Universidad Politecnica de Madrid
Facultad de Informatica

Co-evolutionary and Reinforcement Learning Techniques Applied to Computer Go players

A thesis submitted for the degree of

Doctor of Philosophy in Computer Science - Artificial Intelligence

Author:
Msc. Wester Edison Zela Moraya

Directors:
Dr. Maria Aurora Martinez Rey
Dr. Jose Gabriel Zato Recellado

June 2013

I would like to dedicate this thesis to my loving parents, Wester and Loyola, to Marita for hers support all of this time to complete this dissertation, and Nicole which hers smile gave me more strength to finalize the thesis.

Acknowledgements

I want to thank my thesis directors: Dr. Jose Gabriel Zato and Dra. Maria Aurora Martinez. To Dr. Jose Zato for his support and advices during all of these years from the moment I was part of his course in the program, and in the last years providing me some facilities as an office in the university to complete this thesis, and Dra. Maria Aurora, which support me in the last year with hers reviews and contributions to this thesis.

To Dr. Juan Pazos who was interested in the topic of my thesis and provided some feedbacks to enrich more this thesis.

To my brothers Paul, who is studying a Master of Artificial Inteligence, and Aurora, who is studying biology, with whom I discussed some topics to realize by myself if some points discussed in the thesis made sense.

To all the proffesors of the programs from which I got introduced to the Artificial Intelligence.

Abstract

The objective of this thesis is model some processes from the nature as evolution and co-evolution, and proposing some techniques that can ensure that these learning process really happens and useful to solve some complex problems as Go game.

The Go game is ancient and very complex game with simple rules which still is a challenge for the Artificial Intelligence. This dissertation cover some approaches that were applied to solve this problem, proposing solve this problem using competitive and cooperative co-evolutionary learning methods and other techniques proposed by the author.

To study, implement and prove these methods were used some neural networks structures, a framework free available and coded many programs. The techniques proposed were coded by the author, performed many experiments to find the best configuration to ensure that co-evolution is progressing and discussed the results.

Using co-evolutionary learning processes can be observed some pathologies which could impact co-evolution progress. In this dissertation is introduced some techniques to solve pathologies as loss of gradients, cycling dynamics and forgetting. According to some authors, one solution to solve these co-evolution pathologies is introduce more diversity in populations that are evolving. In this thesis is proposed some techniques to introduce more diversity and some diversity measurements for neural networks structures to monitor diversity during co-evolution. The genotype diversity evolved were analyzed in terms of its impact to global fitness of the strategies evolved and their generalization. Additionally, it was introduced a memory mechanism in the network neural structures to reinforce some strategies in the genes of the neurons evolved with the intention that some good strategies learned are not forgotten.

In this dissertation is presented some works from other authors in which cooperative and competitive co-evolution has been applied. The Go board size used in this thesis was 9×9, but can be easily escalated

to more bigger boards.The author believe that programs coded and techniques introduced in this dissertation can be used for other domains.

Resumen

El objetivo de la tesis es modelar algunos procesos de la naturaleza como la evolución y coevolución, proponer algunas técnicas que puedan asegurar estos procesos de aprendizaje realmente sucedan y sea útiles para resolver problemas complejos como el juego del Go.

El juego del Go es juego antiguo y muy complejo con reglas simples que sigue siendo un desafío para la Inteligencia Artificial. Esta tesis cubre algunos enfoques que se han aplicado para resolver este problema, y se propone resolverlo mediante metodos de aprendizaje con la co-evolución competitiva y cooperativa y otras técnicas propuestas por el autor.

Para desarrollar, implementar y probar estas teorías se han utilizado distintas estructuras de redes neuronales, un framework disponible para competir los jugadores Go desarrollados y donde además se codificaron algunos programas. Estas técnicas se codificaron por el autor, se realizaron varios experimentos para encontrar la mejor configuración para que la co-evolución progrese y se discutieron los resultados obtenidos.

Utilizando el aprendizaje coevolutivo se puede observar algunas patologías que afectan a que la coevolución progrese. En este trabajo se introduce técnicas para resolver algunas patologías de coevolución como la pérdida del gradiente o declopamiento, dinámica ciclicas y el olvido. Según algunos autores, una de las soluciones para resolver estas patologías es la introducción de diversidad en las poblaciones que evolucionan. En esta tesis es introducida una técnica para medir la diversidad del genotipo de las redes neuronales con la intensión de introducir mas diversidad en la poblaciones cuando sea necesario. Además la diversidad del genotipo fue analizada en términos de su impacto en el fitness global de las estrategias evolucionadas y su grado de generalización. Adicionalmente fue introducida un mecanismo de memoria para las redes neuronales para reforzar ciertas estrategias en los genes de las neuronas que evolucionan con la intensión de que las buenas estrategias aprendidas no sean olvidadas.

En esta tesis se presenta algunos trabajos en los que se ha aplicado coevolución cooperativa y competitiva que luego son discutidas . El tamaño del tablero utilizado para realizar los experimentos en este trabajo fue 9×9, pero puede ser fácilmente escalada a tableros de mayor dimensión. El autor sugiere que los programas codificados y técnicas introducidas en este trabajo se pueden utilizar para otros dominios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation of Thesis

When I was child I was fan of some board games as Chess, Checkers, Monopoly and others, and some computer games as Pac-Man, Space invaders and other arcade games. In the last years after the invasion of many hardware platforms and computer games I played sometimes games as Age of Empires, Star Wars: Galatic Battleground, and other real-time strategy games.

During my courses of the first year in this program, I realized that one of the problems in the computer game industry was that in general the Artificial Intelligence (AI) of the industry was not developed as it was the graphics and other features of computer games, and basically the "intelligence" of these games were written in scripts. So, this gave me an motivation that probably I can contribute providing some techniques in the AI arena that can create agents that have more credible "intelligence" and at the same time less expensive, avoiding the necessity of write scripts by experts in the domain of the problem.

Fortunately, I found a course in the program that gave me the opportunity in which I started exploring methods and techniques that are used in Artificial Intelligence in computer games. This course was called "Challenge to the AI: Go game", and from there I started to research more seriously in this area.

I have many motivations to complete this thesis, and initialize when I started my intention was create a outstanding computer Go players using the neuro-evolution and co-evolution techniques but during the implementation, experiments and observing the results, the motivations of this thesis were evolving. So, I can say that work in this thesis was useful to understand how the evolution process works or at least how should works to create more complex strategies, or in other words how can create more complex intelligence. So, to ensure this natural process happens I was needed to introduce some techniques in this thesis.

In the last year it was presented a work in an international event which was

accepted Zela & Zato [2011] and currently the author is working in next paper which is going to be present the findings discussed in this thesis.

So, I can summarize my objectives in this Thesis as the followings:

- Model the co-evolution process from the nature using agents that cooperate and compete to solve some complex problems as Go game which still is considered a challenge for Artificial Intelligence. The intention of this thesis is show some evidences that these natural process can be simulated and be useful in some complex problems.
- Show some evidences that techniques proposed by the author are useful for co-evolution learning process. So, in this thesis is discussed that to ensure that co-evolution learning happens it was needed these techniques to avoid some co-evolutionary pathologies or some dynamics as red-queen.
- Some authors suggest that keep diversity in the population can avoid some co-evolution pathologies. So, in this thesis it was introduced some techniques to keep the diversity of the population during evolution process.
- Measurement of the generalization of strategies learned during this process is good driver to measure what general solution are discovered and if strategies learned can be useful in other environments in which the populations were not trained.
- Discuss about techniques for monitoring whether co-evolution is progressing and learning more complex strategies, in other words whether the global fitness is growing, and testing these solutions against a external agents. To prove this was used known computer Go players which can be used to observe whether more complex strategies are learned.

1.2 Major Contribution of Thesis

In this thesis was introduced some techniques that were useful to ensure the evolution process. These techniques are:

- Competitive Fitness Sharing Augmented (CFSA): One of the contribution is to propose to use CFSA as fitness sharing function which is based on Competitive Fitness Sharing (CFS) proposed by Rosin & Belew [1997]. This function considers number of games wins and lost by players after competitions against opponents with the intention to maintain the phenotype diversity of solutions learned.

- Dynamic Sizing of the artificial neural network or neural network structures: As it is discussed in this thesis the use of fix size of neural networks can come with two issues, the first one is how to decide what should be the best number of neurons in a neural network?, and the second issue is about as it is discussed by other authors that network structures as SANE Moriarty & Miikkulainen [1996a] contains some redundancy because irrelevant connections that can be formed. So, the intention to introduce this technique is that the evolution process itself can find the best sizes for the networks structures for the computer Go players under the pressure of the evolution process.

- Replacement Immigration Rate (RIR): The intention to introduce this technique is control the rate of the flow of artificial neuron immigrants to the population of neurons that are co-evolving and can be calculated automatically based on the results that are been obtained from the competition of different populations. This technique will introduce more diversity when is needed and adjusted automatically.

- Introduce a memory mechanism: It is introduced a memory mechanism for computer Go players to reinforce good strategies in the genes of these computer players, and all players which share neurons with these genes, and keep it for long time.

- Refinements to the fitness functions: Based on the results of the experiment was needed to introduce more complex fitness functions that can ensure the evolution of populations. Some of the fitness functions tested were CFSA, CFS, and others which included the mentioned functions and adding other features of the game as average scores obtained and number of games won during competitions. Other refinement to the fitness function included was that players of Hall of Fame can be take in account the fitness obtained in the previous generations, this technique will ensure that some strategies are not lost after the application of selection methods in next generations.

- Measurement of Diversity and Generalization of the strategies evolved: In this thesis has been proposed based on previous works from other authors a way to measure the diversity of the artificial neural networks structures evolved. There are some authors, which are going to be discussed in this thesis, who propose that to keep the generalization of the strategies evolved the population diversity is a key factor. So, in this thesis is applied some techniques to keep the diversity of the population during the co-evolution process and it is calculated the generalization of the strategies learned.

1.3 Summary of the Chapters

This thesis is divided in eight chapters. This starts with the Chapter One which describe the motivations, objectives and major contributions of this thesis which was described already.

The Chapter Two introduce to readers to the history how computer games as been a testbed of different Artificial Intelligence techniques. It gives a brief description how the complexity of some board games are calculated, and different AI techniques that has been applied to solve the Go game. This chapter introduce the Go game presenting the history of the game and brief description of the rules and some terms of the game. This chapter finalize with the review of the state of the art of computer games in Go game, reviewing the different techniques that has been applied to solve this game.

The Chapter Three introduces the theory about the evolution and co-evolutionary learning process. It starts with the reviews of some definitions as evolution, co-evolutions and some dynamics that can be observed in these processes as red-queen dynamics. The chapter continue with the review of some definitions as evolutionary computation, and competitive and cooperative co-evolution, describing the progress that have been observed in that areas. Later, it is discussed the advantage of using co-evolutionary techniques as a learning technique, and describes the pathologies that can be found in a co-evolutionary process, and why it is needed to be monitored and measured the co-evolution progress. Finally, in this chapter is introduced how solution concepts are defined in co-evolution environment, and some of fitness measures used in co-evolutionary learning processes.

The Chapter Four presents two works from other authors in which co-evolutionary learning is proposed as an alternative search method for difficult problems in which traditional approaches are difficult to solve. The first work is about the simulation of co-evolution of predators and preys, which is a case of pursuit and evasion problem. The second work is the application of the co-evolution to discover complex strategies for the trading of some securities in the capital markets.

The Chapter Five starts with the definition of the solution concept for the co-evolution of computer Go players, and continue with the discussion of some approaches to solve this game using evolutionary approaches, as dividing the board using the symmetry of the board. Later, it presents the contribution of the author to solve the co-evolutionary pathologies as introducing dynamic size structures of artificial neural networks, immigrant rates to control of flows of artificial neurons, new competitive fitness sharing called CFSA (Competitive Fitness Sharing Augmented), and some refinements to the fitness measures. In this chapter is introduced the memory mechanism for players to introduce in the genes some strategies learned in the competition. Later, this chapter describes what how the diversity is measured and how to monitor is the co-evolution of

computer Go players are progressing based on generalization and other measures. Finally this chapter ends with a discussion about the evaluation functions used in this thesis to ensure the co-evolution of two competing populations.

The Chapter Six presents the results of the experiments performed of evolving and co-evolving Computer Go players in 9×9 boards applying the techniques discussed previously. The chapter starts with the description of the architecture implemented and the results of the experiments in the evolution of a Computer Go player using techniques proposed against known computer Go player. It discusses the results of the evolution of black and white population players, analyzed the impact of the techniques proposed and coded by the author. In the next sections present and discussed the results of the experiments obtained for the co-evolution of two population using the techniques discussed in the previous chapters. Finally the chapter is ending with the analysis and discussion of the diversity and generalization rates obtained during the experiments, and if the co-evolution pathologies were mitigated in the co-evolution processes.

The Chapter Seven discusses the conclusion of the thesis and Chapter Eight discusses about the future actions and other possible applications of the techniques proposed in this thesis.

Chapter 2

Background

This chapter starts with some discussion about how the artificial intelligence research community has used the games as testbed during the history. In the section 2.2 describe briefly some key points of the game theory. In the section 2.3 describe the Go game, some rules and some terms used in this game. In the section 2.3 describe how the complexity of the games are calculated. In the section 2.4 is presented an overview of the techniques applied to the Go game, for instance, tree search techniques, combinatorial game theory, Montecarlo and other techniques.

In section 2.5 is reviewed the artificial neural networks and some Neuro-evolution techniques that will be useful for this thesis, and finally the section 2.7 discuss about the state of the art of the computer Go and some comments about the contributions of the author in this thesis.

2.1 AI and Games

Since the beginning of Artificial Intelligence (AI), mind games, such as Checkers, Chess, Poker, Chinook and others, have been studied as application fields for AI. For example, according to Copeland [2004], in 1948, working with his former undergraduate colleague, DG Champernowne, Alan Turing, who is considered a father of the modern Computer Science and Artificial Intelligence, began writing a Chess program for a computer that did not yet exist and, in 1952, lacking a computer powerful enough to execute the program, played a game in which he simulated it, taking about half an hour over each move.

After Alan Turing and the beginning of the modern computing era, the computer games are used to test new proposed AI algorithms. There are many game playing computer programs than have reached an expert level using a search-based approach. For example, in Chess this approach achieved a very good results and

produced a big impact around the world, when the World Champion Kasparov was defeated by IBM Deep Blue in the 1997 exhibition, actually the machine, with human intervention between games, won the second six-game match against world champion by two wins to one with three draws as is mentioned in the article Chess Bump by W. Saletan. The search-based approach was called brute force, because the IBM computer was capable of evaluating 200 million positions of the game per second. Nowadays, some computer programs are better than human players in most classical games as Deep Blue in Chess, Chinook in Checkers, Logistello in Othello, Victoria in Go-moku as it was described by Allis [1994], Bouzy & Cazenave [2001] and other authors.

As Pazos [1987] described, one of the reasons why the AI research community used games as research field is because, in the majority of the times, real problems are fuzzy that generally is difficult to express them in a acceptable way for the computers, by contrary, games are well formulated, but at the same time simulate real problems, the nature of the games are not numeric with not deterministic behaviors.

The games has been considered important matter for mathematicians and computer scientist to develop and test the performance of computational techniques. As for example, when Babbage developed his machine he though in the possibility that this can solve some of this type as Chess. Turing was one of the first to write about this in his article "Digital Computers Applied to Games", as he mentions "there is not excuse, or we needed, or we have to use it." Pazos [1987].

For other games as Go, in the last years have been make good progress but still yet the computer programs needs some handicap to beat professional players.

Go is a popular ancient board game, played by an estimated 25 to 50 million players, in many countries and with official competitions around the world. According to some authors as Allis [1994], Go is by far the most complex popular board game in the class of two-player perfect-information games .

The Go game still is a challenge for the Artificial Intelligence, because in spite it has few rules, the number of games and strategies that can be played it is huge. As Allis [1994], Bouzy & Cazenave [2001] and other authors discussed, the application of only known Tree Search techniques is not useful, new techniques or algorithms has to be proposed to create computer programs with more high level of play.

In the last years were explored different techniques to create good computer Go players as described by Bouzy & Cazenave [2001], Brugmann [1993], Muller [1995], Enzenbergerl [1996], Moriarty & Miikkulainen [1998b], van der Werf [2004] and others which is discussed in this thesis, but in most of the cases in small boards. In general, AI research community have been used games as testbed for AI techniques to explore the potential of these techniques as combinatorial

game theory in Muller [1999] in Go, Montecarlo techniques as Chaslo [2010] and other games, Temporal Difference Learning (TDL) in Runarsson & Lucas [2005] in Go, Lucas & Runarsson [2006] in Othelo , Kotnik & Kalita [2003] in Gin rummy, Darwen [2001] in Backgammon, some tree searching techniques reviewed at van der Werf [2004], Neuro-Evolution techniques as Chellapilla & Fogel [1999] in checkers, Particle Swarm Optimization (PSO) co-evolutionary techniques in Franken [2004], Papacostantis *et al.* [2005] in tic-tac-toe and checkers, Burrow & Lucas [2009] using TDL and co-evolution in MS pac-man and other games.

Annually in the world there are computer game competitions or congresses, maybe the most famous is the Computer Olympiad which is organized by the International Computer Games Association (ICGA). Go is part of these competitions in board of 9×9, 13×13 and 19×19.

There are other formal competitions that occur less formal as KGS Go Server, and Computer Go Server.

For the human-computer competitions there are annually Go congresses around the world where the computer Go programs compete against professional players. Some of these competitions are European Go congress, IEEE WCCI, JAIST Cup, Computer Go UEC Cup and others.

2.2 Game Theory

According to Turocy & Stengel [2002] Game theory is the formal study of conflict and cooperation where game theoretic concepts apply whenever the actions of several agents are interdependent. The agents in this theory can be individuals, groups, firms, or any combination of these. The concepts of game theory provide a language to formulate, structure, analyze, and understand strategic scenarios.

The object of study game theory is the game, which is a formal model of an interactive situation. The game typically involves several players; in case of only one player is called a decision problem, or games against nature. The formal definition describe the players, their preferences, their information, the strategic actions available to them, and how these influence the outcome Turocy & Stengel [2002]. The earliest example of a formal game-theory analysis is the study of a duopoly at 1838, and from there this theory has been applied to diverse areas as economics, war, politics, sociology, psychology and others.

One of the main assumptions in game theory is the players are rational, meaning that the decision made is based in the best outcome that he can obtain, given what his opponent can do. So, the goal of game-theoretic analysis is to predict how the game will be played by rational players, or, relatedly, to give advice on how best to play the game against opponents who are rational Turocy & Stengel [2002].

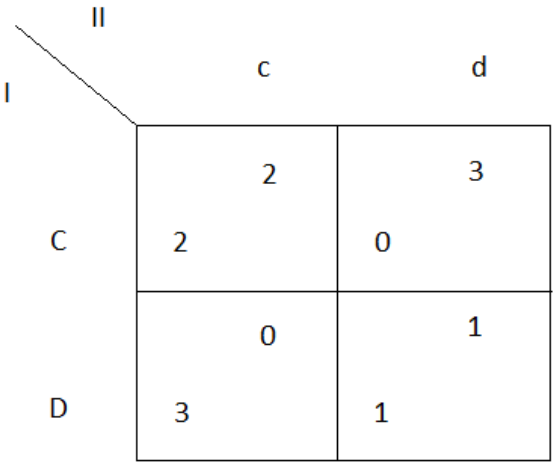


Figure 2.1: The Prisoner's dilemma game

For example, in a scenario of many players, where each player can have two strategies A and B to play, given any combination of strategies of the other players, the outcome resulting from A is better than the outcome resulting from B. Then strategy A is said to dominate strategy B. A rational player will never choose to play a dominated strategy.

2.2.1 The Prisoner's Dilemma

The Prisoner's Dilemma is a well know game which in its basic form is played by two players (prisoners). This was developed by Merrill Flood and Melvin Dresher working at RAND in 1950 Poundstone [1992]. In this game, each player has two strategies, cooperate and defect, which are indicated by C and D for player I and c and d for player II, respectively as it is shown in the Figure 2.1.

The history behind this game is that two prisoners are suspect of a serious crime. There is no evidence for this crime except if one of the prisoners testifies against the other, which is their dilemma. So, if one of them testifies, he will be rewarded with immunity from prosecution (payoff 3), whereas the other will serve a long prison sentence (payoff 0). If both testify, their punishment will be less severe (payoff 1 for each one). However, if they both cooperate with each other by not testifying at all, they will only be imprisoned briefly, for example for illegal weapons possession (payoff 2 for each one). The defection from that mutually beneficial outcome is to testify, which gives a higher payoff no matter what the other prisoner does, with a resulting lower payoff to both.

The Figure 2.1 shows the resulting payoffs in this game for any of these two

strategies selected. The strategies player by Player is in the row (C,D) , and the strategies played by player II are in column (c,d). The strategies combination are described in Figure 2.1. The combination (C; c) has payoff 2 for each player, and the combination (D; d) gives each player payoff 1. The combination (C; d) results in payoff 0 for player I and 3 for player II, and when (D; c) is played, player I gets 3 and player II gets 0.

In this game, players I and II act simultaneously, it means that each player act without knowing the other's action. In this game, defect is an strategy that dominates cooperate. For example, strategy D of player I dominates C since if player II chooses c, then player I's payoff is 3 when choosing D and 2 when choosing C; if player II chooses d, then player I receives 1 for D as opposed to 0 for C. So, D is always better strategy and dominates C. In the same way, strategy d dominates c for player II.

In this scenario, even cooperation is better strategy for these players, because of better payoff for each one, the dominating strategy for these two players is defect.

2.2.2 Nash Equilibrium

John Nash Nash [1951] published this concept in 1951, which had a previous version from 1838 by Antonie Augustin Cournot in a theory of oligopoly. In 1994 John Nash received the Nobel Memorial Prize in Economic Sciences for his contribution to the game theory, and got attention when his life was presented in a Hollywood movie called "A Beautiful Mind" were in a scene he appears playing Go game.

In a game theory, the Nash equilibrium is a solution concept of a non-cooperative game involving two or more players in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only his own strategy unilaterally. In many games there are no dominated strategies as it was described in the example above, so these considerations are not enough to provide more specific advice on how to play these games. So, A Nash equilibrium recommends a strategy to each player that the player cannot improve upon unilaterally, that is, given that the other players follow the recommendation. Since the other players are also rational, it is reasonable for each player to expect his opponents to follow the recommendation as well Turocy & Stengel [2002].

In a game can be more than one Nash equilibrium if these equilibrium are not dominated strategies. For example, in the prisoner's dilemma there are two equilibrium, if two players cooperate or both players defect, but even the first equilibrium when both selects cooperate is better for both players because of obtaining a better payoff, this is a dominated strategy because in any moment

any of other player can chose unilaterally the other strategy defect to obtain a better payoff. So, in this game there is only one Nash equilibrium which is defect by two players.

If there would not dominated strategy and more than one Nash equilibrium exist, the equilibrium selected by the players should be the one that provide a better payoff for both players.

2.2.3 Evolutionary Game Theory

Sandholm [2007] defines evolutionary game theory (EGT) as the studies of behavior of large populations of agents who repeatedly engage in strategic interactions. The changes in behavior of these populations are driven either by natural selection via differences in birth and death rates, or by the application of myopic decision rules by individual agents. EGT provides a framework where contests, strategies, and analytics into Darwinian competition can be modeled.

This theory was introduced by Maynard Smith Smith [1972], Smith [1982], Smith & Price [1973], which adapted the traditional game theory to the context of biological natural selection. He proposed his notion of an evolutionary stable strategy (ESS) as a way of explaining the existence of ritualized animal conflict and has been developed by economist and biologist and applied to different fields as transportation science, computer science, sociology and physics Sandholm [2007].

EGT differs from classical game theory by focusing more on the dynamics of strategy change as influenced not solely by the quality of the various competing strategies, but by the effect of the frequency with which those various competing strategies are found in the population. As in biology, in EGT these strategies are genetically inherited from parents to child and control the individual actions, so the children do not have the ability to change it.

Unlike the traditional game theory, in EGT is not required that players are rational, it is only required that players have an strategy, and the results of the game will test how good the strategy is. The key point in the EGT model is that the success of a strategy is not just determined by how good the strategy is in itself, it depends on how good the strategy is in the presence of other alternative strategies, and on the frequency that other strategies are employed within a competing population.

In biology the payoff is called fitness, and represent of level of reproductive success relative to some baseline level. That fitness of an individual organism can't be measured in isolation; rather it has to be evaluated in the context of the full population in which it lives Easley & Kleinberg [2010].

2.2.3.1 Nash Equilibrium and Formalization of the Solution Concept

Sandholm [2007] introduce a simple model of strategic interaction and defines a solution concept for a symmetric two-player game.

In a symmetric two-player normal form game, each of the two players chooses a (pure) strategy from the finite set S; which we write generically as $S = \{1, .., n\}$. The payoffs of the game are described by the matrix $A \epsilon \Re^{n \times n}$. Entry A_{ij} is the payoff a player obtains when he chooses strategy i and his opponent chooses strategy j; this payoff does not depend on whether that player is player 1 or player 2.

The fundamental solution concept of non-cooperative game theory is Nash equilibrium. We say that the pure strategy $i \epsilon S$ is a symmetric Nash equilibrium of A if

condition (1): $A_{ii} \geq A_{ji}$ for all $j \epsilon S$.

Thus, if his opponent chooses a symmetric Nash equilibrium strategy i, a player can do no better than to choose i himself. A stronger requirement on strategy i demands that it be superior to all other strategies regardless of the opponent's choice:

condition (2) : $A_{ik} > A_{jk}$ for all $j, k \epsilon S$:

When condition (2) holds, we say that strategy i is strictly dominant in A.

As it was described in the prisoner's dilemma example, the strategy for both players defect is the dominant strategy in this game, so, defect is the symmetric Nash equilibrium, if any player choose another option will receive less payoff.

2.3 The Go Game

In this section is going to be presented the Go game, the rules and some strategies as life and death and some terms used in this game.

2.3.1 History of Go

The game of Go, as it is known in the Western hemisphere, originates from China where it is known as Weiqi, in Japan is called Igo, and in Korea Baduk. According to legends it was invented around 2300 B.C. by the emperor Yao Yang & Anl [2005], famous in the Chinese mythology, to teach his son tactics, strategy, and concentration. The game was first mentioned in Chinese writings from Honan dating back to around 625 B.C. Bell [1980].

According to Go Knowledge, Since ancient times in China, music (the lute), Go, calligraphy and painting were considered as the four essential accomplishments of an educated person, so people were trained in them from childhood. Of course, a king had to master them also.

There are some evidences that Go when into Japan around the 7th century and it was called Igo (which later gave rise to the English name Go).

In the 8th century Go gained popularity at the Japanese imperial court, and around the 13th century it was played by the general public in Japan. Early in the 17th century, with support of the Japanese government, several Go schools were founded.

In the late 16th century the first westerners came into contact with Go. There are some evidences that the mathematician Leibniz make some publications about the Go game van der Werf [2004]

The first detailed description of Go in a European language, De Circumveniendi Ludo Chinensium (About the Chinese encircling game), was written in Latin by Thomas Hyde, professor from Oxford university and expert in oriental languages and the history of games, and included in his 1694 treatise on Oriental board games, De Ludis Orientalibus (About Oriental games) Go in Europe.

Oscar Korschelt, a German engineer, is credited with being the first person to try to popularize Go outside of Asia. He learned about the game from Honinbo Shuho (Murase Shuho) when he worked in Japan from 1878 to 1886. Korschelt published a detailed article on Go in 1880. A few years later he published a book based on this article. He brought the game to Europe, especially to Germany and Austria, and thus became the first person to systematically describe Go in a Western language. Since he learned Go in Japan, the terms of Go in Western languages come from Japanese, not Chinese. By the early 20th century, Go had spread throughout the German and Austro-Hungarian empires.

In 1905, Edward Lasker learned the game while he was in Berlin. When he moved to New York, Lasker founded the New York Go Club together with (amongst others) Arthur Smith, who had learned of the game while touring the East and had published the book The Game of Go in 1908 Hutchinson [1995]. At 1934, Lasker published a book Go and Go-moku. All of this, stimulated the popularity of the game in US.

Nowadays, Worldwide, Go is now played by 25 to 50 million players in many countries, of which several hundreds are professional.

Although most players are still located in China, Korea, and Japan, the last decades have brought a steady increase in the rest of the world, which is illustrated by the fact that Europe nowadays has over one thousand active players of amateur dan level, and even a few professionals.

In Madrid there some Go clubs as Club Go Nam-Ban, which promotes this game in Spain with events and tournaments.

2.3.2 Rules

The game of Go is played by two players, Black and White, who consecutively place a stone of their color on an empty intersection of a square grid. Usually the grid contains 19×19 intersections, but it is used boards of 9×9 and 13×13 as well.

There are some variations of the Go rules. For example Chinese and Japanese rules have some variations mainly in the scoring method.

Different associations around the world has adopted different scoring method, for example, the Namban association has adopted the Japanese scoring method.

There some other rules that are accepted as American Go Association (AGA) rules, SST(Ing), New Zeland and Tromp-Taylor rules. But even the variations to the Go rules, the different set of rules lead to the same results.

For a good description of the game, specially for the amateurs, in this thesis is used the rules published by the AGA Committee [1991] which is accepted by some associations as France, and British Go Association.

- Rule 1. The Board and Stones: Go is a game of strategy between two sides usually played on a 19×19 grid (the board). The game may also be played on smaller boards, 13×13 and 9×9 being the two most common variants. The board is initially vacant, unless a handicap is given (see Rule 4). The two sides, known as Black and White, are each provided with an adequate supply of playing tokens, known as stones, of the appropriate color.

  The traditional 19×19 Go board can be observed in the Figure 2.1.

  To test the new technique discussed in this thesis, it was selected the board of 9×9. Even this board is considered for training propose for the novices, this have the enough complexity to be used for testing propose. According to Allis [1994], the 9×9 board has the same complexity as Chess. Usually the boards used for testing as less size than this.

- Rule 2. Play: The players alternate in moving, with Black playing first. In handicap games, White moves first after Black has placed his or her handicap stones. A move consists in playing a stone of one's color on an empty intersection (including edges and corners), or in passing. Certain moves are illegal (Rules 5 and 6), but a pass is always legal (Rule 7). Points are awarded for controlling space in a manner described below (Rule 12). The object of the game is to end with the greater total number of points.

- Rule 3. Compensation (or Komi): In an even (non-handicap) game, Black gives White a compensation of $7\frac{1}{2}$ points for the advantage of the first move. This compensation is added to White's score at the end of the game. In

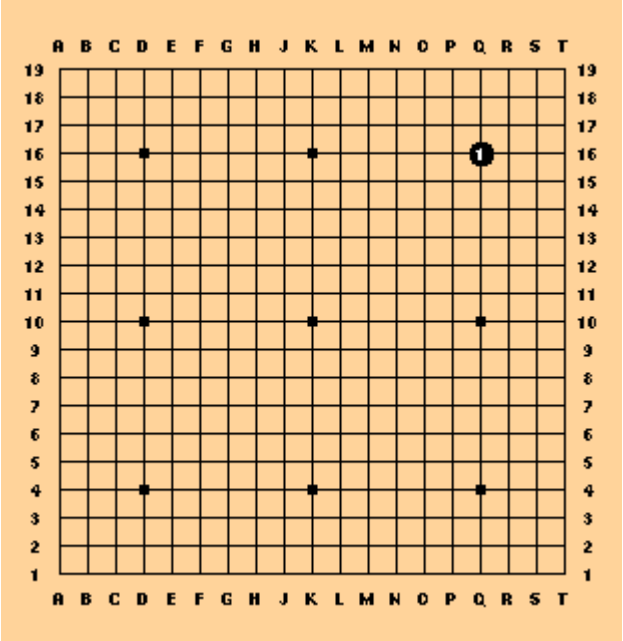


Figure 2.2: Traditional 19x19 Go board

handicap games, Black gives White $\frac{1}{2}$ point compensation. This avoids draws.

