer.RandomAgent"
  AgentToken="SimpleSoccer.PLAYER_B1"/>
  <agent class="games.game.simplesoccer.RandomAgent"
  AgentToken="SimpleSoccer.PLAYER_B2"/>
  <addGameMeasurement class="measurement.game.GameMeasurementGroup"
  AgentToken="SimpleSoccer.PLAYER_A1">
   <addAgentMeasurement class="measurement.game.
   WinDrawLossFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   StandardDeviationFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   MinimumFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   MaximumFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   AverageFitnessGameMeasurement"/>
  </addGameMeasurement>
 </addMeasurement>
 <addMeasurement class="measurement.multiple.
```

```
 TemplateOpponentGameMeasurement" numberOfEvaluations="15000">
  <agent class="games.game.simplesoccer.RandomAgent"
  AgentToken="SimpleSoccer.PLAYER_A1"/>
  <agent class="games.game.simplesoccer.RandomAgent"
  AgentToken="SimpleSoccer.PLAYER_A2"/>
  <addGameMeasurement class="measurement.game.GameMeasurementGroup"
  AgentToken="SimpleSoccer.PLAYER_B1">
   <addAgentMeasurement class="measurement.game.
   WinDrawLossFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   StandardDeviationFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   MinimumFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   MaximumFitnessGameMeasurement"/>
   <addAgentMeasurement class="measurement.game.
   AverageFitnessGameMeasurement"/>
  </addGameMeasurement>
 </addMeasurement>
</measurements>
```

C.4 Simulation

```
<?xml version="1.0"?>
<!DOCTYPE simulator [<!ATTLIST algorithm id ID #IMPLIED>
<!ATTLIST problem id ID #IMPLIED><!ATTLIST measurements id ID #IMPLIED>]>

<simulator>
 <!-- Insert problem definition here -->
 <!-- Insert algorithms definition here -->
 <!-- Insert measurements definition here -->
 <simulations>
  <simulation samples="1">
   <algorithm idref="compcoevol">
    <algorithm idref="pso"/>
    <algorithm idref="pso"/>
    <algorithm idref="pso"/>
    <algorithm idref="pso"/>
   </algorithm>
   <problem idref="SimpleSoccer"/>
   <measurements idref="fitness" />
   <output format="TXT" file="measurements.txt"/>
  </simulation>
 </simulations>
</simulator>
```