In case of the Japanese rule, the Komi is $6\frac{1}{2}$. In Chinese rule, the Komi is $7\frac{1}{2}$.

- Rule 4. Handicaps: The game may be played with a handicap to compensate for differences in player strengths. The weaker player takes Black, and either moves first, giving only $\frac{1}{2}$ point compensation to White, as in Rule 3 (this is known as a "one stone handicap"), or places from 2 to 9 stones on the board before the first White move.

  In the of AGA rules and Japanese, the handicap positions of the stones are fixed. In case of the Chinese rules the handicap positions of the stones are free.

- Rule 5. Capture: Stones of the same color are said to be connected if they are adjacent along horizontal or vertical–not diagonal–lines on the board. A string of connected stones consists of those stones which can be reached from a given stone by moving only to adjacent stones of the same color. A string of connected stones is surrounded by stones of the opposite color if it has no empty points horizontally or vertically–not diagonally–adjacent

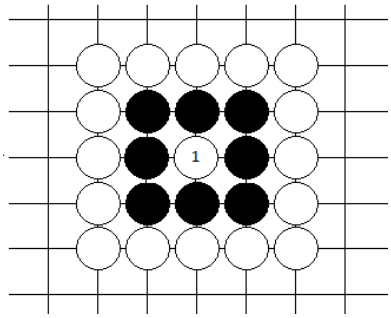


Figure 2.3: White 1 captures the black stones

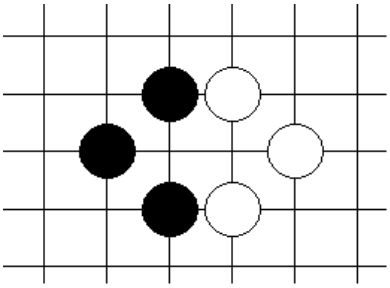


Figure 2.4: An example of Ko

to any of its member stones. (Such adjacent empty points are known as liberties of the string.)

After a player moves, any stone or string of stones belonging to the opponent which is completely surrounded by the player's own stones is captured, and removed from the board. Such stones become prisoners of the capturing player. It is illegal for a player to move so as to create a string of his or her own stones which is completely surrounded (without liberties) after any surrounded opposing stones are captured.

This can be observed in the Figure 2.2.

- Rule 6. Repeated Board Position (Ko): It is illegal to play in such a way as to recreate a previous board position from the game, with the same player to play.

  This can be observed in the Figure 2.3.

  In the case of Japanese rule, is Ko exist or position repeat, the game may be replayed. In case of Chinese rules, repetitions are forbidden.

- Rule 7. Passing: On his or her turn, a player may pass by handing the opponent a stone, referred to as a pass stone, rather than playing a stone

on the board.

This rule is different to Chinese and Japanese rule. In this rule, not stones is give to the opponent. For the easy implementation of the Computer Go player, this rule is not considered.

- Rule 8. Illegal Moves: An illegal move is one violating the rules. If a player makes an illegal move–such as moving twice in a row (i.e., before the opponent has made a response), attempting to play on an occupied intersection, self-capture, or retaking a ko so as to repeat the full board position, the player must take back his or her move (both moves, if he or she moved twice in succession), it shall be treated as a pass, and a pass stone exchanged.

  An illegal move must be noted as such by the opponent before he or she makes his or her move. When a player moves, he or she is tacitly accepting the opponent's previous move as valid. In particular, if it is discovered that an earlier move by one of the players was illegal, the game must nevertheless be continued as it stands unless both players agree to restore the earlier board position and proceed from that point.

  For simplicity in the thesis in the computer Go programmed, the illegal moves are not allowed, and the pass stone exchange is not considered.

- Rule 9. Ending the Game: Two consecutive passes signal the end of the game. After two passes, the players must attempt to agree on the status of all groups of stones remaining on the board. Any stones which the players agree could not escape capture if the game continued, but which have not yet been captured and removed, are termed dead stones. If the players agree on the status of all such groups, they are removed from the board as prisoners of the player who could capture, and the game is scored as in Rule 12. If there is a disagreement over the status of some group or groups, play is resumed as specified in Rule 10.

  For the simplicity of the computer Go programed in the thesis, when the two players passed consecutive the game end.

- Rule 10. Disputes: If the players disagree about the status of a group of stones left on the board after both have passed, play is resumed, with the opponent of the last player to pass having the move. The game is over when the players agree on the status of all groups on the board, or, failing such agreement, if both players pass twice in succession. In this case any stones remaining on the board are deemed alive. Any stone or group of stones surrounded and captured during this process is added to the capturing player's prisoners as usual.

In the thesis, this rule is not implemented as it was commented in the rule 9.

- Rule 11. The Last Move: White must make the last move–if necessary, an additional pass, with a stone passed to the opponent as usual. The total number of stones played or passed by the two players during the entire game must be equal.

  As it was commented, the passed stones is not considered in this thesis.

- Rule 12. Counting: There are two methods for counting the score at the end of the game. One is based on territory, which is known as Japanese scoring, the other on area, which is known Chinese scoring. The players should agree in advance of play which method they will use. If there is no agreement, territory counting shall be used.

  - Territory: Those empty points on the board which are entirely surrounded by live stones of a single color are considered the territory of the player of that color.
  - Area: All live stones of a player's color left on the board together with any points of territory surrounded by a player constitute that player's area.

  For this Thesis the Chinese counting scoring method has been selected.

2.3.3 The basic Ko rule

As it was discussed in the Rule 6, the Basic ko only prevents direct repetition in a cycle of length two. Usually longer cycles are always allowed. If we can prove a win (or loss), when analyzing a position under the basic-ko rule, it means that all repetitions can be avoided by playing well.

In this thesis for simplicity only the basic ko rule has been considered. In the implementation chapter will be explain how to avoid more long ko cycles.

2.3.4 Life and death

As it was discussed in the Rule 9, 10, and 12, the life and death of groups of stones is normally decided by agreement at the end of the game. In most cases this is easy because a player only has to convince the opponent that the stones can make two eyes, or that there is no way to prevent stones from being captured. If players do not agree they have to play out the position.

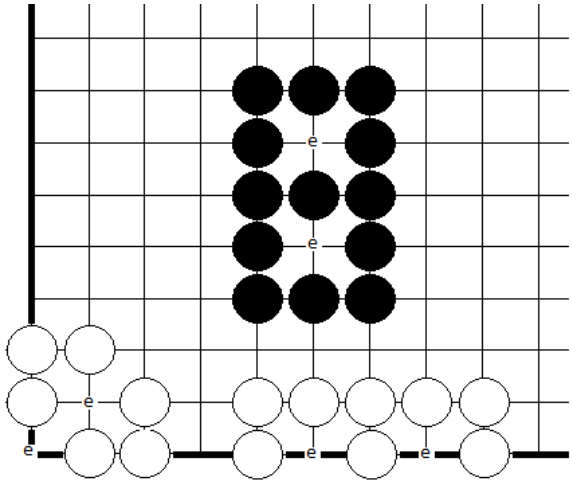


Figure 2.5: An example of Alive stones because the two eyes

For computer go programs, it is difficult to have these agreements, so, the simplest way is continue the game till there is not intersection to fill or every player pass two times.

2.3.5 Glossary of Go terms

Below is an overview about some Go terms used:

- Adjacent: On the Go board, two intersections are adjacent if they have a line but no intersection between them.
- Alive: Stones that cannot be captured are alive. Alive stones normally have two eyes or are in seki. In the Figure 2.3 can be observed some examples.
- Atari: Stones are said to be in atari if they can be captured on the opponents next move, i.e., their block has only one liberty.
- Area: A set of one or more intersections. For scoring, area is considered the combination of stones and territory.
- Baduk: Korean name for the game of Go.
- Block: A set of one or more connected stones of one colour.
- Chain: A set of blocks which can be connected.

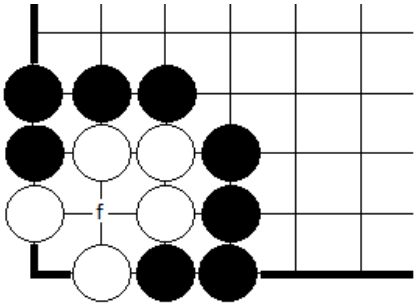


Figure 2.6: An example of dead stones and false eyes marked with f

- Connected: Two adjacent intersections are connected if they have the same colour. Two non-adjacent intersections are connected if there is a path of adjacent intersections of their colour between them.
- Dame: Neutral point(s). Empty intersections that are neither controlled by Black or by White. Usually they are filled at the end of the game.
- Dan: Master level. For amateurs the scale runs from 1 to 6 dan, where each grade indicates an increase in strength of approximately one handicap stone. If we extrapolate the amateur scale further we get to professional level. Professional players use a more fine-grained scale where 1 dan professional is comparable to 7 dan amateur, and 9 dan professional is comparable to 9 dan amateur.
- Dead: Stones that cannot escape from being captured are dead. At the end of the game dead stones are removed from the board.
- Eye: An area surrounded by stones of one colour which provides one sure liberty. Groups that have two eyes are alive.
- False eye: An intersection surrounded by stones of one colour which does not provide a sure liberty. False eyes connect two or more blocks which cannot connect through an alternative path.
- Gote: A move that loses initiative. Opposite of sente.
- Handicap: Handicap stones may be placed on the empty board by the first player at the start of the game to compensate the difference in strength with the second player. The difference in amateur grades (in kyu/dan) between two players indicates the number of handicap stones for providing approximately equal winning chances.

- Jigo: The result of a game where Black and White have an equal score, i.e., a drawn game.
- Ko: A situation of repetitive captures.
- Komi: A pre-determined number of points added to the score of White at the end of the game. The komi is used to compensate Blacks advantage of playing the first move.
- Kyu: Student level. For amateurs the scale runs from roughly 30 kyu, for beginners that just learned the rules, down to 1 kyu which is one stone below master level (1 dan). Each decrease in a kyu-grade indicates an increase in strength of approximately one handicap stone.
- Liberty: An empty intersection adjacent to a stone. The number of liberties of a block is a lower bound on the number of moves that has to be made to capture that block.
- Ladder: A simple capturing sequence which can take many moves.
- Life: The state of being safe from capture. See also alive. Miai Having two different (independent) options to achieve the same goal.
- Prisoners: Stones that are captured or dead at the end of the game.
- Seki: Two or more alive groups that share one or more liberties and do not have two eyes. (Neither side wants to fill the shared liberties.)
- Sente: A move that has to be answered by the opponent, i.e., keeps the initiative. Opposite of gote.
- Suicide: A move that does not capture an opponent block and leaves its own block without a liberty. (Illegal under most rule sets.)
- Territory: The intersections surrounded and controlled by one player at the end of the game.
- Weiqi / Weichi: Chinese name for the game of Go.

There are series of books recommended by the Go community that can be useful to learn more about this game Kim & Soo-hyun [1994], Kim & Soo-hyun [1995] and Kim & Soo-hyun [1996] and others. Some more information about the Go game and information about Go community can be found in this site Sensei's Library.

2.4 Game properties used to Calculate the Complexity of Games

The following subsections summarize the properties to the board games which be useful to understand the complexity and the approach to solve these games.

2.4.1 Perfect information

The perfect-information property divides the set of games into two disjoint subsets: the set of perfect-information games and the set of imperfect-information games Allis [1994].

In a perfect-information game, all players, at any time during the game, have access to all information defining the game state and its possible continuations Allis [1994].

2.4.2 Convergence

The convergence property labels games as either converging, diverging or unchangeable. A game is called converging game if the number of pieces decrease during the game. In case of Go, this game is called diverging, because the number the pieces in the board increase during the game Allis [1994].

2.4.3 Sudden death

A sudden-death game may end abruptly by the creation of one of a pre-specified set of patterns. A fixed-termination game lacks sudden-death patterns Allis [1994].

Go is considered a fixed-termination game, when the two players (white and black) decide to finalize the game, usually when once of these two players pass more than two times consecutively.

2.4.4 Complexity

To define the complexity of the games analyzed by Allis [1994] it was used the Search Tree which by history, in Chess and Go, has been called Game-Tree van der Werf [2004]. Basically it is proposed two measures: state-space complexity and game-tree complexity

2.4.4.1 State-space complexity

The state-space complexity of a game is defined as the number of legal game positions reachable from the initial position of the game Allis [1994].

For example, for tic tac toe is obtained by calculating that the 9 spaces can be occupied by X or O or empty, so, the upper bound for the State-Space complexity for the tic tac toe is 3^9 (including illegal positions). but excluding all illegal positions the real state-space complexity is 5478 Allis [1994].

To calculate this for different games, Alli uses a super set of all positions (including illegals), an evaluation function to identify if a position is illegal and Montecarlo simulation to identify the fraction of legal positions,so that multiplying the percentage of legal positions by superset of positions the state-space complexity is calculated.

2.4.4.2 Game-Tree complexity

To define the game-tree complexity, Allis [1994] uses two more definitions:

Solution depth: The solution depth of a node N is the minimal depth (in ply) of a full-width search sufficient to determine the game-theoretic value of N.

Solution Search: The solution search tree of a node N is the full-width search tree with a depth equal to the solution depth of N.

As an example for chess, we consider a chess position N with white to move. White has 30 legal to move, and we can assume that after each legal white move, black has 20 legal moves of which at least one mates white. Then, the solution search tree of N consists of N, the 30 children of N, and the 600 grandchildren of N Allis [1994].

So, the game-tree complexity of a game is the number of leaf nodes in the solution search tree of the initial position(s) of the game. From the previous example, if N were the initial position, the game-tree complexity would be 600 Allis [1994].

The game-tree complexity is an estimate of the size of a minimax search tree which must be built to solve the game. Thus, using optimally-ordered alpha-beta search, we may expect to search a number of positions in the order of the square root of the game-tree complexity Knuth & Moore [1975], Allis [1994]. The minimax and alpha-beta search will be explained in the next section. So, based in the previous properties described above, we can classify the Go game as a diverging, perfect-information game with fixed termination.

The state-space complexity of Go is $3^{361} = 10^{172}$, it is far larger that any other perfect-information game. Its game-tree complexity, with an average branching factor of 250, and average game length of 150 ply, it is approximately 10 exp(360) Allis [1994]

Table 2.1: Summary of the complexity for some board games

Game	log10(E)	log10(A)	Computer-Human results
Checkers	17	32	Chinook > H
Othello	30	58	Logistello > H
Chess	50	123	Deep Blue ≥ H
Go	160	400	Handtalk ≪ H

According to Bruno Bouzy Bouzy & Cazenave [2001] the table 2.1 summarize the classification of the complexity of the some games, where space states complexity (E) as the number of positions you can reach from the starting position, and the game tree complexity (A) as the number of nodes in the smallest tree necessary to solve the game.

According to Bouzy & Cazenave [2001] the Go 9×9 board has the same complexity as the Chess.

2.5 Overview of Techniques Applied to Computer Go

The following sections will review the most known Searching techniques applied to Go.

2.5.1 Overview of Searching Techniques

In the research of AI in games, the use of Searching techniques have been broadly used as it can be described in Pazos [1987].

2.5.1.1 The Search Tree

The Search Tree can be represented by a directed graph of nodes in which each node represents a possible game state storing some values. In games as Go and Chess, this graph is usually called as game tree.

The first node in the tree is called root node (which represent the position under investigation). The nodes are connected by branches which represent the possible moves that are made between game states.

The node one ply closer to the root is called the parent. Nodes that belong to the same parent are called child nodes. Nodes, at the same depth, sharing the same parent are called siblings. When nodes are not expanded, they are called leaf nodes.

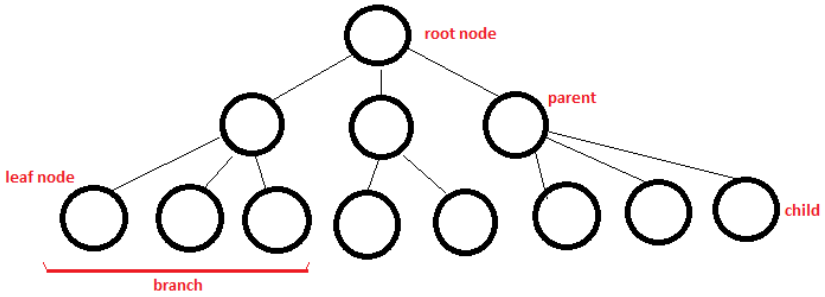


Figure 2.7: Simple Structure of Tree used in Tree Search methods

The legal moves are represented from the node root by branches which expand the tree to nodes at a distance of one ply from the root. In an analogous way, nodes at one ply from the root can be expanded to nodes at two plies from the root, and so on.

When a node is expanded n times, and positions up to n moves ahead have been examined, the node is said to be investigated up to depth n. An example of a tree used in this technique can be observed in the Figure 2.7.

According van der Werf [2004] there are at least three reasons why leaf nodes are not expanded further :

- The corresponding position may be final (so the result of the game is known) and the node is then often denoted as a terminal node.
- There may not be enough resources to expand the leaf node any further (these are not final positions).
- The expansion may be considered irrelevant or unnecessary.

The process of expanding nodes of a game tree to evaluate a position during the game and find the right moves is called searching.

In theory this Search technique can be applied in Go, but in practices, because of the limited computer resources, it is almost impossible to fully expand the game trees and usually leaf nodes do not correspond to final positions. As it was discussed in the previous section, a simple estimate for the size of the game tree in Go, which assumes an average branching factor of 250 and an average game length of only 150 ply (which is quite optimistic because the longest professional games are over 400 moves), leads to a game tree of about 250150 10360 nodes Allis [1994] which is impossible to expand fully.

According to van der Werf [2004] some evaluation functions are used to predict moves of the trees with non-final positions. In theory, a perfect evaluation function with a one-ply search (an expansion of only the root node) would be

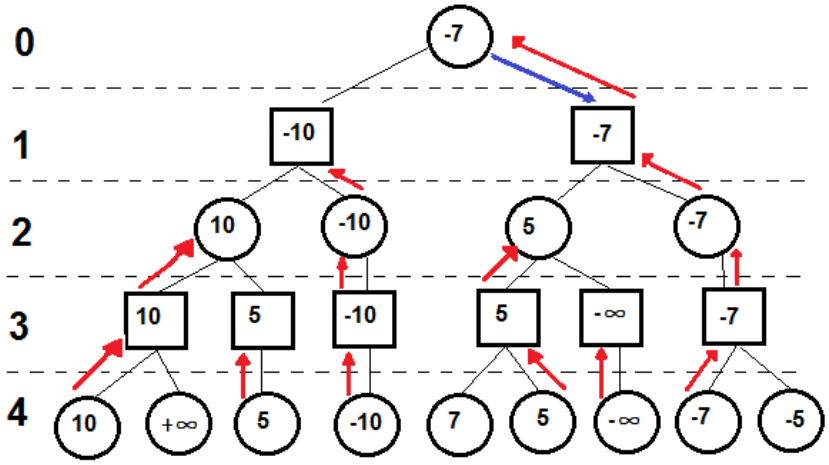


Figure 2.8: MiniMax Search Alghoritm

sufficient for optimal play. In practice, however, perfect evaluations are hard to construct and for most interesting games they cannot be computed within a reasonable time.

Since full expansion and perfect evaluation are both unrealistic, most search based programs use a balanced approach where some positions are evaluated directly while others are expanded further van der Werf [2004]. Balancing the complexity of the evaluation function with the size of the expanded search tree is known as the trade-off between knowledge and search Junghanns & Schaeffer [1997], Heinz [2003].

2.5.1.2 Minimax Search

In minimax search tree, there are two types of nodes. The first type is a max node, where the player to move (Max) tries to maximize the score. The root node (by definition ply 0) is a max node by convention, and consequently all nodes at an even ply are max nodes.

The second type is a min node, where the opponent (Min) tries to minimize the score. Nodes at an odd ply are min nodes.

Starting from evaluations at the leaf nodes, and by choosing the highest value of the child nodes at max nodes and the lowest value of the child nodes at min nodes, the evaluations are propagated back up the Search Tree, which eventually results in a value and a best move in the root node.

The strategy found by minimax is optimal in the sense that the minimax value at the root is a lower bound on the value that can be obtained at the frontier spanned by the leaf nodes of the searched tree van der Werf [2004].

However, since the evaluations at leaf nodes may contain uncertainty, because they are not all final positions, this does not guarantee that the strategy is also optimal for a larger tree.

In theory it is even possible a deeper search, resulting in a larger tree, decreasing performance; but should be considered the game-tree pathology described by Nau [1980], where there is an infinite class of game tree that even increasing the Search deep does not improve the decision quality, instead make the decision more and more random.

In practice, according to van der Werf [2004], the pathology does not appear to be a problem and game-playing engines generally play stronger when searching more deeply.

2.5.1.3 Alpha-Beta Prunning

The origin of the Alpha-Beta Prunning method is not clear, it looks like that Jhon Mc Carthy thought to use this kind of method during a conference in Darmouth in 1956, where apart this field was called Artificial Intelligence and it was systematized Corduck [1979], Pazos [1987]. It looks like that in that conference Alex Bernstein described a program which played Chess and Mc Carthy criticized him because he was not using that method, from there this method has been used in different games from the 50's of the previous century. But others claimed that this method were in their programs to play checkers as Arthur L. Samuel Pazos [1987].

After that events some publications appeared as Hart and Edwards in 1961, Brudno in 1965 and Slagel and Dixon in 1968, and finally in 1975 was published an article by D. Knuth and Moore a detail analysis of this technique and some improvements Pazos [1987].

So, although minimax can be used directly it is possible to determine the minimax value of a game tree much more efficiently using alpha-beta search Knuth & Moore [1975].

This is achieved by using two bounds, alpha and beta. The lower bound is called alpha which represents the worst possible value for the player Max. Any sub-tree of value below alpha is not worth investigating (this is called an alpha cut-off). The upper bound is called beta which represents the worst possible value for the player Min. If in a node a move is found that results in a value greater than beta, the node does not have to be investigated further because the player Min will not play this line (this is called a beta cut-off) van der Werf [2004].

When a search process decides not to investigate some parts of the tree, which would have been investigated by a full minimax search algorithm, this is called pruning. This can be observed in the Figure 2.9.

Some pruning, such as alpha-beta pruning, can be done safely without chang-

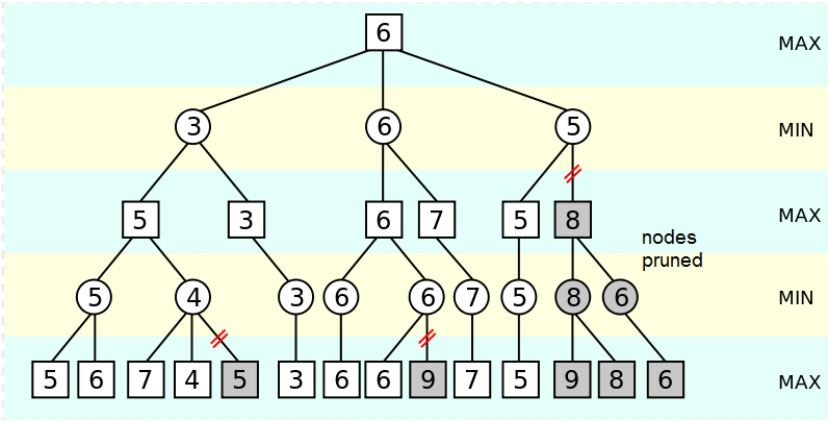


Figure 2.9: Alpha-beta Prunning

ing the minimax result (the node root value). According to Knuth & Moore [1975], the Alpha-Beta Prunning is generally used to speed up such search processes without loss of information.

Knuth & Moore [1975] refined this technique in the following procedure called F2 (where F was minimax procedure) which satisfied the following conditions:

$F2(p, alpha, beta) \leq alpha, if F(p) \leq alpha$;
$F2(p, alpha, beta) = F(p), if alpha < F(p) < beta$;
$F2(p, alpha, beta) \geq beta, if F(p) \geq beta$;

This will imply that:
$F2(p, -infinity, +infinity) = F(p)$

The Algorithm is proposed for F2 by Knuth & Moore [1975]:

```
Function F2(positionp, integeralpha, integerbeta) ;
Integer m, i, t, d ;
Determine the successor of positions p1,..pd;
if d = 0 then
    F2 = f(p) ;
end
else
    m = alpha ;
    for i ← 1 to d do
        t = −F2(pi, −beta, −m) ;
        if t > m then
            m = t ;
        end
        if m ≥ beta then
            goto done ;
        end
    end
    done: F2 = m;
end
```

When a pruning method is not guaranteed to preserve the minimax result it is called forward pruning. Forward pruning is unsafe in the sense that there is a, generally small, chance that the minimax value is not preserved van der Werf [2004].

2.5.1.4 Move Ordering

The efficiency of alpha-beta search algorithm depends on the order in which nodes are investigated.

In the worst case it is theoretically possible that the number of nodes visited by alpha-beta is identical to the full minimax search tree.

For example, with an average or constant branching factor of b, and a search depth of d plies, the maximum number of leaf node positions evaluated (when the move ordering is pessimal or does not exist) is O(b*b*...*b) = O(b exp(d)), the same as a simple minimax search.

If the move ordering for the search is optimal (meaning the best moves are always searched first), the number of leaf node positions evaluated is about O(b*1*b*1*...*b) for odd depth and O(b*1*b*1*...*1) for even depth, or b exp(d/2).

So, in the best case the number of the nodes investigated are square root as in case of the minimax. There are various Move ordering techniques that have been

investigates, some of them has some dependency in search methods and game specific knowledge. Some techniques are Transportation Table, Kill heuristics, or history tables.

According to van der Werf [2004], in principle the Kill heuristics and history table are not game specific knowledge but in Go has be used some game specific properties.

2.5.1.5 Transportation Table

In computer chess and other computer games, developers has been used transposition tables for speed up the search of the game tree. In this site Chess Programming Part II written by Francois Dominic, explain how Transportation table has been in used in chess. According to him, there are often many ways to reach the same position. It does not matter whether you play 1. P-K4 ... 2. P-Q4 or 1. P-Q4... 2. P-K4; the game ends up in the same state. Achieving identical positions in different ways is called transposing. And if the program has just spent considerable effort searching and evaluating the position resulting from 1. P-K4 ... 2. P-Q4, it would be nice if it were able to remember the results and avoid repeating this tedious work for 1. P-Q4... 2. P-K4. This is why all chess programs, since at least Richard Greenblatt's Mac Hack VI in the late 1960's, have incorporated a transposition table Chess Programming Part II.

So, accordingo to Dominic, a transposition table is a repository of past search results. When a position has been searched, the results (i.e., evaluation, depth of the search performed from this position, best move, etc.) are stored in the table. Then, when new positions have to be searched, it is query the table first, if suitable results already exist for a specific position, it is used and bypass the search entirely. Some advantage to this method described in this site are the followings:

- Speed. In situations where there are lots of possible transpositions (for example, in the endgame, when there are few pieces on the board), the table quickly fills up with useful results and 90% or more of all positions generated will be found in it.

- Free depth. For example, if it is needed to search a given position to a certain depth; say, four-ply (two moves for each player) ahead. If the transposition table already contains a six-ply result for this position, not only it is avoided the search, it is obtained more accurate results than it is obtained when it is forced for search it.

- Versatility. For example, every chess program has an "opening book" of some sort of well-known positions and best moves selected from the chess

literature. So, the transportation table can be initialized with this knowledge at the beginning of the game. In Go could be applied some transportation table with some opening strategies. i.e. start playing in the center of the board.

2.5.2 Combinatorial Game Theory

To have an overview of the mathematical part of the Combinatorial Game Theory proposed by Conway you can refer Conway [1976] or Schleicher & Stoll [2005] which has a more accesible introduction to this theory. Berlekamp Berlekamp [1991] and Muller Muller [1999], Muller [1995] proposed a practical applications of the Conway's game theory to Computer Go.

This theory are have a highly successful results if it is applied to specific sub-problems as the endgame positions, and less to the whole game. In the Go community had a tremendous impact because everybody though that professional players were playing the endgame optimally Bouzy & Cazenave [2001].

According to this theory, during the endgame phase, it is partly possible to model a position as the sum of games.

Bouzy & Cazenave [2001] listed the four key features needed in the positions to apply this theory and obtain outstanding results. The first feature corresponds to the identification of groups and territories. As a game nears its end, the identification of groups and territories becomes easier, and thus, the transformation from the position into a list of sub-games becomes possible. In addition, when moves are played, the identification of groups and territories becomes more stable. Stability is the second feature. It is fundamental because, if not verified, the local tree searches in each sub-game would become pointless. The third feature is the independence of the sub-games. As a game nears its end, the moves played in one sub-game have less influence on other sub-games. Therefore, near the end of the game, the sub-games become independent. The four feature is the completion of tree searches. As a game nears its end, local searches become shorter. Therefore local searches can be fully completed and the sub-games described.

Muller [1999] describes his Decomposition Search algorithm which describe a framework to solving games using decomposition, local search, called Local Combinatorial Game Search (LCGS), and applying the combinatorial game theory to the results local game graphs. Let G be a game that can be decomposes into a sum of sub-games $G_1 + ... + G_n$. Let the combinatorial game evaluation of G be $C(G)$. Decomposition search framework is defined as the following four steps algorithm for determining optimal play of G:

- Game decomposition and sub-game identification: given G, find an equivalent sum of sub-games $G_1 + ... + G_n$.

- Local combinatorial game search (LCGS): for each G_i, perform a search to find its game graph $GG(G_i)$.
- Evaluation: for each game graph $GG(G_i)$ evaluate all terminal positions, then find the combinatorial game evaluation of all interior nodes, leading to the computation of $C(G_i)$.
- Sum game play: through combinatorial game analysis of the set of combinatorial games $C(G_i)$ select an optimal move in $G_1 + ... + G_n$.

This method has been incorporated in his program Explorer with better results compared against alpha-beta search.

In other important Go problems have been applied the combinatorial game theory as in Eyes, which is very useful in the life and death problems, and Ko. Even in Go the loopy games are baned by rules, but when it is model the whole game as sum of sub-games, its sub-games may be loopy.

But, the problem is to apply this theory to the whole game, as Bouzy & Cazenave [2001] mentions, even if you have a sub-games model of the global game, the sub-games are greatly dependent on one another. Each model of a global position into sub-games already contains an approximation.

But even this issue, this theory has been incorporated in outstanding computer Go players with very successful results as have been created using this theory as Goliath, three times world champion in 1989, 1990 and 1991, modeled local searches with switches; Explorer which is the program that uses the theory in the most efficient way; Indigo, which use Conways classification (positive, negative, fuzzy or zero). Unfortunately, local searches in some sub-games cannot be completed, and such sub-games are placed in the unknown category Bouzy & Cazenave [2001].

2.5.3 Learning Techniques

2.5.3.1 Supervised Learning vs. Reinforcement Learning

Supervised Learning is useful when the agent has access to correct the test data or examples that need to be learned, and the agent can learn from the errors between its decisions and the correct decisions.

In Reinforcement Learning unfortunately the correct course of actions is unknown, and the agent has to learn the good behavior through trial and error by directly interacting with the domain.

In the computer Go, have been explored these two approaches, but because the large number of strategies exist in this game, it is almost impossible to have the test data to train computer go programs to learn to play go with a good level.

So, recently have been explored more Reinforcement Learning techniques. For example, currently the program that have more records beating to professional players are the MoGo, which is based in the Monte Carlo Tree Search algorithm, a promising automatic knowledge learning technique.

The other approach is to use another computer Go programs as sparring to learn competing against them, but, this can lead us to the deterministic problem, where our computer go program can only beat to the sparring but not other players, even weak players.

In the thesis are presented maybe the most promising techniques for automatic knowledge learning, Temporal Difference Learning, Monte Carlo and its variance, and evolutionary algorithms using Neural Networks and Genetic Algorithms.

2.5.4 Temporal Difference Learning (TDL)

Temporal Difference Learning was proposed by Sutton [1988] and has been applied successfully to other games as Backgammon by Tesauro Tesauro [1993]. According some authors Enzenbergerl [1996], Schraudolph *et al.* [2000] TDL should be applied successfully to the Go game, because as in Backgammon, the positional judgement, and the knowledge about the stones are more important, than the ability to see many moves ahead.

According to Bouzy & Cazenave [2001] there are two important abilities can be identified in games as Go. The first one is to foresee the likely continuation of a game, either by tree search, or by reasoning; and the second one is the ability to assess a position accurately, using patterns and some features of the position, but without calculating explicit move.

Two authors have been applied TDL to Go, Schraudolph *et al.* [2000], who designed a network to played successfully against Wally (low level player public available), and a low level version of many faces of go and Enzenbergerl [1996], who created a computer Go program called NeuroGo, which was able to beat to many faces of go with a medium level of play.

The network of Schraudolph *et al.* [2000] applied TDL in the same way than Tesauro [1993], this network was structured in 82-40-1 initially playing stochastically with not good results, but after some improvements as changing the only one output to the number of intersections, and applied some reflection and rotation properties of the Go game. For example: Color symmetry is taken into account by giving opposite values to the inputs for Black and White (+1 for Black, -1 for White), and by putting the bias neuron to 1 when White has to play first. Weight adjustments take into account rotations and symmetries by sharing equivalent weights, and by adding the errors resulting from different, but equivalent, intersections.

This network was tested against Wally and Many faces of Go playing at level

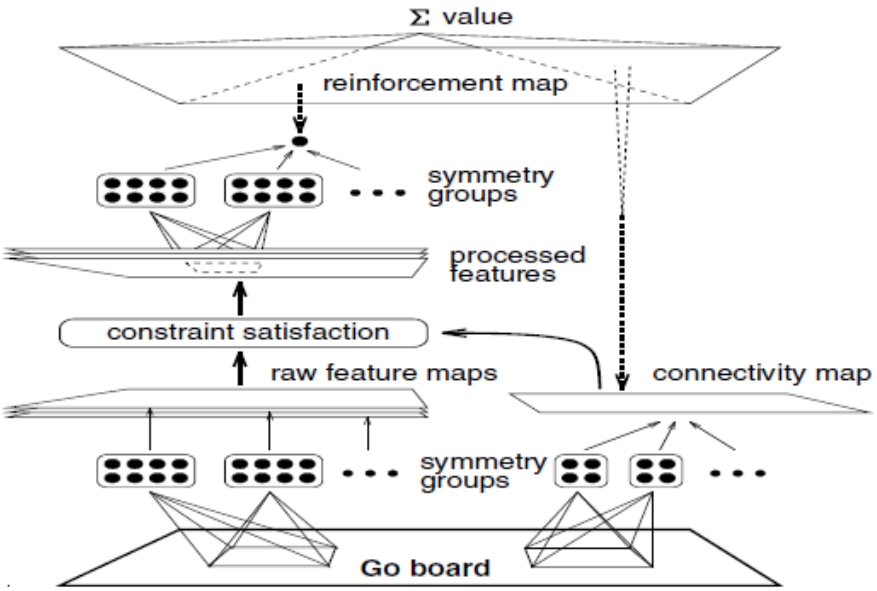


Figure 2.10: The network proposed by Schraudolph that take advantage of board symmetries, translation invariance and localized reinforcement

2-3 (of 20), but took too much games to beat them. The Figure 2.10 shows the architecture used for these tests.

The NeuroGo program of Enzenbergerl [1996] had demonstrated better results that the previous program using TDL but with some differences. he input of the network is constituted of units. One unit can be either an empty intersection, or a string of stones. The architecture of the network is therefore dependent on the position being considered.

In the Figure 2.11 is show the architecture proposed by Enzenbergerl. This has three components: Feature Expert, which calculates the network's input; Relation Expert, which determine the connectivity and External Expert, which add more knowledge to the system and can override the output of the network.

The Relation Expert uses a priori knowledge to detect relationships between pair of units.

Every unit has its own properties. Some properties considered by the Feature Expert are the followings:

- For Black and White strings:
  - Number of liberties (1,2,3,4, ≥ 5).

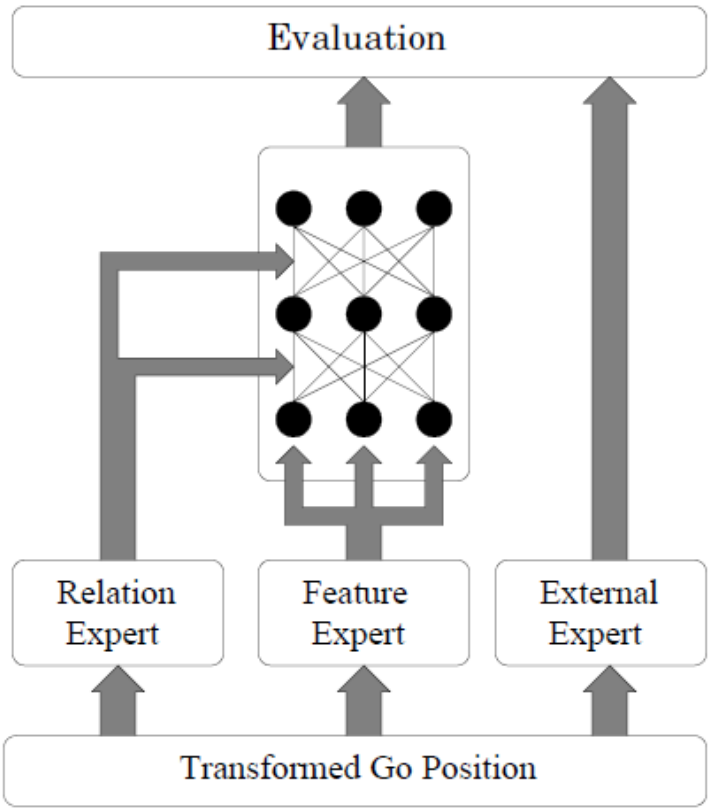


Figure 2.11: The system architecture used in NeuroGo. The position is transformed into a set of strings and empty intersections.

  - Number of stones (1,2,3,4, ≥ 5).
  - Can be capture by a ladder if string colour plays first.
  - Can be capture in a ladder if string colour plays second.

- Empty intersections:
  - Liberties of Black if he plays here (1,2,3,4,≥ 5).
  - Liberties of White if he plays here (1,2,3,4, ≥ 5).
  - Black can be captured in a ladder if he plays here.
  - White can be captured in a ladder if he plays here.
  - Eye for Black, n moves missing (n=0,1, 2, ≥ 3).
  - Eye for White, n moves missing (n=0,1, 2, ≥ 3).

In addition to calculating the properties of the units, NeuroGo detects groups of stones (sets of connected strings), and the distances between strings (connectable in one move or in two moves). It uses this information to link units with weights corresponding to the relations between the units. The set of units of a position is converted into a graph. This graph enables the program to build relations within the neural network. There cannot be more than one relation two units.

The External Expert recognize and evaluates parts of the board completely by its own. It can be used if there is a simple and correct algorithm for evaluating a certain class of local positions.

The program was trained to playing against itself to avoid the deterministic issue to learn to beat to only one opponent. After the training, it was tested against the Many faces of Go playing at level 8 (of 20) with results in a 9×9 board. For the test was used different lengths of hidden layer, from 3 to 24 neurons per unit.

So, these results demonstrated that TDL can be used as well to learn how play, but the bad part is that is needed more knowledge to have a better level of play.

Because these results and other new approaches used for learning knowledge in Computer Go, Runarsson & Lucas [2005] compared TDL using the self-play gradient descent method and the use of co-evolutionary learning (CEL) using evolution strategy with interesting results that will be discussed in the next chapters.

There are other comparisons of TDL using self-play and CEL methods in other games as Lucas & Runarsson [2006] in Othelo , Kotnik & Kalita [2003] in Gin rummy, and Darwen [2001] in Backgammon.

2.5.5 Montecarlo Go

Montecarlo method was introduced by Ulam in 1946 Ulam [1991],Eckhardt [1987], when he was playing Solitarie card game and trying to answer the question what are the chance that a Canfield solitarie laid out with 52 cards will come out successfully, so, he decided to use an heuristic approach playing hundred of times, observe and count the number of successfully plays. This method was immediately applied to resolve some problems in physics in the Alamos and named Montecarlo by Von Neumann and Ulam referring to Monte Carlo casino in Monaco Eckhardt [1987].

The use of Monte Carlo simulation to the Go game was proposed by Brugmann [1993]. This method and its variations in the last years have been applied to different games, not only board games Chaslot *et al.* [2008].

One of the applications of Monte Carlo methods are applied to statistical physics. A statistical system is composed of a large number of particles. The problem is to know how the speed, and the position, of particles evolve over time. A feature of the evolution of the system is that a quantity, such as the energy, or the activity is minimized Bouzy & Cazenave [2001].

In this physics there is a process called annealing, in which at high temperatures, a metal is liquid, and its atoms move randomly, but when the metal is cooled, the atoms put themselves into a configuration that minimizes energy a crystalline structure. longer the cooling, the closer to the minimum of energy the cooled structure is.

The Monte Carlo method try to simulate the annealing process.

The evolution of the system is approximately assessed by choosing a move with a probability that depends on the growth in activity resulting from the move. For example, the probability p(E) that a particle has the energy E at a temperature T is $p(E) = e^{(-E/kT)}$, k being the Boltzmann constant. The move that increases the energy by E is accepted with the probability $p = e^{(-E/kT)}$ Brugmann [1993].

The simulated annealing is useful to find the global minimum of a function. To do this, simulated annealing plays moves randomly in state space. If the value of the function decreases after the move, the move is accepted. If the move increases the value of the function, the move is accepted with a probability which decreases exponentially with the increase of the valued of the function, and also decreases exponentially with the inverse of the temperature Brugmann [1993]. In the simulated annealing process, the temperature decreases with time, depending on the time given to the algorithm to find a solution.

Applying this concept to the game, the Brugmann [1993] proposed: "Each player decides beforehand in what order he wants to play his moves (taking into account that pieces may be played several times onto the same field after captures). A game is played to the end such that if a move in the predetermined

list is not possible, the next one is used. Each move is assigned the average value of all games in which it was played. This value is initially set to zero and then updated after every game. After every update the moves are ordered according to their average values. The strategy is therefore that moves which are more successful on average are deemed most important to be played first. After a large number of games both Black and White will have settled on a sequence of moves which are most successful with regard to the other. The best move for the one to move first, is the first one in the list".

Brugmann [1993] developed a program called Gobble to play Go in a board of 9×9 using simulated annealing to find the "best" move and with just one Go knowledge: "The computer only passes if either no legal move is available or all legal moves reduce the eye space of one of its groups from two to one". This prescription use the concept of the eye in the following way: as the field whose direct neighbors are all of the same color and whose diagonal neighbors contain no more than 1 stone of the opposite color (0 for the boards or corner fields)".

According to Brugmann [1993], the two eyes rule implementation was necessary for the program to play acceptable to the Go, but not sufficient just using random moves.

The results were successful playing two different strategies against Many faces of Go Brugmann [1993]. One of the findings of this program was that moves gave better results than moves onto the boarders or corners, so, good moves (center) were tried first at the beginning of the game.

The other finding was that even the board is symmetric, equivalent moves does not get similar values (for example for the corners), and the explanation was that because best move (center) are tried first, and bad moves are tried later at the end of the game, most of the times these moves are irrelevant and less correlated with the outcome of the game.

2.5.6 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is proposed by Chaslot *et al.* [2008] with the intention to unify Monte-Carlo simulation and Tree Search for Game IA. This technique was used in other games as classic board games as Go, modern board-games as Settlers of Catan, and video games as Sprints RTS.

The basis is the simulation of games where both the AI controlled player and its opponents play random moves. As Chaslot *et al.* [2008] mention, from a single random game (where every player selects his actions randomly), very little can be learnt, but from simulating a multitude of random games, a good strategy can be inferred.

These are the steps that MTC proposes to learn playing random moves. This can be observed in the Figure 2.12.

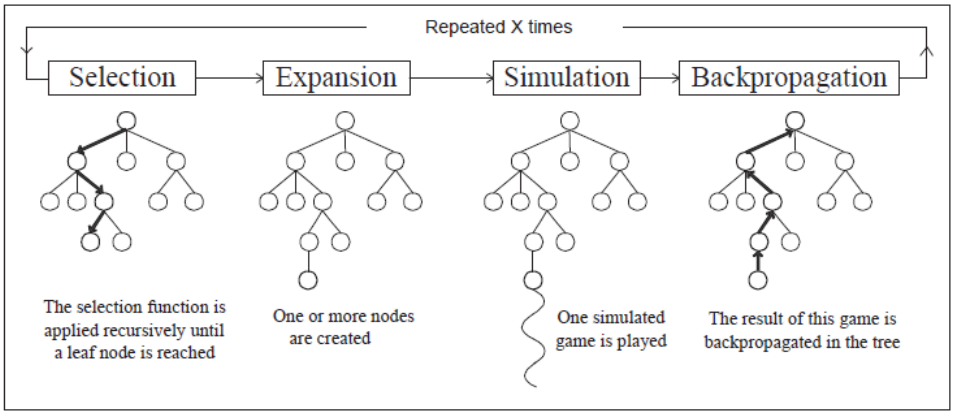


Figure 2.12: Monte Carlo Three Search Algorithm

- Selection. While the state is found in the tree, the next action is chosen according to the statistics stored, in a way that balances between exploitation and exploration. On the one hand, the task is often to select the game action that leads to the best results so far (exploitation). On the other hand, less promising actions still have to be explored, due to the uncertainty of the evaluation (exploration).

- Expansion. When the game reaches the first state that cannot be found in the tree, the state is added as a new node. This way, the tree is expanded by one node for each simulated game.

- Simulation. For the rest of the game, actions are selected at random until the end of the game. Naturally, the adequate weighting of action selection probabilities has a significant effect on the level of play. Chaslot *et al.* [2008] suggest to use some heuristic knowledge and give more probability to the some promising actions, otherwise the level of the monte-carlo program could be weak.

- Backpropagation. After reaching the end of the simulated game, it is updated each tree node that was traversed during that game. The visit counts are increased and the win/loss ratio is modified according to the outcome.

Finally, the game action executed in the actual game is the corresponding to the child that has been more explored. This team created a program called MANGO which is competitive Go program.

As Chaslo [2010] mentions, this technique has some other variants appeared in 2006 as Coulom [2006], Coulom used his variant to create a competitive MCTS program called CRAZY STONE. This program won the 9×9 tournament in 2006 Computer Olympiad.

2.6 Neuro-evolution

In this section is going to be reviews the definition of the artificial neural neurons and some neuro-evolution techniques.

2.6.1 Revision of Artificial Neuronal Networks

2.6.1.1 Artificial Neuronal Networks

The Artificial Neural Network is a mathematical model which is an abstraction of the characteristics of neurons and synapses of a biological brain. The brain is predominantly constructed out of countless interconnected neurons that individually 'fire' or activate a signal along a network of synapses as soon as a particular internal chemical threshold is exceeded. Repeating 'firing' strengthens certain connections, while a lack of 'firing' results in weakened connections, simulating a human learning.

Rosenblatt Rosenblatt [1958] published the first of many papers on computing with networks of artificial neurons in 1958, but few realize that Alan Turing wrote a important essay as early as 1948, entitled 'Intelligent Machinery'. This was written while Turing was working for the National Physical Laboratory in London, the paper did not meet with his employers' approval. Sir Charles Galton Darwin, the granson of Charles Darwin, called it a 'schoolboy essay' and wrote to Turing complaining about its 'smudgy' appearance Copeland [1999]. This paper was the first manifesto of Artificial Intelligence, but sadly Turing never published it.

According to Copeland [1999], Turing introduced a type of neural network that he called a 'B-type unorganized machine', consisting of artificial neurons, depicted below as circles, and connection-modifiers, depicted as boxes. A B-type machine may contain any number of neurons connected together in any pattern, but subject always to the restriction that each neuron-to-neuron connection passes through a connection-modifier.

In 1943 Warren McCulloch and Pitts McCulloch & Pitts [1943] attempted to demostrate that a Turing machine program could implemented in a finite network of formal neurons proposing an artificial neural network which was called Threshold Logic Unit (TLU). In the 60's some improvements was introduced as perceptron using linear threshold function, in the late 80's got more attention and was introduced neurons with more continue shapes, introducing some gradient descendant and other optimization algorithms and others. From there many different models of artificial neural networks was introduced and applied in different in real-world applications.

In the rest of the thesis is used artificial neural network or neural networks

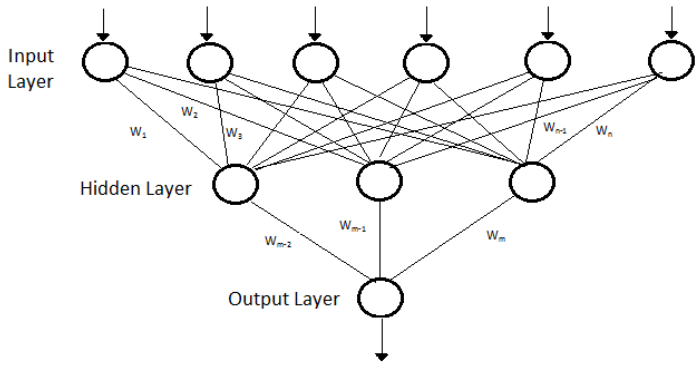


Figure 2.13: Representation of a simple Artificial Neural Network

indifferently for simplicity.

2.6.1.2 Neural Network Architectures

A wide variety of Neural Networks (NN) architectures exist, which can differentiate each other in connection topology, training method, and the learning algorithm. The connection topology defines how the processing units or nodes are connected to each other, the training method is about how the NN learns, and the learning algorithm defines how error is measured during the training process Brabazon & O'neil [2006].

The Figure 2.13 show a simple a artificial neural network. This structure has three layers of neurons interconnect by weights, the input layer, the hidden layer and the output layer. The input layer size vary depending on the features to be considered for the network. The hidden layer interconnect the inputs to the output layer. The size of the hidden layer vary depending on the problem domain. Each neuron in the hidden layer use of an activation function. The activation function is used to calculate the 'firing strength' (output) of each neuron, given the weighted sum of all connected input neurons.

There are three common NN structures: multi-layer percepton, radial basis function networks, and self-organizing maps. Many others NN structures can be developed using the about three common structures Brabazon & O'neil [2006].

2.6.1.3 Activation Functions

Some activations functions used in the NN are the followings:

- Linear Function :

$$F(\tau) = \beta\tau \tag{2.1}$$

- Sigmoid Function :

$$F(\tau) = \frac{1}{1 + e^{-\lambda\tau}} \tag{2.2}$$

- Hyperbolic tangent Function :

$$F(\tau) = \frac{2}{1 + e^{-\lambda\tau}} - 1 \tag{2.3}$$

- Gaussian Function :

$$F(\mu) = e^{-\mu^2/\sigma^2} \tag{2.4}$$

In this thesis has been used the Sigmoid Function for all the experiments performed.

2.6.2 Revision of Some Neuro-Evolution Techniques

The neuro-evolution (NE) is a form of machine learning that uses evolutionary algorithms to train artificial neural networks (NN). These gave good results in some task as described in Stanley & Miikkulainen [2002a]. A distinction is made between NEs that evolve the values of the connection weights for a network of pre-specified structure or topology vs. NEs that evolve the topology of the network in addition to the weights. Networks that have both their connection weights and topology evolved are referred to as TWEANNs (Topology & Weight Evolving Artificial Neural Networks).

Evolutionary algorithms operate on a population of genotypes, and in neuro-evolution a genotype is some representation of a neural network (a phenotype). In Direct encoding schemes the genotype is the same as the phenotype, every neuron and connection is specified directly and explicitly in the genotype. In contrast, in indirect encoding schemes the genotype specifies rules or some other structure for generating the neural network.

There are different NE techniques for evolving weights and weights and the structures of neural networks, the ones reviewed in this work are the following: SANE (Symbiotic Adaptive Neuro-Evolution), ESP (Enforced Sub-Population), NEAT (Neuro-evolution of Augmenting Topologies).

In NE the neural networks make use of networks of artificial neurons (in the future just as neurons), which are processing units that compute some simple function of their input values, producing one or more output values. Typical neurons compute a weighted sum of their numerical inputs, which contain the information of the inputs to the system, then apply a threshold function or a sigmoidal function, if the output function must be differentiable, to produce the output of the system. In NE the networks of neurons are trained to perform a particular task by repeatedly attempting the task and providing corrective feedback.

2.6.2.1 Enforced Sub-Population (ESP)

The main feature of ESP applied to Go is the definition of some regions (i.e. 3x3 lines) as different populations which are evolved as separate population, getting diversity and specializing some neurons to some specific problems or to specific regions (i.e. play in the corner). According to Perez-Bergquist Perez-Bergquist [2001] in general ESP has been more effective that SANE because with the same conditions in the experiment SANE needed networks of 300 neurons to defeat the Gnugo (a known player), but ESP only needed 10 neurons (2 hidden layers). But, the issues faced using ESP is that cant be scalable to bigger boards, for example, good players that evolved in a board of 7x7 were not possible to be used in 9×9 board, in fact the networks that had better performances in the evolution were the networks that started from scratch and not moved from one small board to big board Perez-Bergquist [2001].

2.6.2.2 Neuroevolution of Augmenting Topologies (NEAT)

NEAT is a technique to evolve weights and the structures (topology) for NE proposed by Keneeth Stanley & Miikkulainen [2002b] which belongs to TWEANN (Topology and Weight Evolving Artificial Neural Networks) techniques. The main benefits are the following:

- Dont lose the time to find manually the best structure of the neuron population
- Can evolve from simple structures to more complex structures as in the nature, in similar way that evolving simple strategies to more complex strategies.
- Protecting the innovation through speciation or niching, using historical markings to identify which genes are coming from the same root or parents. The idea is to divide the population into species to compete into their niches

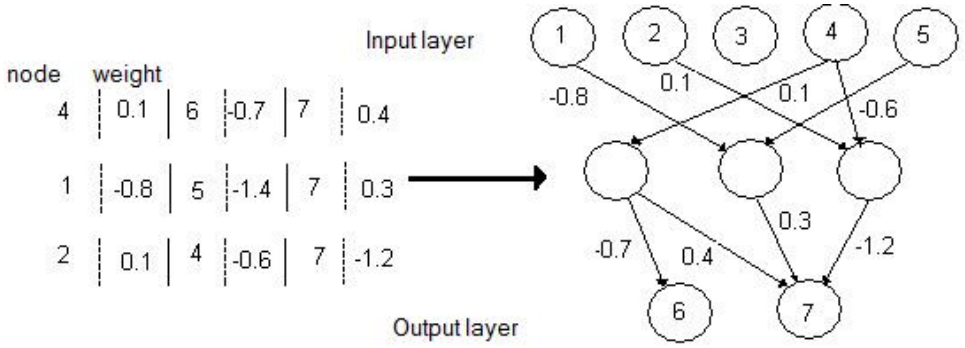


Figure 2.14: Structure of neuron in SANE

protecting the innovation (new structures from mutation) and compete later in the large population.

- Mating similar species using the historical marking of each gene (using the differences function).

According to the author, two genes with the same historical origin must represent the same structure Stanley & Miikkulainen [2002b], although with different weights, since both are derived from the same ancestral gene at some point in the past. To track this historical origin the author propose a global innovation number which is added to the system every time that a new gene that appears. It was demonstrated in the task of pole balancing problem that NEAT is more efficient to others methods as ESP, SANE or CE Gruau *et al.* [1996].

2.6.2.3 Symbiotic Adaptive Neuro-evolution(SANE)

SANE is a NE proposed by Moriarty Moriarty [1997] where the weights and inputs/outputs of the structure of the networks are evolved. This was proved in different problem with good results as Moriarty & Miikkulainen [1998b] , Moriarty & Miikkulainen [1996a]. The neuron contains nodes (that could be the input or the output to the hidden layer) and weights that connect the hidden layer with the input/output. Figure 2.14 shows how the neuron and blueprint network are built.

The size of neurons is the number of nodes and weights that connect that neuron. The network or blueprint networks points to a set of neurons from the population of neurons in every generation. The relationship between the neuron and the blueprints can be observed in the Figure 2.15. The same neuron can belongs to more than one blueprint. The offspring of the members of the population is a sexual offspring being the parent the best neurons from the previous

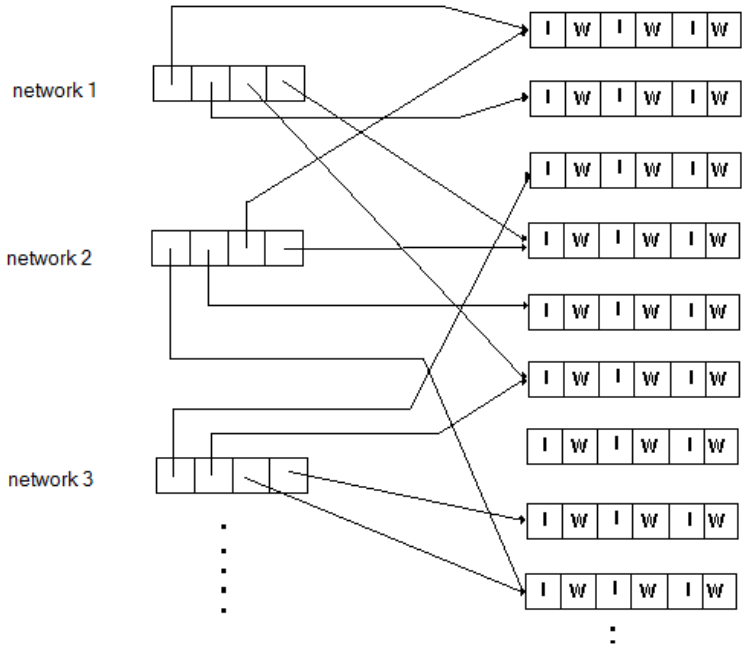


Figure 2.15: Structure of blueprint or network

generation. The activation of the neuron is calculated between the sum of the all input and output multiplied by their weights and passed through this sigmoid function:

$$\sigma(x) = (\frac{1}{1+e^{-x}}) \tag{2.5}$$

SANE searches and maintains the best strategies or solutions to the problem in neural networks and evolve them through generations using genetic operators. SANE has two parts, the evaluation and the reproduction phase. In the evaluation part, SANE simultaneously evaluate the blueprint networks and neurons. The blueprint networks are evaluated by the performance to solve problems. The neurons are evaluated based on performance of the network in which the neurons are participating. The basic steps in evaluation phase are the followings Moriarty [1997]:

- Per each neuron n in the population Pn (initialization)
  - n.fitness = 0
  - n.participation= 0
- Per each blueprint b in the population of Pb
  - neuralnet = decode (b)

  - b.fitness = task (neuralnet)

- Per each neuron n in b
  - n.fitness = n.fitness + b.fitness
  - n.participation = n.participation + 1

- Per each neuron n in the population Pn
  - n.fitness = (n.fitness/n.participation)

In the reproduction part, the neurons that participate in networks with best fitness are maintained in Hall of Fame and crossed replacing neurons with worse fitness, and finally mutation is applied for the worse neurons. The best blueprints are maintained in Hall of Fame and used to be reproduced replacing blueprints less ranked, and finally applied mutation operators to these blueprints less ranked.

2.7 Computer Go and the State of the Art

According to Bouzy & Cazenave [2001], apparently the first Computer Go program coded was written by Lefkovitz Lefkovitz [1960], and the scientific paper on computer Go was published in 1962 by Remus [1962]. The first program to defeat an absolute beginner was by Zobrist in 1968, who in 1970 also wrote the first Ph.D. thesis Zobrist [1970] on computer Go. The second thesis in Computer Go was written by Rider [1971].

The first Go programs were exclusively based on an influence function: a stone radiates influence on the surrounding intersections (the black stones radiate by using the opposite values of the white stones), and the radiation decreases with the distance. These functions are still used in most Go programs. For example, in Go Intellect, the influence is proportional to $\frac{1}{2}$ distance, whereas it is proportional to $\frac{1}{\text{distance}}$, in Many Faces of Go Bouzy & Cazenave [2001].

.

In the mid 1980s, with a a million-dollar prize for the first computer program to defeat a professional Go player offered by Mr. Ing, research in computer Go received a big boost, even that prized expired at 2000 Muller [2000], the research in the computer Go still continue till now. According to van der Werf [2004] at the beginning of the century the number of publications on Go was at least at a par with those on Chess.

Since then, computer Go programs has been evolve applying new techniques, as described in the previous sections, with many tournaments around the world.

Nowadays there are many tournaments and congresses where professional players can play against machines, and in these tournaments, the machines sometimes have beat professional players, but, even that, the computer Go still is a challenge to the Artificial Intelligence, motivating to the research community and the fans of this game to propose new techniques to get more strong players.

The complexity of the game as it was described by Allis [1994] is one of the reason why till now is not difficult to get strong players. The approach to use minimax search based, as in case of Chess, is not valid in Go. Usually the approach in Chess was calling fast and cheap evaluation function, which in case of Go, the evaluation function is slow and complex.

As van der Werf [2004] mentions, as a consequence chess programmers are often surprised to hear that Go programs only evaluate 10 to 20 positions per second, whereas their chess programs evaluate in the order of millions of positions per second.

A direct consequence of the complexity of the evaluation functions used in computer Go is that they tend to become extremely difficult to maintain when the programmers try to increase the playing strength of their programs van der Werf [2004].

At the time of written this thesis (Autumn 2012), the prominent computer go programs include Fuego, based in MonteCarlo, KCC Igo from North Korea, Handtalk/Goemate, developed in China by Zhixing Chen, Go++, programmed by Michael Reiss, Many Faces of Go, programmed by David Fotland, GNU Go is a free computer Go program, available at GNU Go which has won many computer Go competitions as well.

MoGo is another computer program which has some records, this is based in Monte Carlo and Tree Search (TS) algorithm (which is originally based in the UCT algorithm) Mogo. This computer go program on August 7, 2008, running on 25 nodes (800 cores, 4 cores per node with each core running at 4.7 GHz to produce 15 Teraflops) of the Huygens cluster in Amsterdam beat to professional Go player Myungwan Kim (8p) in a nine stone handicap game on the 19x19 board on the KGS Go Server Mogo.

In 2009, MoGo made two new world records by winning a 19 by 19 game with 7 handicap stones against the 9P professional Go player Jun-Xun Zhou and a 19 by 19 game with 6 handicap stones against the 1P professional Go player Li-Chen Chien WCCI 2010

So, MonteCarlo Tree Search has been demonstrated to be a good technique for computer Go, but the problem is has to be incorporated some game knowledge in the Simulation phase to increase the performance of the program. As Chaslot *et al.* [2008] mentions, if all legal actions are selected with equal probability, then the strategy played is often weak, and the level of the Monte-Carlo program is suboptimal.

Other promising techniques accepted for automatic learning knowledge are self-play Temporal Difference Learning (TDL) and Co-evolution Learning strategies (CEL). As it was discussed these techniques were applied in some games as by Chellapilla & Fogel [1999] in checkers, Runarsson & Lucas [2005] in Go, Lucas & Runarsson [2006], Chong *et al.* [2005] in Othello , Kotnik & Kalita [2003] in Gin rummy, Darwen [2001] in Backgammon. These two techniques were compared and according to them, both methods improved its play without the need to play against a human-designed player, but, the according to these authors is that TDL learns much faster, but that CEL eventually achieves a higher standard of play.

According to Runarsson & Lucas [2005], the reason why TDL is learning faster than CEL, is because TDL method uses more information from each game to assign credit to particular parameters of the evaluation function, which make the search space more directed than the relatively evolutionary approach.

Even Self-play TDL learn fast, CEL outperform TDL according to the results of the experiments performed. The reason for this is that CEL players are exposed to more realistic game play variation than is not possible with a single self-play TDL player Runarsson & Lucas [2005].

In this thesis is proposed to follow the approach of CEL methods and propose news techniques to ensure that CEL really happens, some of these techniques were introduced in Zela & Zato [2011]. For the experiments of the thesis was used SANE method, and some known computer Go players, Wally, and weak Go program, and Gnu Go, which are more strongest, both of them free available.

SANE has demonstrated to be good technique for automatic learning knowledge because explore a big search space. This has been demonstrated in the following problems as Moriarty & Miikkulainen [1998a], Moriarty & Miikkulainen [1996b], Moriarty [1997], Lubberts & Miikkulainen [2001] and others.

SANE is a good approach to the credit assignment problem, when there are difficulties to identify which move in the sequence of moves made winning the game.

As Moriarty [1997] mentions, often the best strategy is not to maximize each immediate payoff, because some actions that produce high immediate payoffs may enter states from which high future payoffs are impossible. The decision strategy must consider both immediate and future payoffs of actions to optimize the total payoff.

The problem as how to incorporate local games in the global game are going to be discussed in this thesis and how this approach probably should not applied to this game or partially applied, or at least the approach should be to see the Go board as one game and not multiple games. As Bouzy & Cazenave [2001] mentions, even you have a good sub-games model of a global game, the sub-games are greatly depend on one another, and the independence of the sub-games are a prerequisite to apply combinatorial game theory applied to Go game.

One of the issues that we can face to apply co-evolutionary learning process is the pathologies that can arise during this process, which impact the evolution process not allowing to population to evolve. Some of these pathologies are loss of gradient and disengagement, cycling dynamics and intransitivity, and forgetting. In this thesis the author is proposing some techniques that are going solve the co-evolutionary pathologies, some of these techniques introduce more diversity in the population as a replacement immigration rate which will replace worse performance solution by new solutions. There are other techniques introduced by the author of dynamic size of population of neural networks, a new competitive fitness sharing to evaluate the performance of the solution discovered during the co-evolution and others. The other issue observed in co-evolutionary process is the lack of general strategies learned, which can prepare to computer players for new situations in which the computer player has not been trained. These issues in the co-evolutionary learning process is discussed in the next chapters.

The co-evolutionary learning process in the last years have been applied to different fields as economics, logistics, defense, robotics, design of hardware, multi-objective optimization, security trading in Papacostantis [2009] , high frequency trading in Adamu & Phelps [2010], and other fileds as co-evolving ontologies in Kupfer & Eckestein [2006] solving some complex problems which can not be solved with other search algorithm. There are different approaches in the CEL that has been applied to solve some complex problems, some of them can be found in Lipson & Pollack [2000], Rawal *et al.* [2010], Uchibe & Asada [2006], Salcedo-Sanz *et al.* [2007], Brabazon & O'neil [2006].

Finally, as some authors discussed previously, the solution to the Go game is to identify new techniques that still are not available. Because the complexity of this game still the author believe there many things for improvement and this thesis is a contribution in that direction. But, at the same time, the author believe that the techniques proposed in this thesis can be applied to other fields different to the Go game. At it was discussed, many of the techniques which were used initially in games are re-utilized to be applied in real and more complex problems.

Chapter 3

Co-evolutionary Learning

This chapter introduces the evolution and co-evolutionary as learning techniques and its relation with the nature. It reviews some definitions as evolution, co-evolutions and some dynamics that can be observed in the nature as red-queen dynamics. The sections 3.3 and 3.4 reviews some definitions as evolutionary computation, and competitive and cooperative co-evolution respectively, describing the progress that have been observed in these areas.

In the section 3.5 is discussed the advantage of using co-evolutionary techniques as learning technique. Some advantages as the necesity to create some deterministic players, avoid the inductive bias, or i when is not possible or very expensive to obtain all or a very good set of test cases. In the section 3.7 is presented the pathologies that can be found in a co-evolutionary process, and why it is important to monitor and measure the co-evolution progress. It will be discussed why the presence of these pathologies can be the reason why the co-evolution is not progressing.

Finally, in this chapter is introduced how solution concepts are defined in co-evolution environment, and some of fitness measures used in the co-evolutionary learning process.

3.1 Introduction

In biology, the are some definitions of co-evolution, as for example the one provided by Jansen [1980] as an evolutionary change in a trait of the individuals in one population in response to a trait of the individuals of a second population, followed by an evolutionary response by the second population to the change in the first. In genetics there is another definition, a change in one gene in one species simulates an evolutionary change in one gene in a second species, which in turn simulates another evolutionary change in the first species, and so on.

Co-evolution can be microscopic, as correlated mutations between amino acids in a protein, or as macroscopic as co-varying traits between different species in an environment, as the interactions that occur in the nature as parasite-host, predator-prey or asymmetric arms-race iterations Dawkins & Krebs [1979], where two different species or populations compete against each other. Each specie in a co-evolutionary relationship is exposed to selective pressures on the other, thereby affecting each other evolution. The co-evolution only exist when the species interact with high frequency.

This biological concept has been applied to various fields as computer science, sociology, economy, and others.

3.2 Red-Queen Dynamics and Arm Race

Valen [1973] proposed this principle: "for an evolutionary system, continuing development is needed just in order to maintain its fitness relative to the systems it is co-evolving with".

This is based on the observation to Alice by the Red Queen in the book Through the Looking Glass of Carroll Lewis, in which he says "Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else-if you ran very fast for a long time as we have been doing."

Even the original idea of this principle is that the co-evolution could lead to situations for which the probability of extinction is relatively constant over millions of years Valen [1973], this can explain that dynamics of that every improvement in one species will lead to a selective advantage for that species, variation will normally continuously lead to increases in fitness in one species or another. However, since in general different species are co-evolving, improvement in one species implies that it will get a competitive advantage on the other species, and thus be able to capture a larger share of the resources available to all.

This means that because of the fitness increase in one specie, this will reduce the fitness of the other specie. So, the only way that the species involved in the competitive co-evolution can maintain its fitness relative to the others is by in turn improving its design as in parasite-host interaction.

This can be observed in the following Fig 3.1, which describe the Red Queen dynamics from a computer simulation for the host-parasite co-evolution, which lead to the continues oscillation of the genotype of the parasite-host. In this figure is observed that there is a moment where the parasite have improved their fitness which lead to the decrease of the genotype of the host, which simultaneously lead to the reduction of genotypes of the parasite.

The other example of this arm race effect are between predators and prey, where the only way predators can compensate for a better defense by the prey

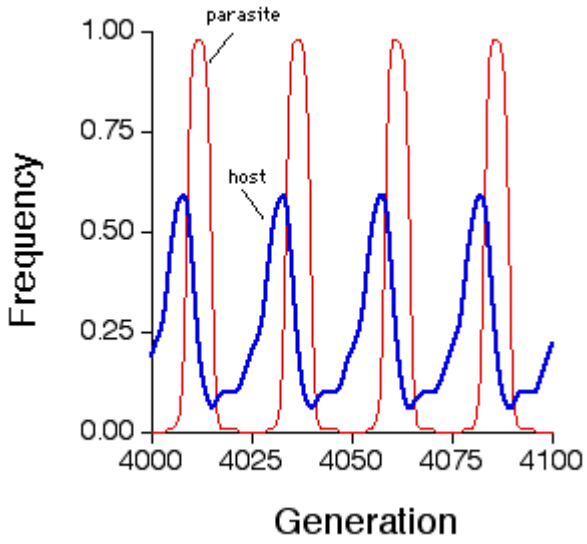


Figure 3.1: Red-Queen Dynamics

(e.g. rabbits running faster) is by developing a better offense (e.g. foxes running faster). In this case we might consider the relative improvements (running faster) to be also absolute improvements in fitness.

3.3 Evolutionary Computation

The origins of Evolutionary Computation (EC) come back to 1950s Back *et al.* [1997], but just from the 80s and 90s got more attention. The applications of EC are varied from engineering, natural sciences, economic, finance, business Back *et al.* [1997], chemistry (i.e. find a new molecules to find the cure for AIDS), military Fogel [2000], hardware design Salcedo-Sanz *et al.* [2007] and off-course to solve some games (i.e. checkers Fogel [2000], Otello Lucas & Runarsson [2006], Go Lubberts & Miikkulainen [2001]).

For Chong [2007] the EC is the study of computational systems that consist of algorithms and procedures inspired and motivated from the process of natural evolution. Other definition can be find at Back *et al.* [1997], where EC mimic the process of natural evolution, the driving process for the emergence of complex and well adapted organic structures. According to Yao [1999] EC techniques can be used in optimization, learning and creative design, which don't require domain knowledge to use it, but this can be incorporated into the EC. Coello [1998], Chen & Yao [2010] applied EC to multi-objective optimization problems, others authors applied in classification problems when neural networks representation

are used.

According to Fogel [2000], the field of evolutionary computation is addressing many problems that were previously beyond reach. Potentially, the field may fulfill the dream of artificial intelligence: a computer that can learn on its own and become an expert in any chosen area.

One of the advantage of Evolutionary Search is the flexibility and adaptability to the task at hand, in combination with robust performance (although this depend on the problem class) and global search characteristics Back *et al.* [1997].

According to Fogel [2000], Chong [2007], Handa *et al.* [2006] Evolutionary Algorithms (EA) can be applied to solve problems that traditional optimization-based search algorithms cannot be used since it is very difficult or impossible to construct an absolute measurement of solution quality (i.e. the fitness function).

The majority of the current implementations of EAs descend from three strongly related but independent developed approach: genetic algorithm (GA), evolutionary programming (EP) and evolution strategies (ES) Back *et al.* [1997]

Although the development of GA dates from 60's, they were first brought to the attention of a wide audience by Holland. GA is an optimization algorithm inspired by a biological metaphor and applied a pseudo-darwinian process to evolve good solutions to real-world problems. GA adopts a populational unit of analysis, wherein each member of the population encodes a potential solution to the problem interest Brabazon & O'neil [2006].

Some key points that have to be decided when EC, i.e. GA, is applied for searching the space of solutions to some problems; the representation of the structures to evolve (i.e. binaries, numbers, neural-networks, rules, etc), the evolutionary operators as mutation and recombination (or crossover), the self-adaptation methods for the evolutionary operations (or just being constants) which search the own strategies for these parameters.

Other things that have to be decided is the selection method of individuals for the next generation. The (μ, λ)-evolutionary strategy uses a deterministic selection schema, in which the μ parents create $\lambda > \mu$ offspring by recombination and mutation operators, and the best μ offspring individuals are deterministically selected to replace the parents. In this case, the individuals at generation t+1 will perform worse than in generation t Back *et al.* [1997]. The other evolutionary strategy is $(\mu + \lambda)$ which select the μ survivors from the union of the parent and off-spring, such that a monotonic course of evolution is guaranteed. Both strategies are valid and produced good results as per different authors.

The individuals of the population are evaluated based on a fitness function which will guide in the search space to find the solutions to the problem. The better individuals performs under certain environment conditions (solve the problem) the greater the chances that these individuals can live longer and generate offspring which will inherited his parental genetic information.

The general algorithmic framework for EAs can be described in the following Back *et al.* [1997], Yao [1999], Chong [2007]:

1. Initialize the population, P(t = 0).
2. Evaluate the fitness of each individual in P(t).
3. Select parents from P(t) based on their evaluated fitness.
4. Generate offspring from parents to produce P(t + 1).
5. Repeat steps (2-4) until some termination criteria are reached.

The framework emphasizes two specific features that distinguish EAs from other search algorithms Yao [1999]. First, all EAs are population-based whereby at any time, an EA is operating on at least a single population of individuals (e.g., candidate solutions). Second, there are mechanisms of information exchange between the population from one generation to the next Chong [2007].

Neuro-evolution (NE) is another EC technique, which uses evolutionary algorithms to train artificial neural networks as it was described in the previous chapter. In NE the neural networks make use of networks of neurons, which are processing units that compute some function of their input values, producing one or more output values. A Typical neuron compute a weighted sum of their numerical inputs, then apply a threshold function, sigmoid function or any other function, if the output function must be differentiable to produce the output. In NE the networks of neurons are trained to perform a particular task by repeatedly attempting the task and providing corrective feedback.

3.4 Competitive and Cooperative Co-evolution

Probably the firsts co-evolutionary application of solve some problems were introduced by Kauffman [1993], Angeline & Pollack [1993], Hillis [1990].

The co-evolutionary learning can be classified as cooperative and competitive co-evolution, but that does not mean that we should select one of them in the co-evolutionary learning process to solve our problems, specially in complex problems as it is discussed in this thesis. In the cooperative co-evolution usually the entire population is a complete learning system, where each individual in the population represents a part of the solution and that they must work together to form a complete solution. In the competitive co-evolution usually each individual in the population represents a complete solution.

Generally cooperative co-evolution has to solve some problems as decomposition and credit assignment. Some works in cooperative co-evolution can be found

in Potter & de Jong [2000], which proposes a technique of evolving solutions in the form of interacting co-adapted subcomponents, where each of the subcomponents represent part of the solution. In Potter & de Jong [2000] is described an architecture for evolving subcomponents as a collection of cooperating species as is shown in the Figure 3.2.

The architecture models an ecosystem consisting of two or more species. As in nature, the species are genetically isolatedmeaning that individuals only mate with other members of their species. Mating restrictions are enforced simply by evolving the species in separate populations

The species interact with one another within a shared domain model and have a cooperative relationship. Each species is evolved in its own population and adapts to the environment through the repeated application of an EA.

Different methods are chosen to select the representative of each specie to collaborate, as selects the best members of each specie or select them randomly.

As in some complex problems is not possible to decompose by hand a problem or when the evolution is stagnated, new niches of species (solutions) should emerge to contribute to the solution of the problem, so, in this technique new populations are added dynamically. These new added individuals are evaluated based in the overall contribution to the ecosystem. The species that do not make contributions are destroyed.

The other key feature of this method is that heterogeneous representation of the populations are supported, as populations of neural networks, rules, vectors and others. These species can be evolved using any kind of EAs.

The other application of cooperative co-evolution is found in SANE Moriarty [1997], ESP Perez-Bergquist [2001], Yong & Miikkulainen [2007] described previously, where a individuals from a neuron population is grouped in blueprints to solve a problem. Every member of this neuron population has to cooperate in the blueprint to get the best solution to the problem. The other approach can be found in Axelrod & Dion [1988] applied to the prisoner's dilemma for which under certain conditions prisoners cooperate instead of defect in long term evolutionary approach.

In competitive co-evolution, can be used a single population or multiple populations. In case of single-population, each individual competes against each other for survival, changing the roles of being solutions and test cases during the evolution. One example is the co-evolutionary learning of symmetric two-player games such as the prisoner's dilemma Axelrod [1987].

Other work in competitive co-evolution can be found in Rosin & Belew [1997], where two populations are competing against each other as parasite-host interaction, where fitness of individuals of one population are in direct competition with some individuals of other populations and learning their own strategies.

Some of the problems identified by Rosin & Belew [1997] are that in some

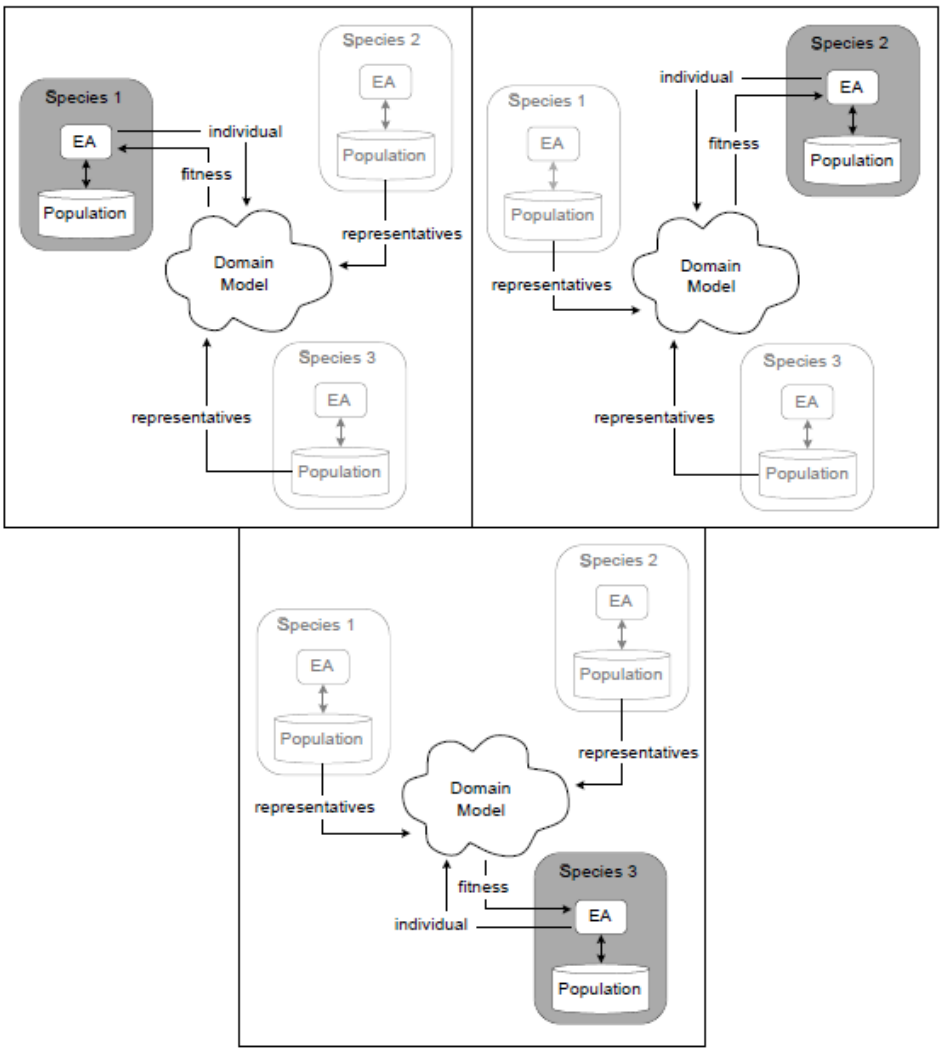


Figure 3.2: Cooperative co-evolutionary model of three species shown from the perspective of each specie

co-evolutionary learning process, is important to maintain an adequate diverse set of test cases or parasites, and appropriated for the current level of host. The parasites must be neither so difficult that the host rarely win, not so easy that the host always win.

The second issue is that even a finite number of genotypes are in the population, and even the genetic operators as crossover and mutation can introduce new genotypes, it is important no lost some genotypes that could contributed significantly to the solution, even now is not doing it, maintaining some kind of niches.

There is another strong assumption in co-evolutionary strategies, that new evolved solutions can defeat to the previous solutions, if that assumption is not guarantee, the system can get stuck on weak strategies that defeat each other in a cycle. For Rosin & Belew [1997] the "optimal solution" is one that defeat all possible opponents.

To solve these issues, Rosin & Belew [1997] proposed three different innovations in co-evolving these two populations as competitive fitness sharing, shared sampling, Hall of Fame, which will be discussed in detail in the next chapters.

According to Chong [2007] both styles of co-evolution (competitive and cooperative) can use multiple, reproductively isolated populations; both can use similar patterns of inter-population interaction, similar diversity maintenance schemes, and so on. Aside from the novel problem-decomposition scheme of cooperative co-evolution, the most salient difference between cooperative and competitive co-evolution resides primarily in the game-theoretic properties of the domains to which these algorithms are applied.

These two classes of co-evolutionary learning has been applied to different games as in Axelrod [1987], Axelrod & Dion [1988], Lubberts & Miikkulainen [2001], Runarsson & Lucas [2005], Darwen [2001], Pollack & Blair [1998], Yao [1997], Potter & de Jong [2000] and others.

3.5 Advantage to Using Co-evolution Learning

The author believe that there some reasons why use co-evolutionary learning approach as an alternative to other learning algorithms discussed previously. In the sections sections is going to discussed five reasons:

- As it was discussed in Zela & Zato [2011], it is needed to apply co-evolution to avoid create systems with deterministic strategies. There exist some domains that are open-ended in which exists an infinity of possible behaviors in which the co-evolution is needed Ficici [2004].

- Co-evolutionary learning require less (human-supplied) inductive bias than

other search methods Ficici [2004]. This can produce strategies not known previously.

- There exist domains in which there is not possible to provide all set of test cases. As Ficici [2004] mentions, there exist domains that intrinsically require co-evolution because these are interactive in nature, such as games.
- Co-evolutionary algorithms make more efficient use of finite computational power by focusing evaluation effort on the most relevant tests. For example, those that best distinguish the quality of potential solutions Ficici [2004]. Co-evolutionary learning always trying to discover high level strategies.
- Maintain the diversity of the solutions (population) during the solution space search.

3.5.1 Avoid Deterministic Players

As it was discussed in Zela & Zato [2011], the training of computer Go player against a computer Go player with no randomly movements can create deterministic strategies, in which players learn how to beat that player with which was trained. This phenomenon was observed by other authors as Lucas & Runarsson [2006], Darwen [2001],Lubberts & Miikkulainen [2001] and others.

In some domains, as in Go game, the number of possible strategies are almost infinity, which can be considered as an open-ended domain. To solve this issue some authors proposed self-play TDL technique which has obtained good results, but co-evolutionary approach has achieved higher standard of play according to Lucas & Runarsson [2006].

In Zela & Zato [2011], the computer Go player using SANE was trained against two known free available computer Go players; Wally, which is a weaker computer Go player, for which it just was needed some few generations to beat it. Other trainings were performed against GnuGo 3.8 level 5 (because the low time response whether high level of play of GnuGo is used), for these experiments were needed more generations but finally the system beat GnuGo 3.8 level 5. For the last experiments were applied some improvements to the previous algorithm which is going to be discussed in the next section.

But in both cases, the computer Go player obtained a low level of play against other known computer Go players and non professional human players. So, co-evolutionary learning strategy was needed to fix this issue as will be discussed in the next chapters.

According to Ficici [2004], perhaps the most dramatic illustration of co-evolution potential is Sims [1994], about the co-evolution of virtual creatures

that compete to obtain control of a cube. The domain in that work is essentially open-ended.

3.5.2 Avoid the Inductive Bias

Using co-evolutionary learning can be avoided the human-supplied inductive bias in the system. This is very useful when is not possible to include all the knowledge or strategies in the system, or in the worse case, when we introduce knowledge that are not necessarily the best ones which can guide to wrong solution space. Using co-evolution, the system can discover by itself this knowledge. One of the reason why is that is because as two different populations (i.e. two or more and if it is cooperative or competitive co-evolution) of species are interacting and evolving frequently, which induce to the system to discover new (or better) strategies which is not possible in other way.

There are some works that used co-evolutionary learning to avoid this inductive bias. Chellapilla & Fogel [1999] introduced a strategy that plays expert-level checkers through co-evolution using neural networks. When given an appropriately flexible substrate, co-evolution can discover the salient characteristics of a good strategy that are otherwise difficult to articulate by hand Ficici [2004].

Lipson & Pollack [2000] co-evolved a robot morphology and control (called body-brain co-evolution). body-brain co-evolution provides a way to gradually achieve both complex morphology and competent control of that morphology, without the considerable inductive bias that is otherwise required. Angeline & Pollack [1993] used co-evolution to obtain Tic-Tac-Toe strategies. A particularly interesting aspect of this work is that the evolving players are not informed about of the games rules, and the system was able to learn to play legal Tic-Tac-Toe movements as part of their learning process. If a player made an illegal move, then that was punished to avoid in the future, so, the system was able recognize that illegal moves. Building such a capability represents much less inductive bias than building a legal move generator.

3.5.3 It is Not Possible Provide All Test Cases

There are some domains in which is costly or even not possible to provide all set of test cases to give the opportunity to the system to learn from there. So, co-evolutionary learning is the best approach to discover these test cases (i.e. the evolution of parasites in the host-parasite interaction).

For example, in games as Go, it is not feasible to provide to all the test cases to train the computer go player because of the huge number of strategies than can be created in the game. This domains are considered intrinsically interactive, as other games, in which the source of the tests used to recognize solutions and

direct search is the domain itself Ficici [2004]. It is possible to use strong computer players to used a trainer, or strong computer players playing some random moves, but at the end, the system will learn only how to beat that specific player, as it was discussed previously.

As Ficici [2004] mentions, How can be possible to discover the perfect strategy for the game of checkers?, and How do we construct a metric of quality for a checkers player?. There are some approaches to these problem discussed as:

- Use all possible strategies for testing, which in open-ended domain or domain which almost infinite of strategies are not feasible as in case of Go;
- Use only perfect strategies for testing, which creates another issues as Juille & Pollack [2000] described, that testing against only the hardest cases provides too little gradient for search to progress. According to Juille & Pollack [2000] the application of co-evolution can identify an "ideal" trainer which it is a significant improvement over previously known best rules for this task.
- Use a sample (random or otherwise) of strategies for testing, which in the case of complex domain as in Go, the results are poor players, as will be discussed in the next chapters; and finally
- Use a hand-built metric of goodness that measures aspects of a strategys play for testing. The issue with this approach is to identified how many test cases are representative for all the possible behaviors.

But as Ficici [2004] mentions, regardless of our approach, it is needed to have a solution concept in mind to integrate outcomes over multiple tests.

So, co-evolution provides a response to the previous issues discussed, in which the new strategies are discovered interacting populations, using the outcome as indicators of over-all quality Ficici [2004]. we can regard co-evolution as a heuristic to obtain a metric of merit for interactive domains Ficici [2004].

There are some works for intrinsically interactive domains as Tesauro [1993] where even he used self-play TDL algorithm, according to Ficici [2004], because of the nature of the backgammon game, the interaction between one player against it self, always accumulate skills, that is why this technique god a superior computer player. The other work is Chellapilla & Fogel [1999] which co-evolved a neural network to evaluate board positions for game-tree search. They used conventional co-evolutionary algorithm and were able to obtain an expert-level checkers player.

3.5.4 Efficiency in Searching Solutions

This advantage is based on the assumption if the trait are heritable (i.e through crossover in sexual reproduction). Thus, if the parent pass a test case, the offspring or child will pass it as well, so that offspring test cases can be omitted. but, this general property not only achievable using co-evolutionary algorithm, it fact, this is evolutionary algorithm's property.

If a evolutionary algorithm selected to solve a problem can be made this assumption, the search of solutions will be efficient, but, the risk is if this assumption can be applied.

In the co-evolutionary approach proposed in this dissertation the test cases are networks of neurons (or blue-prints), which are created from the population of neurons. The big challenge in applying co-evolutionary approach is to decide if the test case (or blue-print) create in an previous generation can be created again in the future in other generation. The author thinks that previous test cases which were discarded (because where not selected for reproduction or were mutated) in the future could appears and compete or being tested against another set of opponents in a different context.

The only way to avoid to repeat test cases is to ensure that the best test cases saved in every generation contain the best strategies of that generation, and these strategies are not discarding any knowledge, in fact containing the knowledge of the test cases that was discarded in that generation. If we can ensure this assumption, we can incorporate some constraints in the co-evolutionary algorithm to not repeat test cases (network of neurons) in the competition of the next generations, but this algorithm can be very hard and consume a lot of computing resources during the evolution process.

The problem increase if the domain to find the solutions are open-ended, where there are almost infinite possible solutions. Ficici [2005] called to the property monotonicity, where if it is possible repeatedly query a search method (at any time during its execution) on any single run, then the quality of the solution returned by the method should improve monotonicallythat is, the quality of the solution at time t + 1 should be no worse than the quality at time t.

The assumption is that exist the possibility to have almost an infinite memory where the solutions discovered in the previous generation are not discarded, which in real-world is very difficult.

According to Ficici [2005], the solution concept conventionally implemented in a co-evolutionary algorithm is not monotonic; that is, it does not allow us to expect monotonically improving estimations to be produced by the algorithm, even the assumption that knowledge is never discarded, but under certain conditions can be approximated.

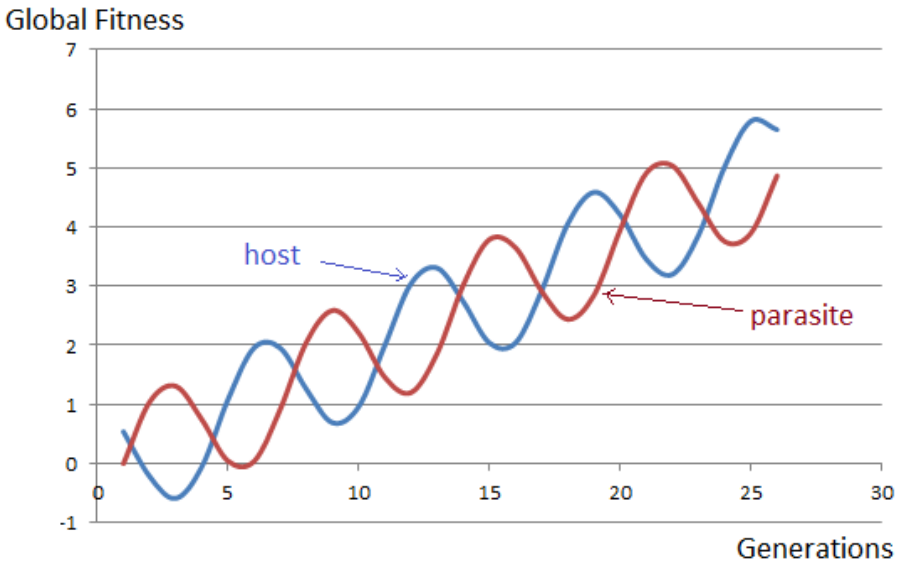


Figure 3.3: Global Fitness in a Co-evolutionary Process

3.5.5 Maintain the Diversity of the population

As it was discussed previously, Rosin & Belew [1997] proposed competitive fitness sharing function with the intention to maintain the phenotype diversity of the population. In this thesis it was proposed another method called competitive fitness sharing augment for the same intention. In the experiments performed it was observed that both methods maintain the phenotype diversity of the solutions found.

The intention of both functions is to value more diverse solutions and select them for the next generations. There is a mechanism called Hall of Fame with should keep these diverse solutions which have obtained good results. These methods are discussed in Chapter 5 and 6.

3.6 Monitoring the Progress of Co-evolution

How to monitor the progress of the co-evolution when species are evolving?, in fact, How to evaluate that populations are in arm race dynamics?. A global fitness value should increase during the co-evolution process as it is shows in the Figure 3.3.

According to Ficici [2004], if co-evolution is progressing, then individuals collected in the history memory later in time will out-perform those collected earlier in time (if the co-evolutionary algorithm is monotonic). This method is efficient

because we can stop evaluating a potential collection member as soon as it loses to some current member. This memory solution is connected to forgetting pathology found in the co-evolutionary process.

There are another methods in which individuals are added to a collection (memory) if and only if it defeats all other individuals already in the collection. This method ensures that no intransitive cycle can exist amongst the individuals in collections Ficici [2004].

Individuals observed quality is a function during interactions used to test those individuals; that is, observed quality (and ultimately fitness) is contextual. For example, in a zero-sum game, a good individual interacting with superior individuals will appear poor; on the other hand, a mediocre individual interacting with poor individuals will appear superior. Thus, if we simply monitor population fitness values (whether mean or maximum), we cannot reliably detect co-evolutionary progress Ficici [2004]. This difficulty can lead to a manifestation of Reed Queen dynamics Valen [1973].

Luke & Wiegand [2003] discusses that fitness assessment in co-evolutionary algorithms (CEA) are subjectives, because this depend on the context of the individuals and not in an external objective measures. Thus, the co-evolutionary algorithms are not clear under what conditions a CEA would be expected to optimize in the same way that a traditional evolutionary algorithm solve a static problem. Actually understanding how the CEA are progressing it is possible to know if really the co-evolution pathologies exists or have been solved.

Luke & Wiegand [2003] defines external progress measures as measures which do not affect the dynamics of a running algorithm, by contrary, internal progress measures are used directly or indirectly to affect the progress during the execution of the algorithm. Objective measures are those in which a given individual receives a measurement value irrespective of other individuals (with which interacted), and by contrary, subjective fitness measures depend on individuals which either currently exist in some population participating in the evolution (interacting), or existed at some point during the evolutionary run (interacted in previous generations).

Luke & Wiegand [2003] propose two types of co-evolution progress measures,first, the measure should give us some indication of an algorithm's performance in terms of the optimization problem we want to solve. Second, there should be some reason to believe the measure is somehow connected to the problem in terms of the real dynamics of the algorithm.

Some external objective progress measures have been proposed, as for example the one proposed by Rosin & Belew [1997], using static (and external) representative sampling set of the strategy space. This proposal it is interesting, but, at the same time is difficult to identify which set of test cases to be used for this sampling. In this thesis the approach will be measure the progress of the

co-evolutionary technique proposed will be used another strong computer player, i.e Gnugo , competing this player against the best player saved in every generation. This is a good approach because is available some players (weak and strong players) but probably in other domain that is not feasible.

Other use of having an objective measure to the co-evolutionary algorithms is to have the possibility to compare these algorithms, and identify which ones are more efficient or which ones in fact are optimizing. Using strong players as external agent as objective measure can be useful to improve algorithms and monitor the solutions or strategies that are found during the co-evolutionary learning.

Luke & Wiegand [2003] proposed a model using a single population with non-parametric and objective fitness (a ranking function) with full mixing (all individuals compete with each other in the population).

3.7 Pathologies in Coevolution

Different authors are found some pathologies observed during the co-evolutionary process Ficici & Pollack [1998]. In this section is discussed Loss of Gradient and disengagement, Cyclic Dynamics and Forgetting pathologies:

3.7.1 Loss of Gradient and Disengagement

According to Ficici [2004], the co-evolutionary learning imply two search problems, the primary search problem which concern to the domain of interest, i.e. identify a good computer player in the population, and the second search problem is the discovery of interactions that will allow for the search of the primary domain effectively and recognize solutions, i.e. the find the best testers to identify the good solutions of the primary domain.

In case of the game as Go, and other games, the first and second search problem are not different search problems, for example, if we are co-evolving two different populations to find the best computer go player, from the perspective of the host, in the host-parasite interaction, the identification of the best host population is the primary search problem, and the identification of the best parasites to test the host population is the second search problem, which in principle are other best Go players. If playing black stones, the co-evolution will progress if is tested against good White stones players.

The lost of Gradient can be defined as the following, from Ficici [2004]: Let $\mathcal{L}$ denote our current set of evolving individuals (i.e., search space locations) in our primary search problem; we need to evaluate the members of $\mathcal{L}$. Let $\mathcal{T}$ denote our current set of interactions, obtained through the secondary search effort, that we

have available for testing the members of $\mathcal{L}$. If no member of $\mathcal{T}$ can distinguish any two members of $\mathcal{L}$, then we have a loss of gradient in the primary search effort. When the primary and secondary search problems involve separate populations, then a loss of gradient means that the populations have become disengaged. Let us assume a state of gradient loss between $\mathcal{L}$ and $\mathcal{T}$; further, let us assume that there exist members of $\mathcal{L}$ that are sub-optimal. Given the state of $\mathcal{L}$, if the solution concept for the secondary search problem $\mathcal{O}_{\mathcal{T}}$ judges the members of $\mathcal{T}$ to be superior to (secondary) search space locations not in $\mathcal{T}$, then $\mathcal{O}_{\mathcal{T}}$ is either improper or has been incorrectly implemented.

One example of this disengagement is observed when two different populations are co-evolving, if one population has always superior results to the other population, this will not allow to these two populations future co-evolution. The implementation of competitive fitness sharing Rosin & Belew [1997] should maintain the diversity in the population and prevent from this pathology.

There are some works that discuss about this gradient loss and disengagement, and what should be the solutions to solve this issue, as the one described by Rosin [1997], which describe phantom parasite method, which operates in conjunction with their competitive fitness sharing mechanism in the context of zero-sum games. Assuming the case where a member α of population X defeats all members of the opposing population, then α loses by definition to the phantom parasite; if another member β in population X loses to some member of the opposing population, then β wins by definition against the phantom parasite. So, this method prevent the accumulation of the best individuals α not allowing the creation of superior population, and promoting the accumulation of easier β population and keeping the engagement between these two populations.

There are another approaches to prevent that these perfect individuals take over the population, i.e. individual with perfect score again opposing population are discount their fitness Ficici [2004].

In this thesis is proposed another technique to avoid the loss of gradient or disengagement avoiding the accumulation of superior members in the populations, this is a variation of the techniques proposed by Rosin & Belew [1997]. There are other technique proposed to introduce more diversity in the population based on replacement immigration rate introducing new members when the population are obtaining more successful results in competitions against the opponent population. When the player population win more times against the opponent, this rate is greater, when the player win less times, this rate value is lower.

3.7.2 Cyclic Dinamics

Cycling population dynamics is other co-evolutionary pathology which has been investigated by the community which is caused by intransitive superiority struc-

tures.

A relation is transitive if whenever it relates some A to some B, and that B to some C, it also relates that A to that C; and intransitive happens when that relation is not transitive. The other strong property is anti-transitivity when this relation do not happen at all. An example of transitivity can be observed in the food chain, wolves feed on deer, and deer feed on grass, but wolves do not feed on grass. Thus, the feed on relation among life forms is intransitive, in that sense. Cycling is produced because this in-transitivity behavior.

The term in-transitivity is often used when we talk about scenarios in which the relations describe the relative preferences between pairs of options, and weighing several options produces a "loop" of preference as in this example: assuming three options A, B, and C. Assume the relation is transitive. Then, since A is preferred to B and B is preferred to C, also A is preferred to C. But then, since C is preferred to A, also A is preferred to A. Therefore such a preference loop (or "cycle") is known as an in-transitivity.

There are some examples on in-transitivity as the Rock-Paper-Scissor child game, which is a symmetric zero-sum game for two players in which each player have three options play Rock, Paper or Scissor at the same time; these strategies are sorted in an intransitive cycle: Rock beats Scissors, Scissors beats Paper, and Paper beats Rock. There is an unstable Nash equilibrium in this game which is the mix of these three pure strategies with a probability one third each one.

Because this cycling behavior, the co-evolutionary process cannot succeed because the strategies are just jumping from one strategy to another, actually the same strategies. This produce an overspecialization with fragile results, and the way to solve this issue is apply broaden mechanism to the evolution (or promote diversity). As Ficici [2004] mentions that the cyclic dynamics obtained from intransitivity display a process of overspecialization, and the solution to this is a greater genetic and behavioral diversity thus maintained broadens selection pressure and dilutes the effect of in-transitivity.

That cycling manifestly indicates improper implementation of a solution concept, or in the worse case solution concept not exist Ficici [2004]. This cycling dynamics can be observed in the Red-Queen dynamics, where to maintain a level fitness in a dynamic environment, a specie must continuously evolve, but the difference is that in case of Red-Queen dynamics is the co-evolution or "arm race" is happening, which is not the case of cycling behavior.

There are some techniques proposed to solve the intransitivity issue using memory mechanisms can dampen cyclic dynamics. The Competitive Fitness Sharing is another method to enhance phenotype diversity and thereby broaden selection pressure. Using of different isolated populations during the evolution is another method to provide diversity in the solution search space.

In this thesis was used the competitive fitness sharing Rosin & Belew [1997]

and SANEi Zela & Zato [2011] to introduce an immigrant population mechanism in every generation to promote the diversity in the population of neurons. This mechanism is incorporated in the technique that is proposed in this thesis.

3.7.3 Forgetting

As Ficici [2004] described, the pathology of forgetting entails the process of trait loss. A trait can be defined as any measurable aspect of behavior (phenotype). To explain how this trait loss happen, assume that at time t the population contains some individuals with some trait x, So, this trait can be lost for the following three reasons: 1) The selection pressure act against the trait x, i.e individuals with that trait have a less fitness, on average, than individuals without the trait, 2) The selection pressure has not acted strongly over the the trait x and is left to drift according to biases in the variational operators, and 3) the trait x is selected for the next generation, but is difficult maintain it in the next generations, that is, the variational operators are strongly biased against it, making offspring likely to lack the trait. These causes of trait loss eventually lead to a population at some point later in time (after some generations) where no individual has trait x Ficici [2004].

The trait loss becomes an instance of forgetting when after some generations, we have a population where 1) no individual has trait x and 2) some individual would like to have the trait x because this can increase its fitness. Thus, in forgetting, a previously acquired and subsequently discarded trait is once again desired Ficici [2004]

As in other pathologies, the general solution to the problem of forgetting is to maintain a sufficient diversity of selection pressures. For example, forgetting can be mitigated by using multiple, reproductively isolated populations or diversity methods such as competitive fitness sharing Ficici [2004].

There are other approaches to solve this pathology, as the use of memory mechanisms that maintain a collection of good individuals discovered during the evolution; the intention of this memory that this can collect a wider range of phenotypes than are typically found during the co-evolving population at any one moment.

Design a memory mechanism to remember what should be collection of phenotypes to remember and mitigate this forgetting issue is related to definition what is the solution concept in a domain. What collection of traits constitutes the desired or correct set, and what properties does this collection have?.

The answer to these questions are related to some issues as when a domain forces mutual exclusivity between certain traits, or when an evolutionary representation or genotype, cannot simultaneously encode all desired traits Ficici [2004].

After the solution concept is defined, the next step is define what organizing principle do we use in the memory mechanism to obtain it?. The memory mechanism can be viewed as an accumulator of traits; so, when the trait enters and remains in the memory only if it is worth remembering, according to our organizing principle Ficici [2004].

The memory mechanism approach used in this thesis use of "best of generation" (BOG) model as Hall of Fame of Rosin & Belew [1997]. The BOG is described by 1) the most fit individual in each of the m most recent generations is retained by the memory mechanism and 2) l of the m retained individuals are sampled without replacement for use in testing individuals in the current generation Ficici [2004]

SANE has their own implementation of Hall of Fame and the memory collect the best networks from the m= ∞ and the l an integer value, i.e. 30, which is the number of blueprints that are going to be keep as opponents for the next generation.

3.8 Generalization and Diversity

Chong *et al.* [2008] defines the Generalization as the ability of the learning system to find the solution, which can be viewed in the context of input output mappings, that best predicts the required output for any new input that has not been seen during the training process.

According to Liskowski [2012] probably the first attempts to adopt the generalization framework and investigate the coevolutionary learning through the prism of generalization performance were carried out by Darwen & Yao [1995], Yao *et al.* [1996] which used a large number random test strategies.

As Chong [2007] mentions, co-evolutionary learning involves a training process where training samples are instances of solutions that interact strategically to guide the evolutionary process.

As it was discussed in the previous section, there is a consensus that a general approach to the pathologies discussed is the maintenance of genetic and phenotype diversity in the population. There many approaches to maintain diversity, some of them slow genetic convergence by halting evolution in a population, the use of multiple and reproductively isolated populations, various types of fitness sharing as competitive fitness sharing and sample sharing, selective method of combination, methods to achieve speciation in the population ESP Perez-Bergquist [2001] and others.

One of the issues discussed previously is the maintenance of the phenotype diversity as a solution for the second search problem, which solution solve the loss of gradient pathology, but the diversity maintenance is needed not only for

evolutionary learning processes as was stated by Epstein [1994], who argues that a knowledge-based approach to game learning provides a systematic exposure to different aspects of a game.

As mentioned before, some of the heuristic remedies to identify this teaching sets were provided by Rosin & Belew [1997] as competitive fitness sharing and phantom parasite, in this thesis is introduced a variation of competitive fitness sharing called competitive fitness sharing augmented (CFSA) and the replacement immigration rate (RIR) which still maintain the diversity of the population.

In Chong [2007] there is an approach to identify the relationship between the maintenance of diversity and how impact this to the generalization of the solutions searched. So, according to Chong, the generalization performance of co-evolutionary learning can then be obtained with respect to the evolved solutions in terms of the notion of how solutions can best predict the required output for any new input that has not been seen during the co-evolutionary learning process. So, for example in the context of game-playing, the notion of generalization can refers to how well a strategy (solution) can best predict the necessary strategic responses (output) to any new opponents play (input).

According to Chong, there are some results that compared co-evolutionary learning with and without diversity maintenance that showed that the introduction and maintenance of diversity do not necessarily lead to a significant increase in the generalization performance of co-evolutionary learning. Another results suggested that diversity maintenance that leads to speciation among individual solutions in the population during the evolutionary process can result with a positive and significant impact to the generalization performance of co-evolutionary learning if an ensemble is constructed from the speciated population Chong [2007].

The intention of the introduction of some techniques that promote the diversity is because co-evolutionary process could produce more general strategies that are able to compete and obtain successful results against opponent strategies that were not tested during the co-evolution. In this thesis is measured the generalization of the computer Go players evolved which is discussed in the chapter 5.

3.8.1 Estimated Generalization Performance

Chong *et al.* [2008] defines an Estimated Generalization Performance for a co-evolutionary learning approach. For Chong, the generalization performance of a strategy as its average performance against all test strategies. So, considering this definition, the best generalization performance for a coevolutionary learning system is the one that produces evolved strategies with the maximum average performance against all strategies Chong *et al.* [2008].

So, below is how Chong propose how to calculate the estimated generalization performance for co-evolutionary learning: In co-evolutionary learning, it is considered the performance of a solution relative to other solutions through interactions as in game-playing problem. Let's have two strategies i and j, and let $G(i,j)$ be the game outcome of strategy i playing against strategy j.

So, strategy i is said to solve the test provided by strategy j if $G_i(j) \geq G_j(i)$.

For example, It can exist a game where the strategy i wins against j if $G_i(j) > G_j(i)$, but loses otherwise. Different games will have different game outcomes (and how they are defined).

For co-evolutionary learning, let us take a co-evolved strategy i, and Let have test strategies j obtained from strategy space S. Chong defined the true generalization performance of co-evolved strategy i, as G_i in the following equation:

$$G_i = E_{P1(j)}[G_i(j)] = \int_S G_i(j)P1(j)dj \tag{3.1}$$

Where G_i is the expectation of strategy's i performance against j, $G_i(j)$, with respect to the distribution $P1(j)$ over strategy space S (i.e., the distribution with which opponent strategies j are drawn).

So, G_i can be calculated as:

$$G_i = \frac{1}{M}\sum_{j}^{M} G_i(j) \tag{3.2}$$

Which is simply its average performance against all test strategies j and M it is all possible strategies in the solution space. but, G_i could not feasible to calculate in the equation 3.2 because all the possible solutions can be a huge number. So, Chong propose to calculate an Estimated Generalization Performance using the following equation:

$$\hat{G}_i(S_N) = \frac{1}{N}\sum_{jES_N} G_i(j) \tag{3.3}$$

Where S_N is the sample of N test strategies randomly selected from S. For simplicity it is used $\hat{G}_i$ for $\hat{G}_i(S_N)$.

Now, it is wanted to know how accurate is the estimate $\hat{G}_i$ compared with the true generalization perfromance G_i, or how small is $|\hat{G}_i - G_i|$, but as G_i is unknown, it is not possible to calculate $|\hat{G}_i - G_i|$.

But, as Chong proposed, it is possible can make a statistical claim with a confidence and with the degree of accuracy that $|\hat{G}_i - G_i| \leq \epsilon$

For this, Chong use the Chebyshev's theorema which can be check it for more details in Gnedenko & Gnedenko [1998]. So, applying the Chebyshev's theorema

can be derived the next equation:

$$P(|\hat{G}_i - G_i| \geq \epsilon) \leq \frac{\sigma_i^2}{N\epsilon^2} \tag{3.4}$$

Where $D_N = \hat{G}_i - G_i$ as the random variable as $\hat{G}_i$ is obtained through S_N.

Where $\sigma^2 = Var_{P1(j)}[G_i(j)]$ and the random variable $G_i(j)$ distributed over the interval $[G_{MIN}, G_{MAX}]$ has $\sigma^2_{MAX} = \frac{(G_{MAX}-G_{MIN})^2}{4} = \frac{R^2}{4}$.

So, Chong defines the following lemma:

Lemma 1: For a strategy i, let $\hat{G}_i$ be the estimated generalization performance with respect to N random test strategies and G_i be the true generalization performance with respect to N random test strategies and G_i be the true generalization performance. Consider the absolute difference $\hat{G}_i - G_i$, which is a random variable with distribution P_N taken on a compact interval $|G_{MIN}, G_{MAX}|$ of length $R = G_{MAX} - G_{MIN}$. Then, for any positive number $\epsilon > 0$:

$$P_N(|\hat{G}_i - G_i| \geq \epsilon) \leq \frac{R^2}{4N\epsilon^2} \tag{3.5}$$

The framework proposed by Chong is independent to the complexity and distribution of strategies in the solution space of the game, and it is independent to co-evolutionary learning algorithm used. The other assumption is that the values G_{MAX} and G_{MIN} should be known a priori because belongs to the problem domain.

For simplicity we can derive the formula 3.5 to the following equation:

$$P_N(|D_N|' \geq \epsilon') \leq \frac{1}{4N\epsilon'^2} \tag{3.6}$$

or

$$P_N(|D_N|' \leq \epsilon') \geq 1 - \frac{1}{4N\epsilon'^2} \tag{3.7}$$

Where $e' = e/R$ and $|D_N|' = |D_N|/R = |\hat{G}_i - G_i|/R$ which is an absolute normalized difference of the generalization performance.

So, as Chong mentions, in practice specifying a N as large as possible and small precision values ϵ it is possible calculate the estimated generalization performance near to the true generalization performance.

So, for example considering an estimate based on a random test sample of size $N = 50000$ and accuracy $\epsilon = 0.01$, it is obtained $1 - P_N(\epsilon) = 1 - 0.05 = 0.95$. So, it is possible to claim that with 95 % confidence the absolute difference between the estimate and the true generalization performance whould not exceed 0.01.

So, using the previous framework, it can be defined the generalization performance. For zero-sum games like Go, the game outcome for a strategy can

be easily defined, e.g., win, lose, or draw. When it is applied the generalization framework with this definition of game outcome for Go, we refer to how well co-evolutionary learning can produce a strategy that can win against as many strategies as possible Chong *et al.* [2008].

Chong introduces the generalization frameworks in terms of the number of wins as in case of zero-sum games and in terms of the average pay-off in case of iterated prisoner's dilemma (IPD) game where there is not a winner, instead, the best strategy is calculated based on the total pay-off.

In this thesis is applied the generalization framework in terms of the number of wins, so, this generalization performance is defined as: If $g(i,j)$ refers directly to the average payoff per move to strategy i when it plays against an opponent with strategy j, then the game outcome, which can be either win or lose, is defined as:

$$G_W(i,j) = \begin{cases} C_{WIN}, & \text{for} \quad g(i,j) > g(j,i) \\ C_{LOSE}, & \text{for otherwise} \end{cases} \tag{3.8}$$

Where C_{WIN} and C_{LOSE} are constant where $C_{WIN} > C_{LOSE}$, so using the generalization performance estimated using a N sample of strategies is calculated:

$$\hat{G}_W(i) = \frac{1}{N} \sum_{j \epsilon S_N} G_W(i,j) \tag{3.9}$$

And using the equation 3.7, we can conclude that to calculate the generalization performance in equation 3.9 getting a confidence of 95% near to true generalization performance it is needed 50,000 test cases (or strategies) taken from the space S of all the test strategies, i.e. all the strategies of Go game. But in the results of the experiments provided in Chong *et al.* [2008] for the IPD game it was needed some small sample of strategies.

In Chong *et al.* [2008] is provided some algorithms how to select the biased test sample from the solution space which was called partial enumerative search, which is basically selecting randomly strategies in every iteration, which will have to compete in a round-robin competition and selecting the best one of that competitions, and keeping the best solution of every iterations.

The interesting contribution of this framework is that can be used to compare different co-evolutionary algorithms in terms of generalizations, and identify in which generations of the evolution the solutions are less general than in other generations as it was show for the IPD game.

But in Chong *et al.* [2012], Chong provides some examples of statistical robust estimation of generalization performance in co-evolutionary learning using less number of test samples to the distribution-free framework using Chebyshev's. According to Chong *et al.* [2012] for the IPD game it was needed 2000 sample

test cases, in case of the Othello, and it was needed 5000 test samples to test accuracy the generalization performance.

In sample theory Azorin & Sanchez-Crespo [1986] there is mechanism that can be useful to calculate what should be sample size that should be used and be valid with certain level of confidence the number of test cases to test the strategies evolved.

$$n = \frac{z^2_{\alpha/2} pqN}{E^2 N + z^2_{\alpha/2} pq} \tag{3.10}$$

Where n is the sample size calculated based on the $z^2_{\alpha/2}$ which is value related to the confidence level, which in case of 95% of confidence level this value is 1,96. p is the proportion parameter, which in worse case is 50%, considering the worse case in which is not possible to know the variability of the population, this value selected should be 50%, and $q = 1 - p$. E is the level of accuracy and N is the size of the population.

Considering the N a big number because of the big number of strategies that can be found in Go game and using this equation 3.10, it is possible to calculate what should be the size of sample test cases which can be used to measure the generalization of the solutions learned. In this thesis it was created 180,000 players randomly, from which are selected the best players 1800 sample players after the competition of all of these players. So, we can claim that with a confidence level of 95% and accuracy of 0.023 the number of sample players is around 1800 players.

In Chong *et al.* [2012] large fluctuations of using traditional co-evolutionary learning (CCL), instead using the improved co-evolutionary learning (ICL) which use the generalization performance estimate as fitness measure $\hat{G}_i$ has not observed major fluctuations. This fluctuation happens because the overspecialization of the population to some specific strategies. Chong performed some experiments including both calculations, the traditional relative fitness of the CCL and the estimated generalization performance as fitness in the three-choices IPD game, using just mutation as genetic operator. The results were that what is really contributing to the generalization performance of the strategies created are the estimated generalization performance used as fitness and not the traditional fitness used CCL.

In Chong *et al.* [2009] is discussed some diversity techniques and the relationship between the maintenance of the generalization of the strategies during a co-evolutionary process and the diversity techniques. For Chong *et al.* [2009], the maintenance diversity techniques can be divided in implicit techniques, which emphasize the diversity maintenance through the selection process, and explicit techniques that emphasize the diversity maintenance through the variation pro-

cess.

Some implicit diversity techniques are speciation or niching, in which the fitness of the solutions are reduced to lower values if these solutions are more similar, one example of this technique is competitive fitness sharing Rosin & Belew [1997] and implicit fitness sharing proposed in Darwen & Yao [1997] where the shared fitness for each strategy i is calculated based on the competition against σ opponents randomly selected, and assigning to the best opponent (which obtained the best payoff) against i the high fitness, but other opponents will receive a less shared fitness (the average of n opponents that tie against i).

The other one techniques are Pareto co-evolution de Jong & Pollack [2004] which use the concept of Pareto Dominance and multi-objective optimization (every opponent is an objective) to select the strategies that are Pareto non-dominating (i.e. comparing strategies i, j) for the next generations and Reducing Virulence technique Cartlidge & Bullock [2004] which is inspired from the host-parasite relationship and which is observed when there is a lot of parasite that can kill hosts and which at the same time reduce the population of parasites (disengagement pathology). So, one way to reduce the virulence of the parasites is to lower the fitness (rescaling based on ν parameter) of the top performing strategies in the co-evolving population that beat the largest number of opponents.

The explicit techniques that maintain the diversity are the variation operators used after the evaluation of the members of the population, the most used operators are combination (crossover) and mutation. If it is needed to increase the diversity of the population, the values of these operators can be increased accordingly.

In Chong *et al.* [2009] is discussed some diversity measurements based on phenotypic and genotypic. The genotypic diversity relates to the level of variety in the genotypic space (the search space that is specified by the solution representation) and phenotypic diversity relates to the level of variety in the phenotypic space, for example, fitness values used in the selection process of co-evolutionary learning

The genotype diversity measurement can be calculated using the edit distance Ekart & Nemeth [2002],Burke *et al.* [2004]. In a co-evolving population can be calculated by taking the average value of pairwise distances of strategies in the population:

$$\frac{2}{\text{POPSIZE}(\text{POPSIZE}-1)} \sum_{1 \le i < j \le \text{POPSIZE}} \text{dist}_{\text{ed}}(i,j) \tag{3.11}$$

Where POPSIZE is the population size and the edit distance can be calculated

in the following equation:

$$dist_{ed}(i,j) = \frac{1}{n_e}\sum_{k=1}^{n_e} d(i_k, j_k s) \tag{3.12}$$

Where i_k, j_k are the elements of the strategies i, j ,and $d(i_k, j_k)$ is the distance metric on i_k and j_k, and n_e is the total number of elements of the direct lookup table used (for 4-choice IPD game $n_e = 17$). For the distance metric was used the following equation:

$$d(i_k, j_k) = \begin{cases} 0, & i_k = j_k \\ 1, & i_k \neq j_k \end{cases} \tag{3.13}$$

Burke *et al.* [2004] applied the diversity measurement in genetic programming, in this, genotype diversity counts the numbers of unique trees, not considering the fitness neither the behavior of the tree. Two trees are equally if they contain the exact the same structure. Phenotype diversity consider the number of unique fitness values of the population.

Using the n-choice IPD game in Chong *et al.* [2009] was compared which diversity maintenance technique introduce more diversity in the populations and improve the generalization performance during the co-evolution. According to the results of that experiments, there are some techniques as competitive fitness sharing which introduce more diversity and has a positive impact in the generalization performance, comparing with other techniques as reduced virulance, implicit fitness sharing and pareto evolution.

Sallam *et al.* [2008] proposed some measures for the genotype and phenotype diversite. In Sallam *et al.* [2008] is reviewed the Hamming distance and the Edit distances. As it is indicated, as per the result of evolution, chromosomes have different lengths, the same genes may have different locations in different chromosomes, and some genes may be enabled or disabled, which make difficult to measure the genotype diversity in these neuron populations. This work propose an improving the Edit distance. As it was discussed that diversity can be defined as the amount of variety in a population and measuring diversity is an attempt to quantify the variety in that population.

In information theory the hamming distance between two strings of equal length (same number of characters) is the number of positions for which the corresponding symbols (or character) are different. Edit distance between two strings of characters is the minimum number of operations required to transform one string into the other, i.e. using some genetic operations as crossover and mutation (removing and adding new genes to change the network structure). Sallam *et al.* [2008] called Neuro-edit which is specific measure for diversity of evolved neural networks since, it based on the semantics of neuroevolution operations, crossover and mutation. Neuro-edit measures the distance between neural

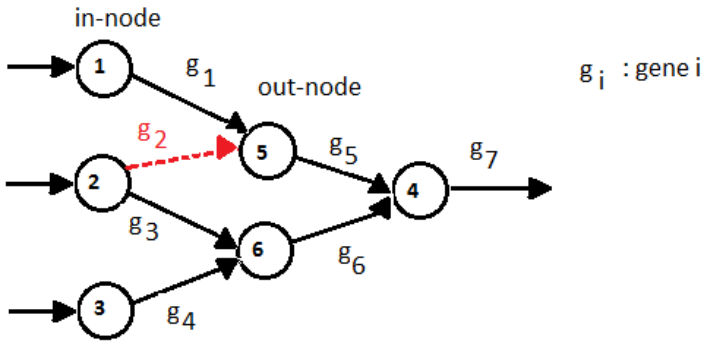


Figure 3.4: NEAT architecture and Edit distance

network structures in terms of connection genes "addition, removing, and substitution".

Sallam *et al.* [2008] propose two edit distances, called neuro-edit, to measure the diversity of two neural network based in the similiraty of the genes and weigths in the network. The first edit distance which is called common genes distance happens when two same genes (using the NEAT structure Stanley & Miikkulainen [2002b], genes are the connection between two nodes) appears in two neural networks, in these cases the distance is measure base in the normalized difference of the weights of that connections. This can be observed in the equation:

$$d_{common} = \frac{1}{n}\sum_{i=1}^{n}\frac{[st(g_i)_{C_1} * |w(g_i)_{C_1}| - st(g_j)_{C_2} * |w(g_j)_{C_2}|]}{max[st(g_i)_{C_1} * |w(g_i)_{C_1}|, st(g_j)_{C_2} * |w(g_j)_{C_2}|]} \tag{3.14}$$

Where n is the number of the common genes, $st(g_i)_{C_1}$ and $w(g_i)_{C_1}$ are the state and weight of a common connection gene $g_i \in C_1$, and $st(g_j)_{C_2}$ and $w(g_j)_{C_2}$ are the state and weight of a common connection gene $g_j \in C_2$. In case that there are genes with status disable (in the NEAT architecture), the distances is 1. In the NEAT structure, the connections has enabled or disable status to represent if that connection exist. This can be observed in the figure

In the Figure 3.4 the gene g_2 which connect the node 2 (in) and node 5 (out) is disable and the other genes in the chromosome are enable.

The second edit distance, which is called uncommon genes distances, is measure when two genes are different or the connection does not exist between the nodes (not even with status disable), in other words, when the genes are different

in the compared chromosomes. . This can be observed in the equation:

$$d_{uncom} = \frac{1}{n}\sum_{i=1}^{n} st(g_i)_{c_1} + \frac{1}{m}\sum_{j=1}^{m} st(g_j)_{c_2} \tag{3.15}$$

Where $st(g_i)_{c_1}$ is the state of the uncommon connection gene $i \in C_1$, and $st(g_j)_{c_2}$ is the state of the uncommon connection gene $j \in C_2$. As it can be observed the distance for uncommon genes depend on the status (1 in case gene is enable and 0 in case is disable). So, the total distance of chromosomes C_1 and C_2 is calculated :

$$d_{NE}(C_1, C_2) = \frac{1}{3}(d_{com} + d_{uncom}) \tag{3.16}$$

Where $0 \leq d_{NE}(C_1, C_2) \leq 1$. So, Finally the genotype diversity of the population is calculated with the following equation:

$$div = \frac{1}{\frac{n(n-1)}{2}}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{NE}(i,j) \tag{3.17}$$

Where $d_{NE}(i,j)$ is the neuro-edit distance between individuals i and j.

3.9 Solution Concepts in Co-evolution

A solution concept is a formalism for predicting how a game will be played Ficici [2004]. These predictions are called solutions, and describe the incentive structures of the players which interact strategically, or recommendations for the players of which strategy to play, and, therefore, what will be the result of the game. This definition is taking from Game Theory described previously.

Probably the leading solution concept for non-cooperative games is Nash equilibrium. In the Game Theory, the solution concept players are assumed to be rational and so strictly dominated strategies are eliminated from the set of strategies that might feasibly be played.

In the adaptation of the Game Theory to the Evolutionary Game Theory (EGT), one of the main assumptions is that the players are not necesarily rational, and it is only required to have an strategy. The result of the game will test how good or bad the strategy played.

In EGT the success of an strategy is not because how good was the strategy, it also depends how good was the strategy in the presence of the other alternatives strategies, and the frequency of the other strategies were used in the competition.

For the search of solutions to a problem in a domain should be defined a solution concept which can identify what is the solution and what is not the

solution to that problem, so, the solution concept partitions the search space into two classes: solutions and not solutions detectable by some attributes that different these solutions. Sometimes this solution concept is not explicitly defined, but in fact, this exist.

According to Ficici that the fundamental problem to the pathologies are the lack of solution concept for co-evolutionary algorithm. There are many solution concepts, depeding of the game, some of the them are example Nash equilibrium, Pareto Optimality, BITE (Best In The Ecology) and BSS (Best Score Strategy) proposed by Ficici [2005] and others.

In Ficici [2005], the solution concept BITE is defined for a conventional single-population co-evolutionary algorithm. In this scenario, the solution can be embodied by a single individual, to be (the strategy implemented by the phenotype of) the individual in the final population with the highest fitness.

In this single population scenario the fitness of the individuals are obtained from the sum of payoffs obtained by the interaction of the individuals with all other members of the population (even interaction with itself). In this scenario the fitness is frequency-dependent, it means, that the fitness is sensitive not only to which strategies are represented in the population, but also the proportions with which each strategy is represented.

The other solution concept proposed by Ficici [2005] is BSS, in which given the set S of unique strategies, this solution concept returns the strategy $\hat{s}$ in S that obtains the highest average score from interaction with each member of S. The main difference between BSS and BITE is that BSS assumes that the set S not to contain duplicates; so that the choice of strategy $\hat{s}$ does not depend upon the frequency with which different strategies might appear in the population.

So, the Solution concepts form the nexus of search problems and processes to solve them. Failures to obtain the desire solutions can indicate a wrong implementation of the solution concept.

As Ficici [2004] mentions, any real-world search problems are usually difficult enough that we must be content with satisficing rather than optimization, and any satisficing problem can be formally stated as an optimization problem, where a solution is any result that it deems good enough.

The solution concepts are intrinsic to search problems and implemented by search algorithms. All of these algorithms must implement the same solution concept for the same search problem to be consistent with it, if not they will solve another search problem.

For the efficiency of the search space is needed a metric for goodness to assess locations with even finer grain.

There are domains where the solutions is obtained through the interactions of the agents against other agents, and the properties that are needed to identify the solutions are only revealed through these interactions. This imply the definition of

the solution concept to the secondary search problem, which will identify a priori what are the interactions that promote effective search to identify the solution in the domain, the primary search problem.

According to Ficici [2004], a conventional co-evolutionary algorithm believes that a perfect strategy for a game (i.e. chess or Go) is also the perfect metric for an opponents ability. For example, Kasparov will beat both a beginner as well as an intermediate player, but this not provide an indication that intermediate player is better than the beginner players.

As per Ficici [2004], the solution concept does not operate directly upon the problem domain, but rather upon the intermediary measurement function, which additionally specifies metrics of behavior, therefore, we will say that solution concepts solve measurement functions, rather than domains.

3.9.1 Formal Definition of Solution Concept

Ficici [2004] defines the solution concept in the following way which will be used for this thesis. There is a measurement $M(i, E, v)$ which represents the success of a particular behavior, acting in a particular context, with respect to a single metric (or dimension) of behavior; where i is a role in event E according to a particular objective metric v.

The metric v is obvious in some domains as board games, i.e. scores of the players, but difficult to identify in other domains as the robot-locomotion domain Sims [1994], in which is wanted to discover an effective combination of robot morphology and control. The measure that can be choice are the average rate of movement, distance traveled from the starting point, and the surface area of the robot that comes in contact with the ground; The intention of this is because we can prefer robots that travel fast, but not in circles, and do not drag themselves on the ground (to encourage the discovery of limbs, perhaps). Thus, in this domain we can have three different units of measurement in this domain: Rate, distance, and area Ficici [2004].

The solution concept operate in this intermediary measurement function M, which integrate events and dimensions of success to obtain an assessment of behavior success.

A behavior complex (or collection of behaviors) X_i is a subset of the corresponding set of behaviors B_i that are made available to domain role i, where role i contribute to the solution in the domain. A configuration K is an n-tuple of behavior complexes X_i, one for each role i in the domain.

A solution is a configuration K that exhibits some set of properties defined by the solution concept O. And, a solution set is the set of all possible solutions with respect to the given measurement table T and solution concept O, which it is denoted by $S(T, O)$. Where T is a measurement table which contains all

the measurements obtained to apply the function M, or all the outcomes of the interactions between these behaviors.

The previous definition is useful to define a solution concept O either extensionally or intensionally. Extensionally is when a solution concept can simply specify which configurations K belong to the solution set and which do not, without stating any underlying properties of a configuration to be considered a solution. Intentionally, is when a solution concept can state a number of properties that a configuration must possess to be a member of the solution set.

When the solution set contains more than one solution, it is needed an formal preference definition of one solution over other solutions. The application of this preference is called solution concept refinement. For Example, in the IPD there are two Nash equilibrium, both cooperate or both defeat are the solutions to the problem, thus, some solution concept can have different properties, as Pareto dominance or risk dominance in some Nash equilibrium solution concepts, so based on these properties can be formalize the preference of one solution over another.

Finally, as was discussed previously, the identification of solutions to problems using co-evolutionary approach requires to solve two distinct search problems: the first search problem is recognize solutions in the domain, and second, discover appropriate interactions to identify these solutions, and for this second search problem is needed a second solution concept, to identify these sect of interactions.

In this thesis, to identify the set of interaction, or the second solution concept, it is applied some techniques of Fitness Sharing described by Rosin & Belew [1997] that will be discussed in the next section.

3.10 Some Fitness measures in Competitive Co-evolution

In this section is described some fitness sharing proposed by Rosin & Belew [1997] in a context of host-parasite interaction.

3.10.1 Competitive Fitness Sharing

Before define the competitive fitness sharing, we can define a Simple Fitness as the sum of scores obtained by the host across all the competitions against other parasites.

Rosin & Belew [1997] define the competitive fitness sharing assigned to a host p defeating parasites with the set of indexes X as:

$$CSF_p = \Sigma_{j \in X} \frac{1}{N_j} \tag{3.18}$$

Where N_j is the total number of hosts in the population defeating parasite j. Fitnesses are most comparable when all hosts compete against the same set of parasites.

As Rosin & Belew [1997] mention, the effect of this method is to reward hosts that defeat parasites few others can, even though the rewarded host might not defeat as many parasites as others can. According to them, this is important when there do not yet exist hosts can defeat all of the parasites, and that cases, any host that defeats parasites that others cannot may well contain important genetic material.

Thus, the main benefit of using competitive fitness sharing is to keep in the population some rare hosts with good strategies. For example, suppose that host H beat parasites P than other hosts are not able to beat, and this host H are in few quantities in the host population (few individuals with this strategies), so, the fitness sharing method will be promote this host H (through) reproduction to the next generation even was able to beat just the parasite P in comparison with other hosts that are able to beat more parasites (weak parasites).

3.10.2 Pareto Co-evolution

Pareto co-evolution is a co-evolutionary learning technique with implicit diversity maintenance using the multi-objective optimization framework proposed by Noble & Watson [2001], de Jong & Pollack [2004]. In the context of games, it has been claimed that strategy's fitness based on a simple weighted sum of values given by game outcomes can lead to wrong results. This technique has been applied as well using neural networks a population representation in a game called Pong Monroy *et al.* [2006].

It is not possible to know at the time of fitness evaluation are made a priori whether all outcomes are of equal importance (uniform weighting) or that some outcomes are more important than others (nonuniform weightings). This technique use explicitly the opponent strategies (test cases) to represent different objectives of the problem that a particular strategy must address Chong *et al.* [2009].

The selection process in Pareto co-evolution is based on Pareto-dominance relationship. In a single population Pareto co-evolution, to determine whether strategy i Pareto dominates strategy j (learners), considering all $k \neq i, j$ strategies from the population as the N objectives. The objective value $O(i, k)$ is the value

of the outcome of the game of strategy i against the strategy k. Strategy i Pareto dominates j if i has at least the same game outcome as j for every on the N objectives used in the competition, and there are at least one objective for which i has a better game outcome than j de Jong & Pollack [2004] :

$$dom(i,j) \Leftrightarrow \forall k : O(i,k) \geq O(j,k) \wedge \exists k : O(i,k) > O(j,k) \tag{3.19}$$

Where $1 \leq k \leq N$. So, strategies i and j are mutually nondominating if i is better than j on at least one objective and j is better than i on at least another objective, with both objectives being equivalent in the remaining objectives.

And the Pareto front of a set of strategies i or j (the learners), is a subset of all non-dominated learners strategies M for which :

$$m, i \in M, \nexists m : O(m,k) > O(i,k) \tag{3.20}$$

The selection process in Pareto co-evolution is based on the ranking of strategies according to Pareto layers that are obtained from the Pareto-dominance relationship Chong *et al.* [2009]. So, according to this, the mutually nondominating strategies in the Pareto front would be the most preferable strategies for selection, and the second Pareto layer would contain mutually nondominating strategies among the subpopulation that is obtained after removing the Pareto front from the full population, and the next Pareto layers can be obtained with respect to subpopulations that not include the strategies in higher Pareto layer Ficici & Pollack [2001].

3.10.3 Shared Sampling

Sometimes, because some resource computational constrains, it is not possible to use all the parasites of the population to test the host, so, for that is used a sample set of parasites which is taken randomly.

The other way to select a set of parasites (or test cases) from the previous generation, preferentially select parasites with best fitness values, but this method can select parasites from the same niche, So, as Rosin & Belew [1997] mentions, it is preferable to select the parasites from the previous generations that challenge all the segments of the hosts, so, the way to do this is to use competitive fitness sharing for sampling as well for selection, sampling individuals with highest competitive fitness sharing, which is called Shared Sampling.

The technique is described in below:

Initialize current sample to be the empty set;
for *each opponent* i *from previous generation* **do**
  beat[i] = 0 (# in the current sample beating i) ;
end
while *the current sample is not yet full* **do**
  for *each parasite* j *not yet sample* **do**
    samp_fit[j] = 0 (fitness within sample) ;
    for *each opponent* k *beat by* j *last gen* **do**
      samp_fit[j]+ = $\frac{1}{1+\text{beat[k]}}$
    end
    ;
  end
  Let j be such that samp_fit[j] is maximal;
  Add individual j to current sample ;
  for *each opponent* i *from previous generation* **do**
    if j *beat opponent* i *last gen* **then**
      increment beat[i]
    end
    ;
  end
end

Algorithm 1: Shared Sampling algorithm

This method selects a set of strong individuals from the previous generation which in theory can provide complete set of tests cases or parasites.

Shared Sampling provide some effects of the competitive fitness sharing method basically because of it is using a sample fitness function which is similar to the competitive fitness sharing function (inverse of number of times that the opponent lost). Thus, in addition to choosing parasites that challenge all hosts, shared sampling will tend to chose more representatives of parasite types that beat a large number of hosts Rosin & Belew [1997]. And as in competitive fitness sharing, parasite types that defeated a large segment of the host population will generally be represented more in the sample than those parasites that defeated a few members of the host population.

3.10.4 Hall of Fame

Hall of Fame method has the propose by Rosin & Belew [1997] to save the best individuals of the previous generations and use them as test cases for the next generation. Thus, the host (or parasite) are testing against the best parasites

(host) of the previous generation or Hall of Fame, together with the current parasites (hosts) of this generation which can be selected randomly.

So, new hosts (or new innovations of hosts) have to be successful against the hall of fame of parasites to be maintained the in the host population. The same for the new innovations of parasites.

In this thesis is implemented the hall of fame for host and parasites, thus, a set of best host (or parasites) are maintained for the next generation, in which the best parasites and the best host compete again, including new sets of parasites and hosts which are selected randomly from their populations. This set of best individuals are maintained in the hall of fame is these continue outperforming against their opponent, if not, they have big possibilities to be removed from the hall of fame.

There are some computer resource issues to implement this method using a big the number of best individuals to be maintained in the hall of fame. In the experiments were used some servers with 8 processors which facilitated the experiments, but it was not good enough, the intention is to use later a big computer called Magerit Magerit to keep a reasonable number of best individuals in every generation and increase the number of competitions per generation.

3.10.5 Phantom Parasite

The phantom parasite method is introduced to prevent disengagement between two populations in an adversarial relationship as parasite-host. As Rosin [1997] described, when there is an unbeatable parasite in the population, this parasite can be spread through its population until the losing host population can not defeat any parasite.

So, the proposed method is has the objective to retain pedagogical individuals when an unbeatable individuals exist to maintain the co-evolutionary process. A phantom parasite is added to each sample of current parasites, and it has not explicit genotype and not actual competitions are run between it and the hosts, instead is used in conjunction with competitive fitness sharing to provide a niche for pedagogical parasites. Each generation after competition outcomes against current parasites, the outcome against the phantom parasite is explicitly defined as follows Rosin [1997]:

1. Hosts that lose to some current parasite defeat the phantom parasite
2. Hosts that defeat all current parasites lose to the phantom parasite.

The phantom parasite is present only when the host are perfect again all parasites, and in this situations only beatable individuals that can be pedagogical individuals receive fitness from this phantom parasite.

Thus, when some hosts defeat all parasites, the weaker hosts defeat a phantom parasite that the strong hosts can not do it. In this way, a niche is developed for the weaker hosts that, when combined with competitive fitness sharing, gives the weaker hosts a fitness advantage when they are less represented in the population.

According to Ficici [2004] this approach distort the fitness value and therefore the equilibrium of the game. According to Rosin [1997] when competitive fitness sharing is used both sides are usually able to retain an adequate diversity to defeat any current opposition.

3.10.6 Other Fitness Sampling

There are other fitness sampling that will be described bellow:

- Full competition: Each solution competes against all other solution. This sampling is computationally very demanding, since every possible pair is evaluated, but at the same time all the solutions are evaluated against each other providing a better information about the solutions and a more accurate competitive fitness is calculated.

- Random sampling: This sampling allows each solution compete against a randomly selected number of solutions. The concern to use this sampling method is that can be excluded favorable solutions in the calculation of the competitive fitness.

- Bipartite sampling: In this sampling method solutions are divided into two teams, where each team compete against each other. Individuals are tested against one of the teams resulting in less competitions.

- Tournament sampling: This method use relative fitness measures to select the best opponents. One method is creating subsets of solutions and find the best solution for each subset using a relative fitness. More subsets are then created using the best solutions and the process is repeated until a final subset of solutions is created which will form the sample solution Angeline & Pollack [1993].

Chapter 4

Co-evolutionary Techniques Applied in Complex Problems

As Chong mentions, co-evolutionary learning is proposed as an alternative search method for difficult, real-world problems which traditional approaches such as optimization-based search algorithms cannot solve as it is very difficult or impossible to construct the absolute quality measure of solutions that is required for these search algorithms Chong [2007].

In this chapter is going to discussed two works selected by the author from other authors which works using co-evolution as learning process as method to solve complex and real problems. The first work is about the simulation of co-evolution of predators and preys, which is a case of pursuit and evasion problem. The second work is the application of the co-evolution to discover complex strategies for the trading of some securities in the capital markets.

4.1 Competitive and Cooperative Co-evolution in Prey-Predator Domain

In this work Rawal *et al.* [2010] two experiments were performed, in the first a team of predators is co-evolved with a single prey, in the second experiment using the results of the first experiment were co-evolved a team of predators with a team of prey, using in both cases cooperative and competitive co-evolution methods were using to evolve complex pursuit and evasion strategies.

4.1.1 Co-evolution of a team of predators with one prey

In this experiment a team of predators try to capture a prey in a simulated environment as can be observed in the Figure 4.1. According to Rawal *et al.*

[2010] the predator-prey domain is openended which requires continuous discovery of good behaviors by predators and preys. In this kind of simulations the agents should be able to adapt to this changing environment because the outcome of any single action of these agent is typically not known.

In the Figure 4.1.a can be observed a Multi-Agent ESP architecture, three neural networks were evolved in parallel to control three predators for the prey-capture task. These predators had to learn to cooperate to capture a single non-evolving fixed behavior prey that none of them could catch on their own. In this figure can be observed that the each population prey has their own population, which have sub-populations from where is taking the neurons to build the network of the predator. These agents move only one step at a time, and all take a step simultaneously. If the predators are in the same position as the prey is said that the predator captured the prey.

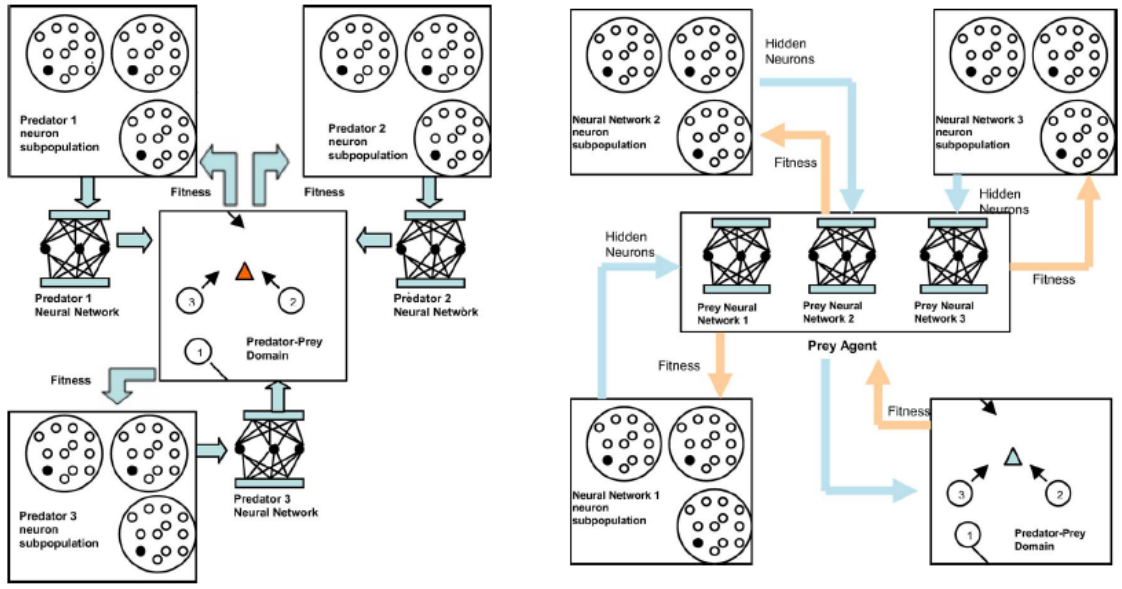
In the first experiments where co-evolved three predators with a single prey which as is shown in the Figure 4.1.a and Figure 4.1.b. In the Figure 4.1.b can be observed that the prey has three neural networks for each predator.

In the experiments performed for each neural network was used a single layer of 10 hidden neurons and sigmoidal activation functions. Each subpopulation consists of 100 neurons and each neuron, or a chromosome, is a concatenation of real-valued numbers representing full input and output connections of one hidden unit. Preys and predators start at random locations each time so that neither of them has an advantage over the other.

Once calculated the fitness of predator and preys, these fitness are assigned to the neurons that participate in the networks uniformly. Once the neurons are ranked because these fitness, the top best neurons in each subpopulation are combined using one-point crossover and replacing the worse performed neurons, and then mutated with a probability of 0.4 choosing new weights for the chromosome.

The inputs of each predator are x, y offset distances to all the prey, and similarly, the inputs of each prey are the x, y offset distances of all the predators from that prey. The output of networks are the different actions that preys or predators can take. Each prey has only four possible output actions in each time step: move east, west, north, or south, and the predators have five: move east, west, north, south, or idle.

The following equations has been used to calculate the fitness of the predators and preys. In case of the predator is more value when the predators capture both preys together instead of individually. In case of prey the fitness function values more is no prey are captured, a less if one of them are captured at the end of the simulation. The best predator and prey teams are saved in a hall of fame, and



(a) Each predator-prey agent is controlled by a single neural network consisting of several hidden neurons

(b) A single predator consist of multiple networks which are dedicated to the opposite team members.

Figure 4.1: Multiagent ESP architecture for the predator-prey domain where circle are the predators and the prey is the triangle. (a) predator with a single network and (b) predators with multiple networks.

used for crossover for the next generation.

$$Z_{predator} = \begin{cases} 25, & \text{if both prey caught} \\ \frac{20*\gamma_m}{n} + 2\gamma_m + \frac{20*d}{n}, & \text{for otherwise} \end{cases} \quad (4.1)$$

Where γ_m is the number of prey caught, n is the total number of prey, and d is the normalized sum of distances from the predator to each of the prey at the end of the simulation.

$$Z_{prey} = \begin{cases} 25, & \text{if neither prey caught} \\ 12.5, & \text{if one prey caught} \\ \frac{12.5*\rho}{R}, & \text{if both prey caught} \end{cases} \quad (4.2)$$

Where ρ is the number of time steps for which at least one prey remained alive, and R is the maximum possible number of time steps.

Different experiments were co-evolved using these configurations, initially a team of three predators and one prey, and tree predators and two preys. In the simulation of three predators and one prey it was identified that the prey needed three neural networks, one per predator to track each predator and ensure that a successful co-evolution is performed.

4.1.2 Co-evolution of a team of predators with a team of preys

The main experiment was the co-evolution of three predators with two preys to investigate the cooperative and competitive co-evolutions and the complex strategies that can evolve. These co-evolutions has different layers, at the low level there is a cooperation between neurons of each network which belongs to the predators or preys. In the next level there is another cooperation between the agents of the same species (predators and preys), and the top level is the competitive co-evolution between the predators and preys. So, there is cooperative co-evolution between the predators as they learn to work as a team to surround the prey and capture them, and the prey also cooperate as a team and their goal is to evade the predators.

So, in these experiments every predator has two networks one for each prey, and the prey has three networks, one for each predator. In these simulations the predators are aware of prey positions and the prey are aware of predator positions. However, there is no direct communication within a prey or predator team. The prey and predators move at the same speed, so, to capture the preys the predator has to evolve more complex strategies than just following the prey. The simulations have a time limit to ensure that the predators do not keep moving at random and capture the prey by accident. Instead, they need to surround the

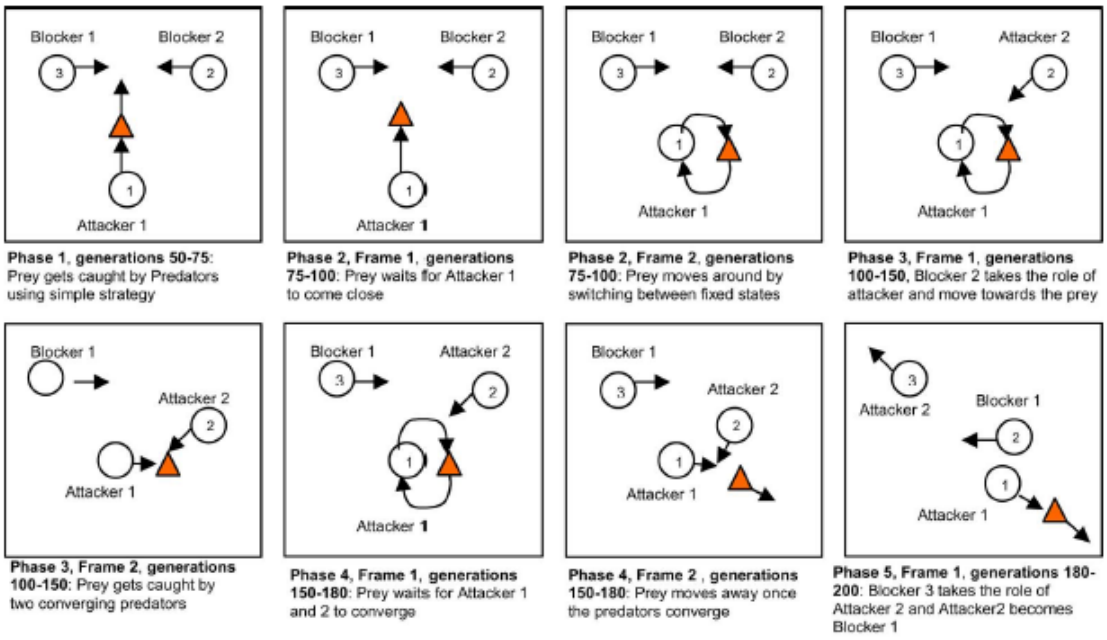


Figure 4.2: Some strategies evolved during the experiments with three predators and one prey

prey from different sides so that they do not have anywhere to escape before they can catch them.

The predator assumes two roles, as attacker or blocker, in the predator assume this role, the predator chase to the prey till capture it, in case assume the role of blocker it assume some positions to obstruct the prey. The following section will describe the strategies learned by predators and preys.

4.1.3 Results of the experiments

Using the configuration described above, in the experiments with three predators and one prey it was observed an arm race. In some generations the prey discovering good strategies successful scape from the predators, in the next generations predators playing different roles, attackers or blockers, and discovering more complex strategies were more successful to capture the prey, and that cycles happened during the different generations. Some of these strategies described in Rawal *et al.* [2010] can be observed in the Figure 4.2.

In the experiments with three predators and two preys it was more value if the predators capture at the same time the preys instead of capturing them one by one. As in the previous scenario different complex strategies were evolved during

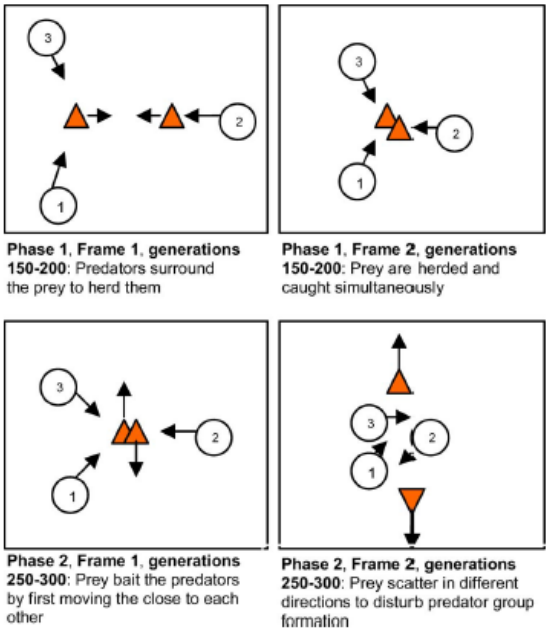


Figure 4.3: Some strategies evolved during the experiments with three predators and two preys

these experiments. i.e in some generations the predator learn how to capture the preys together, but in the next generations the preys learned strategies to escape.i.e. when the predators were near to the preys, these moved in different directions to escape. According to these authors, these behaviors co-evolve in cycles resulting in complex final behaviors for predators and preys. Some of these behaviors can be observed in the Figure 4.3.

According to Rawal *et al.* [2010] these experiments demonstrated that it is possible to sustain co-evolution of teams of competing and cooperating agents. Based on the initial results, it was discovered that using complex structures as it was described previously it is possible maintain the arms race between different agents. So, it was needed a neural network for each agent in the simulations to track their moves, two networks for the predators, and three networks for the preys. The authors believe that is more easier co-evolve component that cooperate to form a solution, rather than evolve the complete solution directly. The second conclusion at Rawal *et al.* [2010] is that hierarchy of cooperation and competition similar to that in nature was observed to emerge, including various high-level competitive and cooperative strategies in both predators and preys. The authors observed that various predators learned to change the roles dynamically based on stigmergy and learn to herd the preys before to capture

them, and the preys learned other complex strategies to avoid be captured as baiting, scattering, direction reversal and sidestepping.

4.2 Particle Swarm Optimization (PSO) Co-evolution Applied to Security Trading

The intention of include this work "Competitive Co-evolution of Trend Reversal Indicators Using Particle Swarm Optimization" by Papacostantis [2009] is to give another view of using co-evolutionary approach not using GA techniques, instead of that, in this section is going to be presented a co-evolutionary approach using another algorithm called Particle Swarm Optimizations (PSO). The Particle Swarm Optimization (PSO) was first described by Kennedy & Eberhart [1995], and from then, there have been investigate improving its capabilities and applying to different areas as games in Messerschmidt & Engelbreacht [2002], Franken [2004] and some complex problems as in Kennedy & Eberhart [2001], Papacostantis [2009], Sivanandam & Deepa [2008].

This work is presenting an application of PSO co-evolution to finance industry specifically in the trading security area.

4.2.1 Particle Swarm Optimization (PSO)

PSO technique is inspired in the social behavior of flock of birds or fish schooling, where the particles in the swarm can fly in a multi-dimensional search space in a methodical and organized manner, where each particle in the swarm represents a potential solution to the problem domain Kennedy & Eberhart [1995]. In similarity with GA, in PSO swarm is the population of solutions and a particle is a member of that population, but by contrary, the traditional genetic operators as selection, crossover and mutations are not applied to PSO.

The n-dimensional position of each particle in the swarm is determined by itself and by its neighborhoods as it described in the following equation:

$$\vec{X}_i(t) = \vec{X}_i(t-1) + \vec{V}_i(t) \tag{4.3}$$

Where $\vec{X}_i(t)$ is the vector position of the particle i at time t and $\vec{V}_i(t)$ is the velocity of the particle i at time t. The velocity $\vec{V}_i(t)$ is the velocity and direction of a particle that drives the optimization process in a n-dimensional search space. Based on that velocity, the particle moves to the personal best position $\vec{Y}_i$, and neighborhood best position $\vec{Z}_i$. Apart of that, the new position is influenced by two additional coefficients, which gives more influence to the personal best or neighborhood best positions, called cognitive factor and social factor respectively.

The following equation is used to update the velocity of the particle i at time t:

$$\vec{V}_i(t) = w\vec{V}_i(t-1) + \vec{\rho}_1(\vec{Y}_i - \vec{X}_i(t)) + \vec{\rho}_2(\vec{Z}_i - \vec{X}_i(t)) \tag{4.4}$$

Where the w is the inertia weight which adjust the level of local or global exploration of each particle in the swarm; and cognitive factor $\vec{\rho}_1$ is defined as $\vec{\rho}_1 = c_1\vec{r}_1$ and the social factor $\vec{\rho}_2$ is defined as $\vec{\rho}_2 = c_2\vec{r}_2$. c_1 and c_2 are the accelerator factors, and r_1 and r_2 are random values from $[0, 1]$.

The following is the PSO algorithm for optimization Franken [2004]:

1. Instantiate a swarm Φ of particles to a random positions in a n-dimensional hyperspace Ω where each particle $\vec{X}_i$ is a vector representing a potential solution to the optimization problem.
2. Initialize each particle velocity vector $\vec{V}_i$ to zero.
3. Initialize each particle's best position $\vec{Y}_i$ to the current $\vec{X}_i$ position.
4. Repeat the following steps while no convergence, where t representing the current time step:
   (a) Determine each particle fitness using the fitness function F related to the problem domain.
   (b) Update each particle's personal best position $\vec{Y}_i$:
   If $F(\vec{X}_i(t)) > F(\vec{Y}_i(t))$ then $\vec{Y}_i(t) = \vec{X}_i(t)$
   (c) Update the neighborhood's best particle position $\vec{Z}_i$. The following equation update the neighborhood best position using a Star topology (see details bellow):
   If $F(\vec{Y}_i(t)) > F(\vec{Z}_i(t))$ then $\vec{Z}_i(t) = \vec{Y}_i(t)$
   (d) Update the velocity for each particle $\vec{V}_i$ taking into account $\vec{Y}_i(t)$ and $\vec{Z}_i(t)$ according to the equation 4.2.
   (e) Update the n-dimensional of position for each particle $\vec{X}_i$ in the swarm according equation 4.1

The convergence is reached in the following cases, the maximum interaction reached, the average or maximum fitness value does not change significantly over a certain number of interactions, or, an acceptable solution has been discovered that returns a satisfactory fitness.

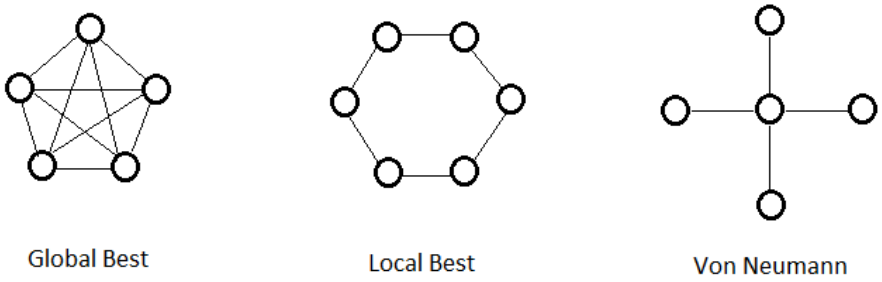


Figure 4.4: PSO Topologies

4.2.2 PSO Topologies

In PSO can be found many topologies, the ones that are described and used in this work are the followings:

- Star Topology: All particles exchange information with each other forming a fully interconnected social network. In this topology all particles are affected for its best personal solution and global best solution, which the is the best solution of the swarm. The algorithm that use this topology is known as GBEST PSO.
- Ring Topology: The neighborhood size is defined in which a number of particles which a particle can exchange and share information with it. if a neighborhood size is 3, so, the particles of immediate left and right are selected. The Algorithm that use this topology is called LBEST PSO.
- Von Neumann: Particles are arranged in a lattice using this topology, where each particles is connected with its immediate top, bottom, left and right particle.

These three topologies can be observed in the Figure 4.4. There are other variations of these topologies as spatial social networks Suganthan [1999], Lewis [2009] where neighborhood are formed based in the euclidean distance in the search space, growing neighborhood Suganthan [1999], small-world social networks Kennedy [1999], and other topologies. Depending of the problem domain, can be selected one of these topologies mentioned. For this work has been used the three described topologies Star, Ring and Von Neumann.

4.2.3 Some considerations in the PSO parameters

In the previous section have been defined some equations which is used in the PSO algorithm which can vary a little bit depending on the topology used. Accord-

ing to some experiments performed some optimization should considered some recommendations to the parameters of the equation 4.5.

The inertia weight w is used to control the exploration abilities in the swarm, large values of w facilitate more exploration, while smaller values focus the search on smaller regions of the search space. The cognitive and social factors are positive accelerator constants used to scale the contribution of the cognitive and social components respectively. The parameters r_1 and r_2 are random values in the ranges of $[0, 1]$. To avoid velocities and positions exploding towards infinite values it was suggested to maintain the accelerator factors c_1 and c_2 according to the following equation Kennedy [1998]:

$$c_1 + c_2 \leq 4 \tag{4.5}$$

Convergence can be ensured by maintaining a relation combing the parameters w, c_1, c_2. This relation was defined in den Bergh [2002] as is shows in the following equation:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{4.6}$$

with $w \leq 1$.

4.2.4 The Problem Domain

The problem domain in this work is to identify the trend reversal movements of the some indexes selected to predict their future movements and obtain some profits. The indexes selected for this work are Exxon Mobil, General Electric, Microsoft, AT&T from US stock market, and HSBC, Vodafone, BP, Vodafone, Rio Tinto Group from UK stock market.

The inputs to be used in these trainings are some technical market indicators (TMI). The TMIs are time series that are derived from applying a mathematical formula to the price data of a specific security or stock price. The price data is broken down into periods, which can be intra-day, daily, weekly, monthly and yearly. The TMI indicators expose some properties and behaviors that usually are not clear by inspecting the price of the stock alone, that properties and behaviors provide an unique perspective on the strength and direction of the underlying security's future price.

In this work was combined the effects of a different indicators to obtained more consistent results for trading decision, according to Papacostantis [2009] the selected TMI indicators complement each other.

One of the indicators used is Aroon indicator, this indicator attempts to detect trend reversals and trend strengths quickly. It measures the number of time periods within a predefined time window since the most recent high price and

Table 4.1: TMI time series for empirical study

Indicators	Time Series
Aroon	AroonUp
Aroon	AroonDown
Boollinger Bands	%b
RMI	RMI
MACD	PriceMomentum
MACD	PriceTrigger

low price. The main assumption this indicator makes is that a security's price closes at a high for the given period in an up-trend, and at a low for the given period in a down-trend Papacostantis [2009].

The calculation of these indicators AroonUp and AroonDown used in this work are shown in the equations 4.7, 4.8, 4.9 and 4.10.

$$\mathrm{AroonUp}_p(t) = 100\frac{\mathrm{HighIndex}_p(t)}{p} \tag{4.7}$$

$$\mathrm{HighIndex}_p(t) = \mathrm{index}(\max\{\mathrm{price}(j)\}_{j=t-p}^{t}) - t + p \tag{4.8}$$

$$\mathrm{AroonDown}_p(t) = 100\frac{\mathrm{LowIndex}_p(t)}{p} \tag{4.9}$$

$$\mathrm{LowIndex}_p(t) = \mathrm{index}(\min\{\mathrm{price}(j)\}_{j=t-p}^{t}) - t + p \tag{4.10}$$

Where p is the size of the time window (periods in days), $\mathrm{HighIndex}_p(t)$ the number of periods within $\mathrm{HighIndex}_p(t)$ the number of periods within p since the most recent highest observed price, and $\mathrm{LowIndex}_p(t)$ is the number of periods within p since the most recent lowest observed price.

In the Table 4.1 is summarized the TMI time series indicators used in this work. For more details how is these TMI indicators are calculated can be refereed to the Appendix C.

The output of the network will be the trend reversal confidence which the values are in the interval (0,1). The aim of the trend reversal is to identify switches from up trends to down trends and vice-versa, how trend switching is identified is explained in the Figure 4.5. So, the trend reversal confidence time series σ is produced by the neural network after providing the time series as inputs at each time step. The trading actions that are used are {BUY, SELL, CUT} according to the following rules:

- If $(\sigma(t)) \leq \theta)$ then BUY

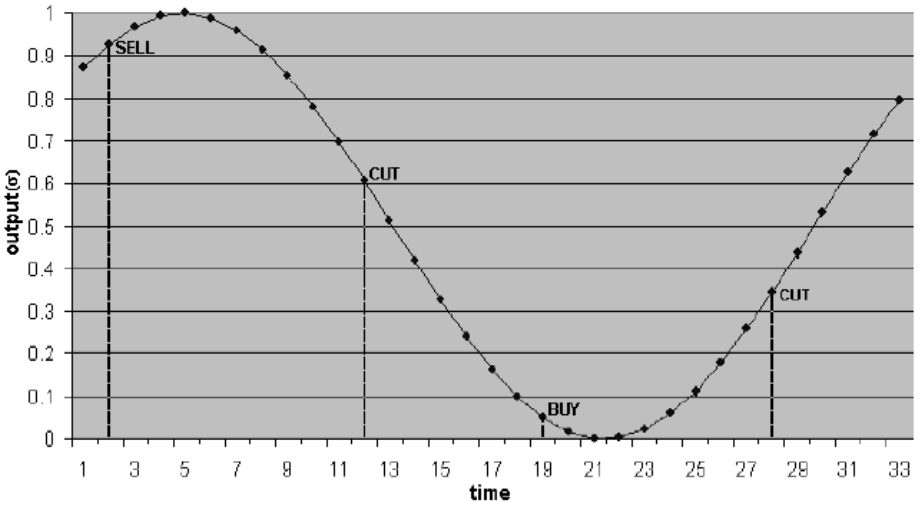


Figure 4.5: Trend Reversal Confidence used to calculate the trade action

- If $(\sigma(t)) \geq 1 - \theta)$ then SELL
- If $(\sigma(t)) > 0.3)$ and $(\sigma(t)) < 0.7)$ then CUT
- Else no trade action is returned

Where θ represents a threshold value of the sensitivity in triggering BUY and SELL actions and should be in the range $0 < \theta < 0.3$. The value θ determines the risk factor that a trading strategy is willing to take on. Small values of θ result in less riskier trades taking place, while large values return higher risk strategies.

A BUY actions entails buying the security, while a SELL action means short selling of the security. A CUT action signals means getting out of a position, if one is held. For example, selling the bought stock or buying back the stock that was sold short.

As it is shows in the Figure 4.5, the value of $\sigma(t)$ oscillates between 0 and 1, indicating the confidence in the trend reversal of that security. Low values of $\sigma(t)$ close to 0 indicates high confidence that a trend reversal from a down-trend to an up-trend is likely. The security in this case is considered to be over-sold and a trend reversal, leading to a price increase is expected. A high value of $\sigma(t)$ close to 1 indicates a high confidence in the trend reversal from an up-trend to a down-trend. The security in this scenario is considered to be over-bought and a drop in price is expected to take place, in this work a value of θ as 0.1 was selected. In the range $\sigma(t) > 0.3)$ and $\sigma(t) < 0.7$ there is not confidence in a reversal, indicating uncertainty about the direction of the price in the analyzed security.

The investment amount is the total monetary amount available for the trading strategy to invest in, for each BUY and SELL action derived. The investment amount can be seen as the investment capital available to the agent. When the neuronal network derive a BUY action, that security is bought to the value of the investment amount. When the neural network derives a SELL action, a short sell is performed.

So, deriving the trade actions for each time step results in the buying, selling short and trade cutting of the stock at various time points. The number of bought or short sold stocks per trade is calculated, based on the value of the investment amount. Depending on how the stock price fluctuates, capital gains and capital losses are calculated when the trade position is closed as is explained in the Table 4.2.

So, for the propose of explaining how the capital gain and losses are calculated an example in the table 4.2 is used Papacostantis [2009]. For this example an investment amount of $ 1,000,000 is used. The agent enters a trade in a specific security for the first time at t=3. No prior trading was done, with no capital gains or losses. The agent buys the stock at the price of $ 400, investing an amount of $ 1,000,000. A total of ($ 1,000,000/ $400) = 2500 shares are bough which are sold at t = 7, given the increase in the stock price by $ 100, the agent obtain a capital gain of $ 250,000. A short sell action follows at t=14, at price of $ 500, where 2000 shares are sold short. The price drops by $ 300 at t=22, resulting in a further realization of $ 600,000 as capital gain. A capital loss is incurred at t = 29, when the price of 5000 short sold shares increases by $ 300, contributing to a $ 1,500,000 loss. A further price drop of $ 100 from t = 35 to t = 40 results in a capital loss worth $ 200,000, because a total of 2000 shares were held. Given the sequence and timing of the agent trade actions in this example, capital gains worth $ 850,000 were accumulated and a capital loss of $ 1,700,000. In total the loss has been of $ 850,000.

The calculation using the transaction costs can be observed in detail in Papacostantis [2009], for simplicity to present this work the transaction cost was not considered. The propose to present this work is to explain how the co-evolution is used with another techniques as PSO.

4.2.5 The PSO co-evolution model

The model proposed by Papacostantis [2009] was called CEPSO, and it is used to co-evolve the best strategies for security trading agents. This is PSO co-evolutionary algorithm are described below:

1. Calculate the TMI time series data, based on the security price data as described in chapter 3. Divide the TMI time series data into two sets,

Table 4.2: Capital gains and losses calculation example

Time	Action	Price	Shares	Capital Gains	Capital Losses
3	BUY	$ 400	2500	-	-
7	CUT	$ 500	2500	$ 250,000	-
14	SELL	$ 500	2000	-	-
22	CUT	$ 200	2000	$ 600,000	-
23	SELL	$ 200	5000	-	-
29	CUT	$ 500	5000	-	$ 1,500,000
35	BUY	$ 500	2500	-	-
40	CUT	$ 400	2500	-	$ 200,000
			Total	$ 850,000	$ 1,700,000

called in-sample and out-sample data, if in-sample data contains n data points and out-sample data m, so, this condition should be accomplished $\mathfrak{m} > \mathfrak{n}$.

2. Create a single warm of particles and randomly initialize the position of each particle within the range $[\mathfrak{r}, -\mathfrak{r}]$.

3. Set the velocity of each particle within the swarm to zero.

4. Initialize each particle's personal best to its current position.

5. Repeat the following steps for a total of n generations.

   (a) Use the current position vector of each particle within the swarm and create a population of feed forward neural network by assigning the particle position vectors as NN weight vectors.

   (b) Using the in-sample data series calculate the trend reversal coefficient σ..

   (c) Using the calculated σ, calculate the performance of each agent simulating the trade actions BUY, SELL, CUT discussed previously.

   (d) Calculate the competitive fitness function for each agent's solution discovered in the previous step.

   (e) Add the best solution to the Hall of Fame.

   (f) Update each particle's fitness according to the calculated fitness.

   (g) Update each particle's best position as per equation 4.3.

   (h) Update the neighborhood best position as per equation 4.4.

   (i) Update velocity of each particle.

(j) Update the position of each particle in the swarm

6. Using the out-sample data, calculate the trend reversal coefficient σ for the agents in the Hall of Fame.

7. Using the out-sample data, calculate the performance using the fitness function simulating the trade actions (BUY, SELL, CUT).

8. The best solution or agent of the Hall of Fame needs to be determined based on out-sample data. So, the best solution for the model is the agent with more high fitness for this security.

4.2.6 Competitive Fitness Function

The Competitive Fitness Function is calculated taking in account net profit and Sharpe ratio in the following equations 4.11 and 4.12. In the equation 4.11 is calculated the profit and loss (P & L), where CG is the capital gains, CL is capital losses and TC is transaction costs of the strategy followed by the agent.

$$P\&L = CG - CL - TC \tag{4.11}$$

In the equation 4.12 is calculated the Sharpe Ratio (SR) to measure the risk of adjusted performance. Where r_S is the annualized return of the security, r_F is an annualized benchmark risk free rate, and σ_S the annualized standard deviation on the returns of the security.

$$SR = \frac{r_S - r_F}{\sigma_S} \tag{4.12}$$

So, the competitive fitness function for each agent in the competition is calculated P&L and SR, where $max_{P\&L}$ and $min_{P\&L}$ are the maximum and minimum trade agent P&L in the set of agents, and max_{SR} and min_{SR} are the maximum and minimum agent SR in the set of agents.

$$F = [\frac{P\&L - min_{P\&L}}{max_{P\&L} - min_{P\&L}}] + [\frac{SR - min_{SR}}{max_{SR} - min_{SR}}] \tag{4.13}$$

4.2.7 Description of the Architecture and Setup of the Experiments

The architecture used to train the TMI time series can be observed in the Figure 4.6. For this training was used a simple neural network with one hidden layer (of two, tree, six till ten neurons), an input layer which will provide the values of the

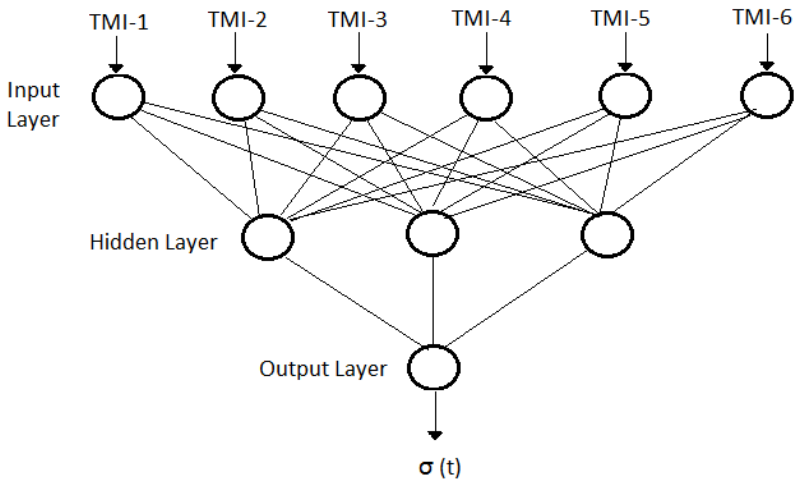


Figure 4.6: Neural Network Architecture used in PSO

TMI time series described in the table 4.1, and the output layer which is providing the value of σ, which indicate the trend reversal of the security analyzed.

The weights of the neural network are the n-dimensional position $\vec{X}_i$ described in the previous section. Apart of that it was used Star, Ring and Van Neumann PSO topologies to update the position velocity and other parameters discussed previously.

The PSO algorithm was executed in 350 generations for each simulation. The swarm size used in the experiments were 10, 25, 50, 100, 150 and 200. For the co-evolution, fulll competition fitness were used, competing each particle against the others in the swarm. So, in the experiments were combined the size of hidden neurons, size of swarm, and the PSO topologies used to identify which one produce the best results for each security analyzed.

The time series used for the price of the securities were from 1st January 2000 till 30th June 2008 which incorporate many events which could affect those prices, as, 11-S, dot com bubble, some economic recessions and others. The in-sample date was selected from 1st January 2000 till 31st March 2006, and the out-sample data from 1st April 2006 till 30th June 2008.

4.2.8 Results of the experiments

Some combinations of PSO topologies, swarm size and hidden neurons produce better results for some companies than for others. For example, LBEST and Von Neumann topologies has better out-sample data performance that GBEST.

In case of Von Neumann solution the best combination is the (200,2, Von Neumann), which means, 200 swarm size, 2 neurons in hidden layer and Von

Neumann topology in which high rates of P&L and Sharpe ratio was obtained. In case of the LBEST topology the best combination was found in (150, 4, LBEST).

Depending on which company was selected, some topologies performed to others. For example, using LBEST topology higher out-sample data performance was obtained better results in General Electric, AT&T, HSBC and BP. In case of Von Neumann topology, the higher out-sample data performance was obtained in Exxon Mobil, Microsoft, Vodafone and Rio Tinto.

Another conclusion is that the use of lower risk factor parameter θ produced good returns, in the experiments with $\theta = 0.05$ it was obtained good results.

The performance of the strategies found by the proposed model out-performed the Bollinger Bands/RMI strategies selected as benchmark. On average the CEPSO agents produce 21.24 % more returns for the in-sample data and 9.09 % for out-sample data.

The other benchmark used for comparison was buy-and-hold (BAH) strategy, used for long-term investment strategies. As in the previous case, the CEPSO model outperform the BAH benchmark providing high returns. In some securities the CEPSO model does not outperformed the mentioned benchmark, but even that, in general CEPSO model produced very good results compared to benchmarks used.

4.3 Discussion on the Works Presented

The intention to include these works is introduce solutions to real-problem from other authors that use co-evolution as learning process. These works and others were useful to understand and propose some techniques discussed in the next chapters. In this section is discussed the works presented previously in this chapter.

4.3.1 Discussion of Predator and Prey simulation work

In this section is discussed the view of the author in the co-evolution of complex strategies for the simulation of predator-prey problem Rawal *et al.* [2010]:

- This works demonstrated that it is possible to evolve complex strategies using co-evolution and complex neural networks structures, and simulate what could happen in the nature in predator-prey interactions. It was observed arm race between predators and preys, but the author believes that is missing a measure whether really the arm race is happening and is not happening red queen dynamics. The author believes that a technique to evaluate whether arm race is happening is test these strategies against

externals agents. For example, probably should be considered to code some agents, i.e. predators or prey based on previous knowledge, and observe if predators or preys have successful results.

- The author believes that one important thing was not analyzed, how general are the strategies discovered?. For example, if the predators evolved are replaced by other predators (evolved in other environments or with strategies hard-coded) the preys can be perform successfully?.

- The other thing that was not analyzed was whether population and subpopulations of neurons are diversified. As it was discussed in the previous chapter, the introduction of diversity is a solution to avoid co-evolution pathologies and evolve more complex strategies, and there are evidences that the diversity create more general strategies. The author believes that measure the diversity during the co-evolution process is good strategy to monitor the process of the co-evolution.

4.3.2 Discussion of Security Trading work

The intention to present this work is show real-world problems can be solved using co-evolution. The author has already started to apply techniques proposed in this thesis for security trading in the financial market taking some techniques from this work and his knowledge in this area. The author has a Master degree in Financial Analysis in Complutense university. Some of the comments to this work are the following:

- The model represents a good approach to model the discovery of strategies using an co-evolutionary learning environment, but, in this work has not been discussed whether really the co-evolution really happened, it means, if the really the global fitness was progressing during the generations. Should be introduce a mechanism to measure the global fitness of solutions learned during the evolution. In the next chapter the author propose a technique how to measure the global fitness.

- It was not analyzed whether the co-evolutionary pathologies was faced during the co-evolutionary process. The author consider that this is a critical factor to ensure that co-evolution is happening.

- It was not discussed how general was the strategies discovered during co-evolution process, as it was discussed in the previous chapter, it is critical in co-evolutionary process to have more general strategies to ensure a good decision (buy, sell or do nothing) when new scenarios happen, i.e. new

events that can happen in the future that can impact the stock prices of companies analyzed.

- It was not discussed how diversity was introduced into the population of particles. Keeping diversity in the populations evolve can ensure that general strategies could be discovered and at the same time mitigate the co-evolution pathologies.

What is missing in both works are the definition of a solution concept for both problems. As it was discussed in the previous chapter this should be the first step to solve the problems in discussion.

These works presented and other analyzed for this dissertation were very useful to understand methods and strategies that are used in co-evolutionary learning of complex strategies and real problems. In this dissertation is used some ideas of these works in the application to solve the Go game, these points are discussed in next chapters.

Chapter 5

Co-evolutionary Techniques Proposed

This chapter starts with the definition of the solution concept for this problem, solve the Go game using co-evolution. This definition is very important to understand how to solved the first and secondary search problem described previously. In the section 5.3 is discussed some approaches than other authors intended to solve this game taking in account the feature of the game as the symmetry dividing the board in different sectors.

In the section 5.4 is presented the Replacement Immigration Rate (RIR) as a solution to introduce in controlled way more immigrants to the population. In the section 5.5 is introduced a memory mechanism to reinforce in the genes of the blueprints good strategies learned in the previous generations. In the section 5.6 is introduced the dynamic sizing method for the blueprint's structures.

In the section 5.7 is discussed about the CFS method, and introduced a new proposed fitness sharing called Competitive Fitness Sharing Augmented (CFSA). In the section 5.8 is presented an co-evolutionary algorithm for co-evolve two populations of players. In the section 5.9 is discussed how the co-evolution pathologies are mitigated.

In the section 5.10 is discussed the method used to measure the generalization of solutions discovered, and in section 5.11 the method to measure the genotype diversity of population evolved.

The section 5.12 describes the techniques that are used in this dissertation to monitor the progress of co-evolution, and finally in the section 5.13 discusses about the evaluation functions used in this thesis to ensure the co-evolution of two competing populations.

5.1 Solution Concept for A Computer Go player in a Co-evolutionary Strategy

Using the definitions discussed in the previous sections, in this section will be defined the Solution Concept used in this thesis.

Having two different populations of neurons, structured as a single layer represented by their inputs, outputs and weights, and defining a blueprint, which is a representation of a computer Go player, which is formed by neurons from player's population; the solution concept for the domain, from the player's blueprint perspective, is the blueprint with the best fitness obtained calculated using an evaluation function after the competition of the player's blueprint against opponent's blueprint from the opponent's population in the last generation. In this domain, it is going to be represented a host-parasite relationship, were host is going to play black stones and parasite white stones, and both players are evolving based on the strategies or test cases discovered by the opponent.

The formal definition of the solution concept is the following: The measurement M of this solution concept O is obtained by comparing the fitness obtained by the player $\mathfrak{i}$ in a competition E against an opponent $\mathfrak{j}$, thus, the measurement ν will be an evaluation function calculated for the player $\mathfrak{i}$.

The solution concepts for this domain will be the highest $\mathsf{M}(\mathfrak{i}, \mathsf{E}, \nu)$ measure in the last generation $\mathfrak{g}$. Assuming that in every new generation new strategies are discovered and maintained in the blueprints, the last generation should have the blueprint, or player, with the best strategy to play Go.

In the previous definition, implicitly is defined the preference between different solutions in the solution concepts obtained in the every generation. Thus, will be preferred the solution from last generation to the solution of the previous generations.

In this domain the solution concept O can be more easy defined extensionally instead of intentionally, as the set of properties needed to defined a configuration K is more complex. Thus, the set of solutions of configuration K for the solution concept O are the blueprint, or players, with the highest values calculated using an evaluation function in every generation.

Some properties of the configuration K are defined implicitly with the application of different evaluation functions as metric ν, in this work is used different evaluation function as Competitive Fitness Sharing (CFS), Competitive Fitness Sharing Augmented (CFSA) proposed by author, and others. Thus, properties as if the game was won, number of stones in the board, territory, area, and number of stones captured are implicitly defined in evaluation functions used and in counting method selected.

Thus, in the experiments are selected the Chinese scoring, which is an area

counting method considering all living stones in the board and the territory surrounded by a player's stone. In case of Japanese is selected should be counted empty points in the board surrounded by living stones and stones captured at the end of the game.

The behaviors complex X_i represent behaviors of the blueprint i, which is a collection of neurons from the population. Every neuron in the population is specialized in some sub-domains of the problem; with the goal to contribute to get the best score as possible when are being part of a blueprint. Thus, the best blueprint, or player i, in every generation t, represent the best combination of the specialized neurons with the behaviors B_i needed to competitive against opponents of that time t.

The main issue of this solution concept is to identify what is the last generation g till these solutions will evolve.

5.2 Evolving a Computer Go player

SANE is a Neuro-Evolution (NE) technique proposed by Moriarty [1997] where weights, inputs and outputs of the network structure are evolved while searching for solutions in the search space, or solving the problem. SANE applied to games as Go, maintains the best strategies of the game keeping the best collection of networks or blueprints (which have obtained the best fitness against opponents) and evolving them through generations. SANE has a Hall of Fame which keep these best blueprints are reproduced replacing worse performing blueprints.

For this work was selected SANE because of easy implementation and because of the main assumption behind this is that all members of the populations to be evolved should be the same specie. This important because of there are other NE methods which evolve structures as Topology and Weight Evolving Artificial Neural Networks techniques (TWEANN) in which different species are combined. As Smith [1989] mentioned, it is demonstrated in nature, genetic distance between two species is highly correlated with mating discrimination and the likelihood that if interspecies mating does occur the offspring will either not survive or be sterile.

Although other methods as NEAT has demonstrated better results than SANE for some problems, it is proposed some techniques which are applied to SANE (which can be applied to other evolutionary methods) and improve search of solutions. So, in this thesis is proposed some techniques that can improve SANE, some of them are described in in Zela & Zato [2011], which introduced the concept of some population of neuron immigrants which are added to original neuron populations to increase diversity. This was called SANEi , and the new neurons introduced to the populations using an immigration rate should belong to the same specie, thus, gene size of the immigrant population must be the same to the

original population.

As it was discussed in the previous sections, SANE has two parts, Evaluation and Reproduction phase. In the evaluation phase, SANE simultaneously evaluate the blueprints networks and neurons. The blueprints are evaluated by the performance to solve problems; in this thesis beat to an opponent Go player, and it was discussed, the best solution to the solution concept of this problem domain is the neural network that gets the best score against other computer Go player.

The neurons are evaluated based in the performance of the blueprints in which the neurons are participating. In an evolutionary approach, the score obtained by every player or blueprints in the game is added to fitness of the each neuron that belongs to the blueprint network. After all networks have been evaluated (played against other go players), the fitness of each neuron is normalized by dividing the sum of the scores by the number of total networks in which the neuron has participated.

In a co-evolutionary approach, the score obtained by every player is replaced by another evaluation functions as competitive fitness sharing (CFS) Rosin & Belew [1997] obtained by players after the interaction or competition against evolving opponents. This other population or opponents can be same species (for this thesis will have same gene sizes) or other species.

In the reproduction phase, SANE uses the genetics operators as crossover and mutation to get new blueprints networks and neurons. The blueprints and neurons used for reproduction belong to Hall of Fame. So, to apply crossover every population (neurons and networks) is ranked based on their fitness obtained after all interactions, and it is defined an elite of members for each population which will used for mating to other members of the population replacing the members who the worse performance. After crossover, mutation operator is applied. So, for worse members of the population of blueprints and neurons mutation operator is applied.

Lubberts & Miikkulainen [2001] mentioned that evolving neurons instead of complete networks, the search space is decomposed and groups of neurons are able to specialize on different parts of the task, or in different intersection of the Go board. This way, diversity is maintained and the algorithm does not get stuck on a suboptimal solution and the blueprint population then searches for effective combinations of neurons.

As Perez-Bergquist [2001] mentions, the use of blueprints ensures some level of consistency in network composition from generation to generation, and helps to prevent high levels of redundancy where multiple identical or nearly-identical neurons are found in a network.

According to Perez-Bergquist [2001], SANE have a few problems, as for example, many times, neurons with completely different weights and purposes are bred, producing offspring that fill no useful role, although sometimes this creates

a new neuron type that is indeed needed. It is difficult achieve a critical mass of a given neuron type, wherein such neurons breed with reasonably similar neurons often enough to produce useful results. According to Perez-Bergquist [2001] the solution to solve the problems found in SANE is to segregate neurons on the basis of type and proposing ESP (Enforce Sub-Population) method.

5.3 Division of the Go Board and Approaches to Solve the Game

Some games has some features as symmetry which can be used as advantage to solve the game reducing its complexity or the number of possible moves as it was discussed by Pazos [1980] in the game called the Star, which the tree used to solve has 1,022 status but using properties as symmetry and anti-symmetry this could be reduced to five nodes, As Pazos mentions that one of advantage of using symmetry if that can be reduced the combinatorial explosion needed to solve a game. In this thesis will be used an application of symmetries discussed by Pazos [1980] with the intention to reduce the computing resources needed to evolve strategies.

In the Go board can be observed some symmetries in the positions played in the board. For example, if we can divide the 19×19 board in four zones as in the Figure 5.1, and the strategies played in any zone of this board could be similar. Some authors as Perez-Bergquist [2001] used this feature to divide the board to create and force some network specialist in these parts of the board.

Rather than having one large pool of neurons with network blueprints collection these neurons and applied by Richards *et al.* [1998] using SANE, ESP maintains a separate population of neurons for each position in the network as can be observed in the Figure 5.2. Building a network then consists of selecting exactly one neuron from each of these sub-populations.

The Figure 5.2 shows the networks of neurons created in the 7×7 board. Each network is composed by a 3×3 portion of the board, trained in different parts of the board, as four networks at corners, four at edges, and one network in the middle of the board, having totally 9 networks. The same was applied to 9×9 board having in total 16 networks as is shows in Figure 5.3.

As it show in the Figure 5.2, two inputs were selected per each positions of the board, with the first position reading 1 if that position is occupied by a white stone, zero otherwise, and the second position reading 1 if that position is occupied by a black stone, zero otherwise.

As the populations of neurons are evolved separately, each population is able to focus on a particular function more quickly (for that particular position in the

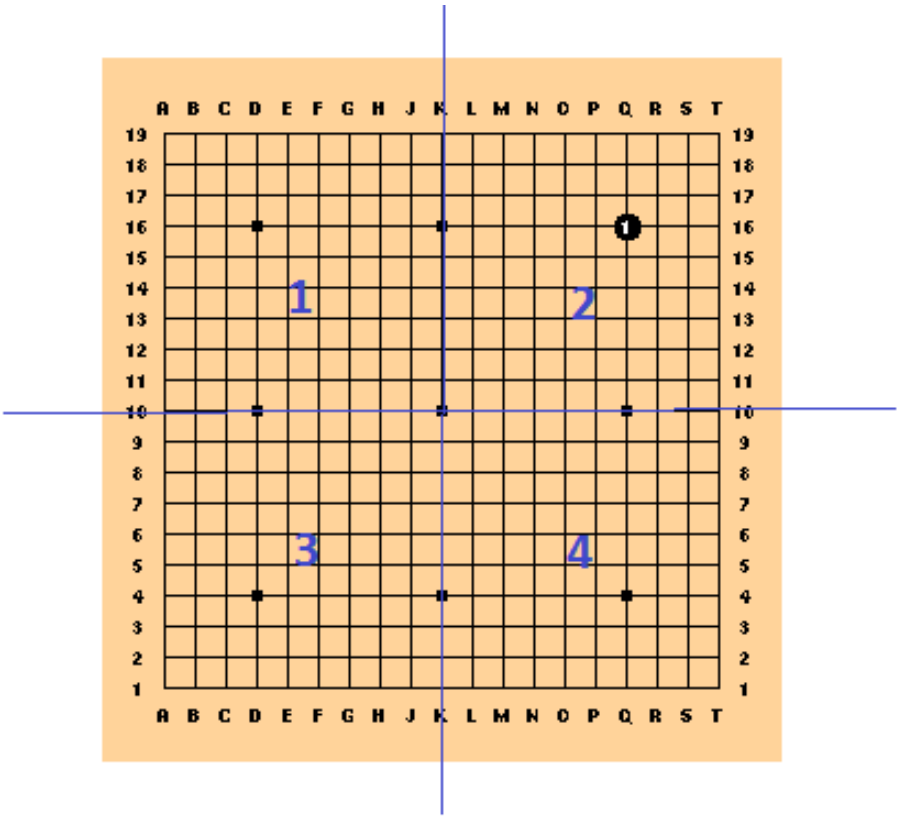


Figure 5.1: Division in Four Symmetric Zones the 19x19 board

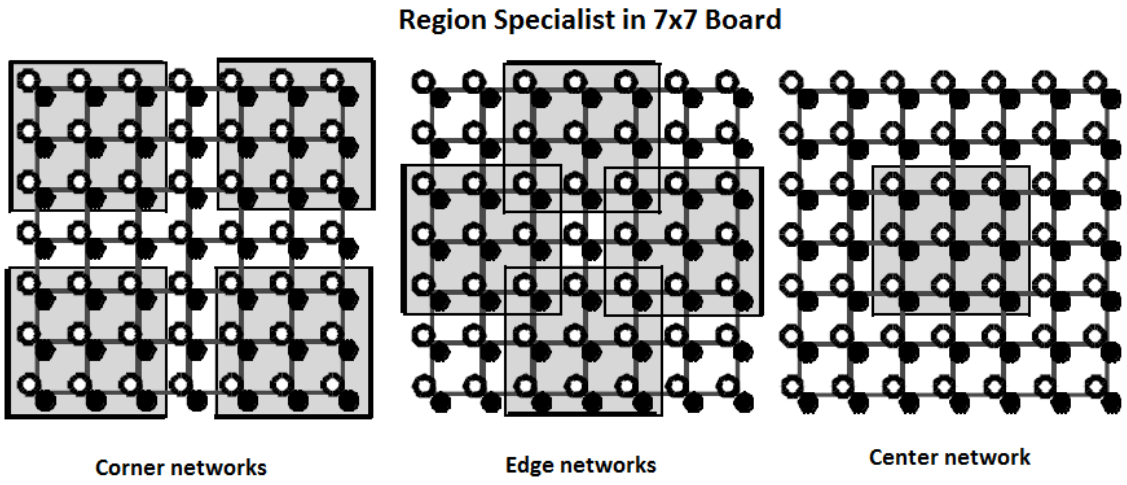


Figure 5.2: Specialist networks implemented by ESP in a 7x7 Board

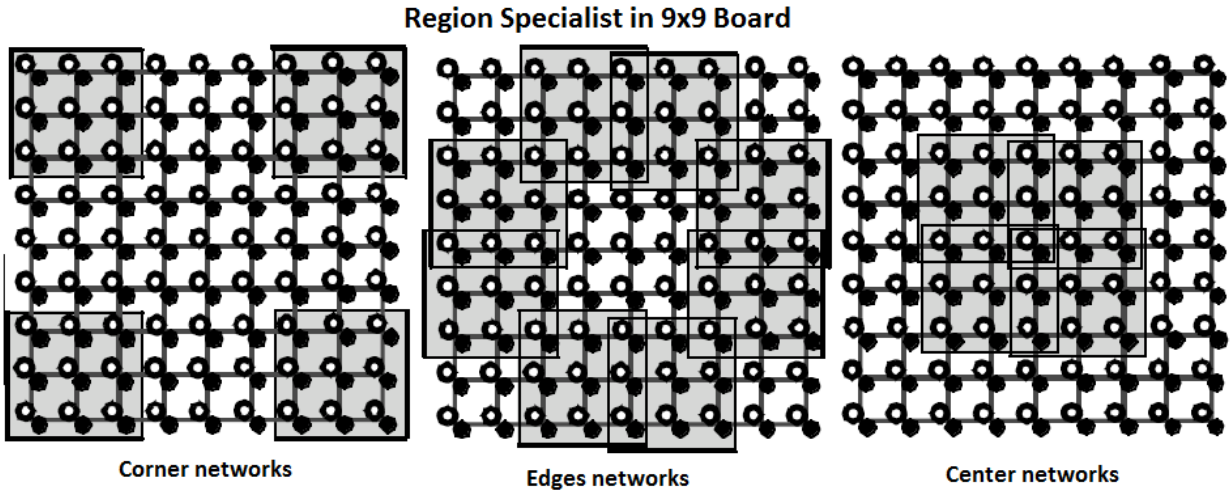


Figure 5.3: Specialist networks implemented by ESP in a 9x9 Board

board), and networks are less likely to have redundant neurons, as it suppose to happen in SANE.

The results of experiments were successful in the sense that ESP learning were more effective than single hidden network for all the position of board using SANE evolving against Gnugo (version available at 2001) playing some random movements. ESP needed less generations to start beating Gnugo that SANE, but at some generations later the performance (number of wins against Gnugo playing random) of both structures were similar.

According to Perez-Bergquist [2001], single hidden network for all the position of board neither ESP techniques worked at board sizes of 9×9 and up, according to him, some sort of change have to be applied there. When size of the board increased the game gets more complicated, and simply throwing more neurons to the hidden layers does not appear to be a solution. A reasonable explanation for behavior is that at small board sizes, a random evolutionary search technique that relies only on end-of-game scores can effectively explore the game space, but that at 9×9 the game becomes sufficiently complex that the evolutionary search cannot even find a hill to climb.

In Richards *et al.* [1998] was found that network with about 300 neurons were needed for optimal performance on a 7x7 board. In Perez-Bergquist [2001] ESP produced good results with just 10 neurons per network. So, ESP was much more efficient encoding than SANE. According to him, this is probably because of SANE networks have large amounts of redundancy in the form of fairly large numbers of identical or near-identical neurons.

The same was observed in some experiments done by the author, it means,

apparently when more neurons are introduced in the networks is not observed better performance in the learning process.

As the author is not expert playing Go, it was consulted to amateurs and some professional players in Go, and the consensus was that it is difficult to divide the board in sectors and play only in that sector without taking care of the entire board, basically because there is a dependency between all the sectors in the board, and this is more notorious at the middle and end game, where the connections between all of these sectors are more clear.

So, to mitigate the redundancy of networks introduced by SANE, the technique proposed will create blueprint networks with number of neurons dynamically different, it means, in every generation new blueprints will contain different number of neurons on it pointing to different positions of the network, and not necessarily to some sectors. Obviously the best structures, the blueprints with better collection of neurons, in terms of quantity and quality, will survive and evolve for the next generations.

So, taking advantage of the symmetry that can be observed in the board, it was reduced the number of inputs to the network which had been setup initially to two inputs for board position, which actually were used by other authors discussed previously. For every board position it was setup one input, the same for the output. In boards of 9×9, the blueprint structures have 81 inputs and 81 outputs. The other reason of doing this is reduce the computation time because of in some preliminary results was not observed a big difference in terms of learning strategies using one or two inputs per boar position.

Considering this configuration, for the co-evolution of two players, Black and White, the input values for the player will be +1, and -1 for the opponent. This readings will be the same if the player is playing parasite or host (in the parasite-host interaction), meaning that from the parasite point of view, any move performed by them will be +1, and -1 if this is performed by the host, and the same from the host view.

Apart of that, to give some indications to the players which was the last move from the opponents and by himself, the readings for the last moves were differentiated from previous moves. Thus, the last move of the player will be +10, and the last move for the opponent -10. The intention of doing this is to distinct clearly last moves when these board positions are evaluated by the sigmoid function, and with the intention to force next moves to be selected around the last moves. In other games as Chess probably these input values should be others and not +1/-1 or +10/-10.

Figure 5.4 (a) shows the architecture implemented to be used in this work. As it is show, for every intersection in the board there is one input and one output.The hidden layer connects the input and the outputs

Coming back to the combinatorial approach discussed previously, some au-

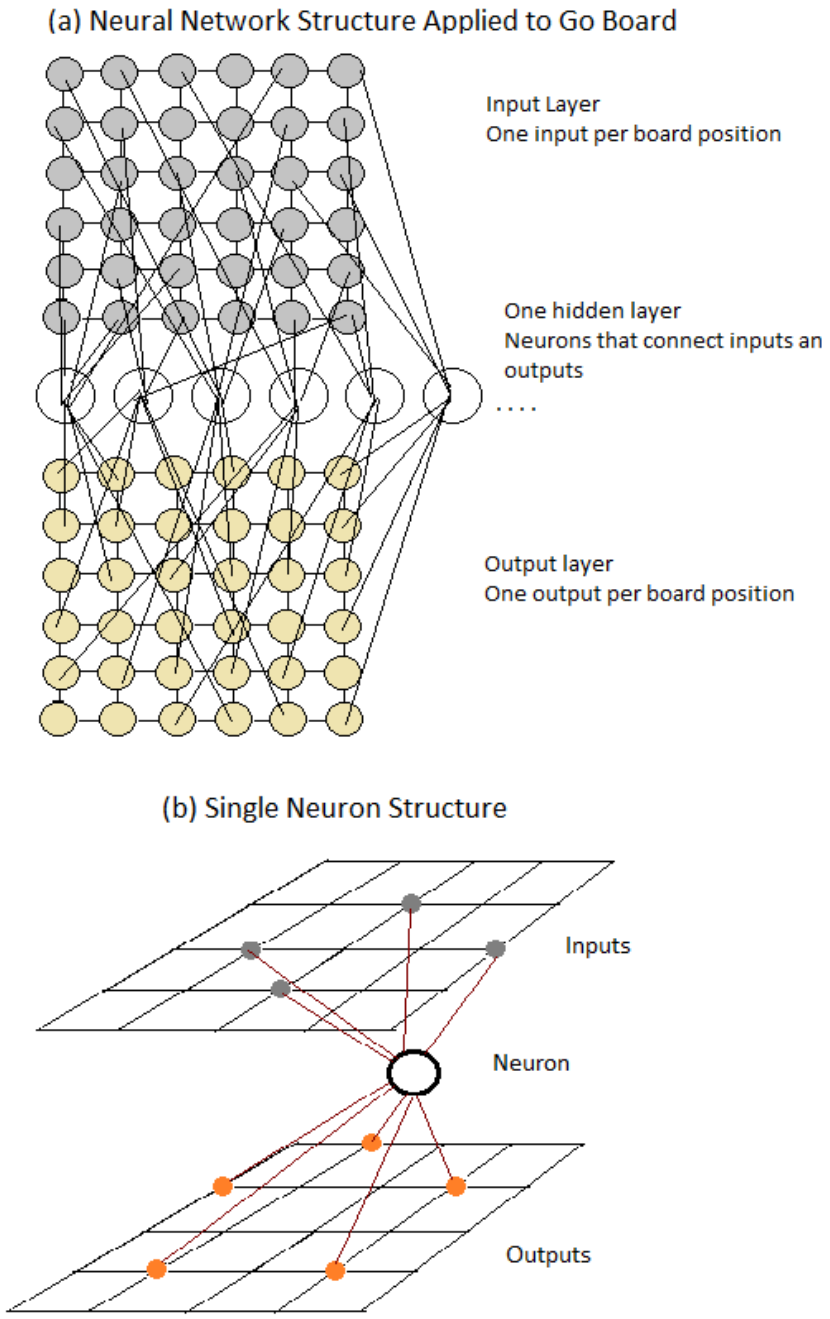


Figure 5.4: Structure of the architecture implemented for the Go player evolved with Sane

thors as Perez-Bergquist [2001], who propose use different population each one being different solutions to the problem, considers that is not necessary to have many populations of solutions because of every neuron of the population in SANE is by itself a different solution to the problem, and because it is difficult and not recommendable to divide the board in different regions, the Go game should be see as only one game in the full board instead to divide it artificially in many games.

So, combinatorial approach is not totally discarded because every neuron which is part of blueprints are partial solutions to the problem, meaning that should be a sub-game of the global game. The blueprint focus in the global game and the neuron focus in sub-game of this global game. So, during game multiple sub-games, from the view of of neurons, are evaluated to select the best move based on the status of the global game, or the view of the blueprint at that moment.

Even in some moments during the game, the focus is in some regions of the board (this can be observed more in some bigger boards as 19×19 board), and after some discussions with some human Go players, my approach to solved this game is see the Go as only one game, from the blueprint's perspective, and not multiple games or at least partially, meaning that sub-games created artificially is solved by neurons. It is fair to say that there are other Go players that see that the solution as the solutions to multiple sub-games. This approach is not totally contrary to some previous approaches to solve this problem as the one propose by Muller [1995] applying a combinatorial game theory to solve the Go game, in the sense that sub-games are not created manually by the programmer, instead this approach sub-games are created randomly and during the competition the best combination of sub-games, or neurons, will be survive and reproduced for next generations and not necessarily will be same because of during the evolution inputs or outputs of that neuron could change. Actually in the neural structure used the number of inputs and outputs are not necessarily the same and for the same section of the board as it can be observed in the Figure 5.4 (b).

The Figure 5.5 shows how a new move is obtained using the architecture explained before, where a blueprint is a collection of neurons from the neuron population and every neuron is connecting inputs and outputs. From a point of view of a player the values of the inputs's positions in the board are the followings: the inputs will have values +1 if the player's stone is in that position, -1 is opponent's stone is there, 0 in case that there is not stone in that position, +10 if position was the last player's move and -10 if it was the opponent's move.

The output is calculated using a Sigmoid function discussed previously in the section 2.6.2.3 (Symbiotic Adaptive Neuro-Evolution), but it was discussed previously any other function can be used. So, the move selected is the output with the maximum value obtained after comparison of all outputs obtained using

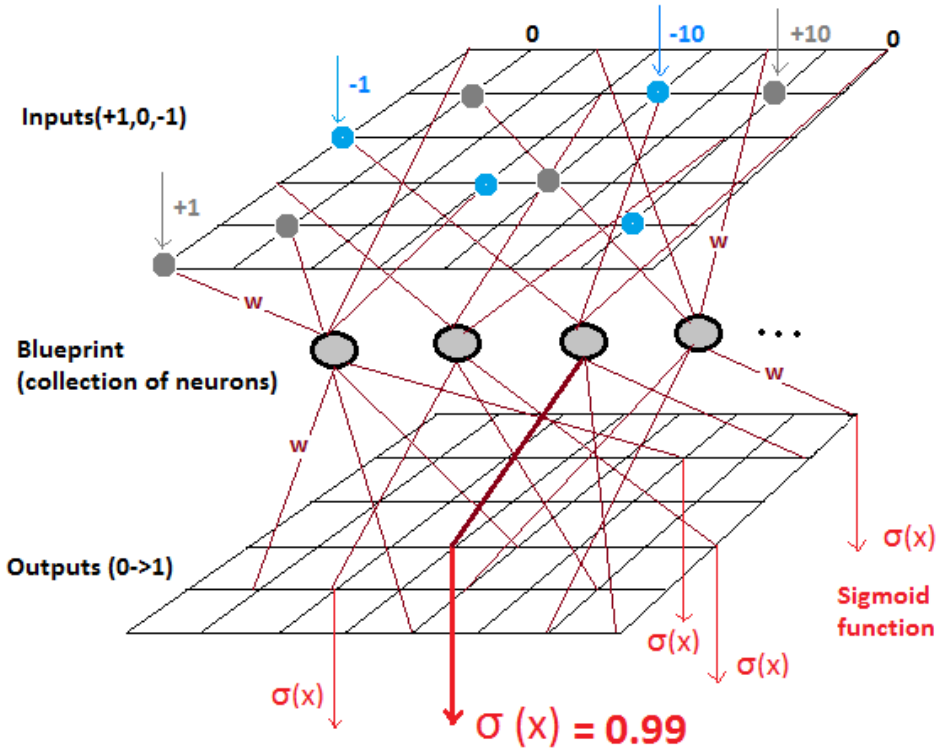


Figure 5.5: How it is obtained Go moves using SANE

sigmoid function. The outputs have values from 0 to 1.

The outputs which has an stone in that position are not considered. After all the moves has been performed and both players have passed two consecutive times, it is calculated the score using the Chinesse scoring method, considering all living stones in the board and the empty intersections that these stones surround.

The same mechanism is performed by all players that compete in every generation. And at the end of the evaluation phase is calculate the fitness function using the evaluation function selected. For the reproduction phase, as it was discussed, the neurons and blueprints evolve using genetic operators.

In the next sections are going to explain the techniques introduced in this thesis by the author for the evaluation and reproduction phase.

5.4 Introduction of Replacement Immigration Rate (RIR)

Zela & Zato [2011] introduced some improvements to SANE method which was called SANEi, because of the introduction of an immigration rate to introduce

new neurons in the population during the reproduction part of SANE. The replacement immigration rate introduce new neurons in the population replacing neurons worse ranked in every generation, with these new members in fact we are simulating a real infinite population of neurons, which apparently is not happening with genetic operators crossover or mutation. The introduction of new neurons in every generation is creating a major diversity and there are indications after the experiments done in Zela & Zato [2011] that SANEi is creating more strategies in the game than SANE.

In demography, replacement migration is the migration needed for a region, or country to achieve a particular objective which could be demographic, economic or social. As in Marois [2008] , some studies using this concept have as an objective to calculate the number of immigrants needed to prevent the total population decline, or the working-age population decline and apply some mitigation policies. This concept can vary according to the context applied, number of annual immigrants, net immigration (different between the number of people entering and leaving a country), and others.

In this context, the replacement immigration will be a rate of immigration needed to keep the net immigration equal to zero, replacing the worse performance neurons and looking for neurons that can contribute better to the tasks of the population. In this location can be found some net immigration rates for all the countries in the world Country Comparison - Net migration rate,

The replacement immigration rate will be auto-calculated based on the progress of the evolution.

$$\mathsf{RIR} = \frac{\beta}{e^{\frac{-\mathsf{GNL}}{\mathsf{TG}}}} \tag{5.1}$$

Where RIR is the replacement immigration rate that will be used to replace the worse performance neurons, GNL is the number of games not lost by player during the previous generation and TG is the number of games played by the players, and β is the parameter used to increase or decrease the RIR in the experiments performed.

When GNL is equal to TG is because the population of the player (i.e. predators) has won all the games to the opponent's population (preys), and this happens when predator's population is more superior to the prey population which means that lost of gradient pathology is observed in the co-evolutionary process. So, the introduction of new neurons using RIR rate will intent to remove this pathology from the predator-prey co-evolution.

So, RIR is calculated automatically after every generation based on the results (i.e. number games won) that the population obtained in the competitions with other populations, promoting more diversity when it is find superior populations. When one population is out performing against other population (i.e. a superior

population), RIR is a greater value, introducing of new members in the population for the next generation. When the population has less successful results against the opponents, the RIR value is less, meaning that less new members are introduced in the population.

In the execution of the experiments was used different values of β which can facilitate more fast or slow the introduction of new members into the population.

5.5 Memory for Reinforcement Strategies

The other contribution introduced in this thesis is that every player (blueprint) will have a memory with the moves done during the game. The use of this memory has the propose to reinforce into the neurons of the population some strategies that obtained the best results in the previous competition. All the blueprints or players that share the same neurons are reinforced with these strategies. This reinforcement is not only to the neurons which belongs of Hall of Fame, actually it is to all neurons in the population that are evolving.

This mechanism was introduced because of during some experiments was observed that some strategies saved in the best players evolved from some generation were lost in the next generations, this is because these players are under the pressure of selection methods and genetic operators. This could be an evidence of forgetting pathology in the co-evolution of these strategies.

According to some Go players consulted and some literature available, some good start-game strategies are to start playing around the center of the board and not in the borders as it was observed in the evolution of players without the use of this memory. So, with the introduction of this memory, we want to reinforce some strategies as to start playing around the center of the board as soon this is discovered during the evolution.

In the experiments executed initially the results were impressive. After some generation, the best players are starting to play around the center of the board as it was expected.

The structure of the memory can be observed in the Figure 5.6. in case of Co-evolution approach the blueprint will save more than one game, in case of an evolutionary approach every player will keep just the moves of one game. So, every move played by the player in different competitions are saved, from there is selected the best set on moves, based on the best fitness obtained, which is selected as strategy to be reinforced.

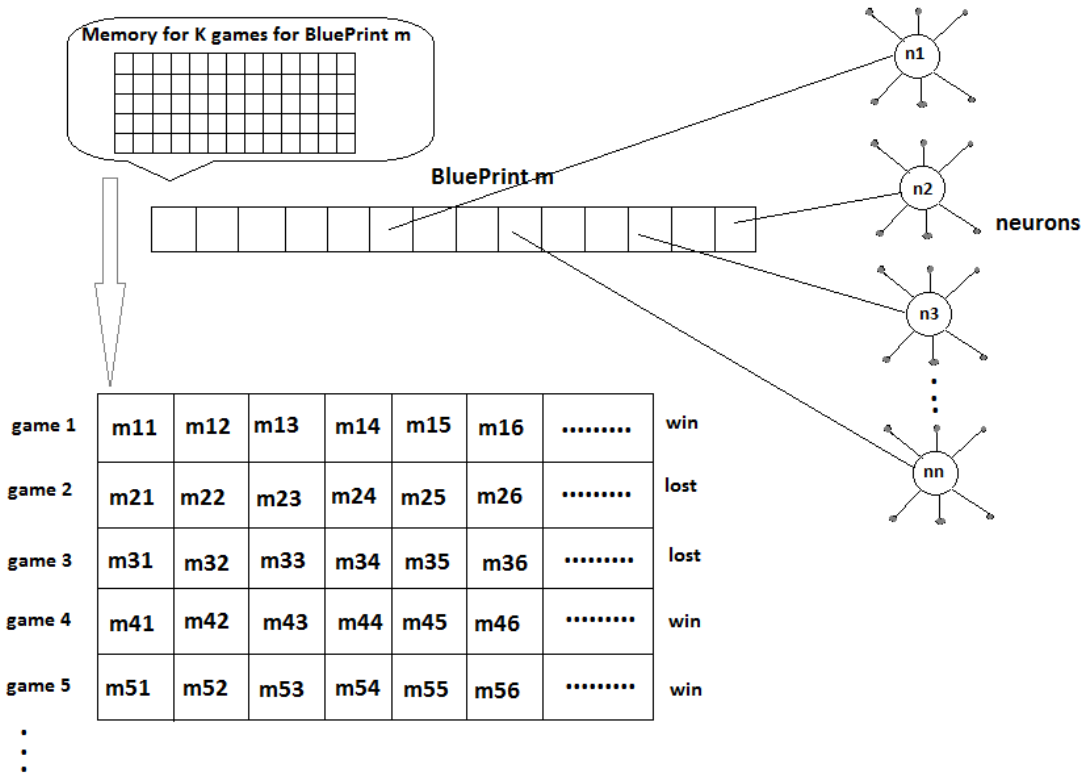


Figure 5.6: Memory introduced in the architecture implemented for the Go players

5.5.1 Memory in the evolution

The implementation of the blueprint's memory during an evolutionary learning of a player (blueprint) which is competing against the same opponent (i.e. Gnugo) is described in the Algorithm 2. In this scenario the best player are the one who have the best score against the opponent (i.e. Gnugo) and its moves are reinforced in the neurons of the players of Hall of Fame for the next generation. These neurons belongs to the neuron's population from which are build the blueprint's population.

```
//For each neuron of the best blueprint, find it in the population and
reinforce the strategies for each neuron i in the population do
    for each neuron j in the best blueprint do
        if neuron i is equal neuron j of blueprint then
            for each move k in the memory of the best blueprint do
                reinforce_strategy (neuron i, move k of best player, pos in
                the memory ) ;
            end
        end
    end
end
// Reinforcement of strategies in the genes of the neuron ;
reinforce_strategy (neuron i, move k of best player, pos in the memory ) ;
for each gene g of the neuron i do
    if gene g is output then
        if gene g is equal to move k then
            gene g+ = 1/(1+e^pos) ;
        end
    end
end
;
```

Algorithm 2: Reinforcement Strategy in an Evolutionary Approach

The algorithm 2 describes how the moves of the best blueprint (the best of Hall of Fame) are reinforced in the neurons of the population using a sigmoid function. The assumption to propose this mechanism is that the best player (which has obtained the best fitness) has the best strategies in its generation, and this should be replicated to the other players. The variable pos is the position of the move in the memory, so, as per the function used, the first moves of this memory are reinforced more than the following moves, and this method is reinforcing more start-game strategies instead of middle-game or end-game strategies. This is because the values obtained applying this function $\frac{1}{1+e^{pos}}$ are going more insignificant when the positions are not the first ones.

5.5.2 Memory in the co-evolution

As it can be observed in the Figure 5.6, the best blueprint, the blueprint which obtained the best fitness compared to other blueprints of the same generation, can play many games in the same generation, so, the best strategy reinforced to the rest blueprints of Hall of Fame, consequently to all neuron's population, is the game in which this best blueprint won and got the best individual fitness $\mathtt{Fitness}_{\mathtt{BBP}_{\mathtt{a}}}$, where BBP is the best blueprint and a is the game played by this blueprint in that generation which obtained the best fitness.

```
//identify the best game in the memory of the best blueprint ;
for each game a of the best blueprint do
    if blueprint won the game a then
        identify the game with the greater Fitness_BBP_a ;
    end
end
//for each neuron belong to the blueprint reinforce the strategy in these
neurons ;
for each neuron i in the population do
    for each neuron j in the best blueprint do
        if neuron i is equal neuron j of blueprint then
            for each move k in the memory of the best blueprint do
                reinforce_strategy (neuron i, move k of best player, pos in
                the memory ) ;
            end
        end
    end
end
// Reinforcement of strategies in the genes of the neuron ;
reinforce_strategy (neuron i, move k of best player, pos in the memory ) ;
for each gene g of the neuron i do
    if gene g is output then
        if gene g is equal to move k then
            gene g+ = 1/(1+e^pos) ;
        end
    end
end
;
```

Algorithm 3: Reinforcement Strategy in a Co-evolutionary Approach

The algorithm 3 describes how the moves of the best strategy of the best blueprint are reinforce in the neurons population using a sigmoid functional.

As in the previous case, The variable `pos` is the position of the move in the memory, so, as per the function used, the first moves in this memory are reinforced more than the following moves, and this method is reinforcing more start-game strategies instead of middle-game or end-game strategies. This is because the values obtained applying this function $\frac{1}{1+e^{pos}}$ are going more insignificant when the positions are not the first ones.

The initial results of the application of this mechanism in an evolutionary and co-evolutionary learning approach gave very good results, where the players are learning good start-games strategies using this mechanism, by contrary not using this mechanism there is not clarity what are the start-game strategies learned. These results will discussed in the next chapter.

5.6 Dynamic Sizing of Players

The other contribution from the author in this thesis is propose dynamic sizing for the blueprints, or in other words, blueprints formed with different number of neurons. As it was discussed in the previous chapters, one of the difficulties using SANE networks is the redundancy in the form of large number of identical or near identical neurons Perez-Bergquist [2001]. With the introducing of dynamic sizing is pretended to mitigate this issue.

Other reason to use dynamic sizing mechanism is because of in some experiments by other authors as Perez-Bergquist [2001], Richards *et al.* [1998] and others, the size of the blueprints used in their experiments are calculated manually after some number of experiments, which sometimes could be a tedious activity and not producing same results because of the context is probably is different in every experiment by different authors. So, with this approach the calculation of the correct size the networks will depend on the process itself and will calculated automatically based on current environment as number of inputs/outputs (i.e. size of the Go board), values of weights that are connecting inputs and outputs, algorithms used for training and others.

When a blueprint is created the size is calculated randomly between a range of minimum_size and maximum_size. The minimum_size value is selected based on the minimum number of neurons that is needed to cover all the input and output in the network assuming that not input and output position is repeated. The maximum_size value can be any value, but this can be selected base of some assumptions as the maximum reasonable number of neurons that blueprints can have. It this thesis, the maximum_value was selected after some trainings performed with blueprints in previous experiments.

One way to identify what should be the range of values used is evolve blueprints and observed what is the number of neurons per blueprint created after some gen-

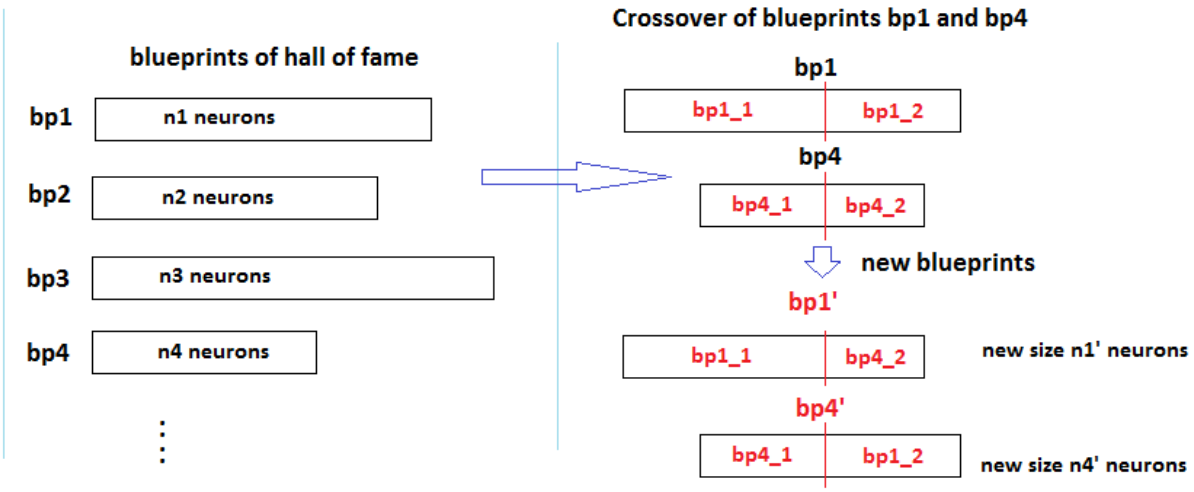


Figure 5.7: Dynamic Sizing and Crossover of Blueprints

erations. It is possible to observe the trend of average size of the best blueprints in Hall of Fame after some generations, and how the blueprints are performing during the evolution (or co-evolution).

In the production part of SANE, the best blueprints of Hall of Fame are combined randomly. In this new scenario, as the size of the blueprints are different, if the size of the new blueprints are between the range setup previously, the new blueprints are created, but if the size of one or both new blueprints are out of this range, these new blueprints are not created.

In Figure 5.7 can be observed the mechanism proposed. In this case it is crossed blueprints bp1 and bp4 with different size creating new blueprints bp1' and bp4' with different sizes as well.

During the experiments the results were interesting, the size of the blueprint networks of the hall of fame were floating around the media of the minimum_size and maximum_size, it was observed that the networks with a size around minimum_size were producing bad results, the same for networks around maximum_size, so, during the evolution (co-evolution) process, the networks were trying by themselves to find the correct size to produce better results.

5.7 Implementing Competitive Fitness Sharing, Hall of Fame and Sharing Sampling

As it was discussed in the previous chapter Rosin & Belew [1997] proposed some methods for the competitive co-evolution. SANE implements some of these techniques implicitly.

5.7.1 Implementing Hall of Fame

Hall of Fame is implemented in SANE as a set of best blueprints that are kept every generation during the evolution for which are applied the genetic operators as crossover and mutation replacing the worse blueprints of the hall of fame. A blueprint is included in the Hall of Fame based on the fitness that it has obtained in the previous generation. This can be observed in the Figure 5.8.

After the evaluation process has finished, in the production phase of SANE, is calculated the fitness for each blueprint that participated during the competition against blueprint opponents. After the calculation of the fitness all blueprints are ordered based on the fitness value (not making distinction if blueprints belongs of hall of fame). The best blueprints will have a greater value of fitness, the worse blueprints will have lower value of fitness.

In an evolution (competition against a fix player) the fitness value of the blueprints is the score that the blueprint obtained against the opponent player. In the co-evolution the fitness value of the blueprint is calculated by the competitive fitness sharing function, or the competitive fitness sharing augmented (CFSA) proposed in this thesis which will be describe in the next sections.

During the co-evolution two Hall of Fame are created, one for the parasite and the other for the host (in the host-parasite interaction). In both cases the way to add blueprints to the hall of fame is the same.

In SANE the number of members of the hall of fame is fix and should be one third of the members of the sample created from the previous generation. So, every blueprint member of hall of fame is evaluated in every generation, and if in the next generation some blueprints are not out performing with respect to other blueprints (according to the fitness ranking), these blueprints are removed and replaced by best performing blueprints.

5.7.2 Implementing Sharing Sampling

As it was discussed previously, the intention to implement Sharing Sampling is to select the best representation of test cases or parasites from the parasite population. As it described in the Figure 5.8, the selection process of parasites

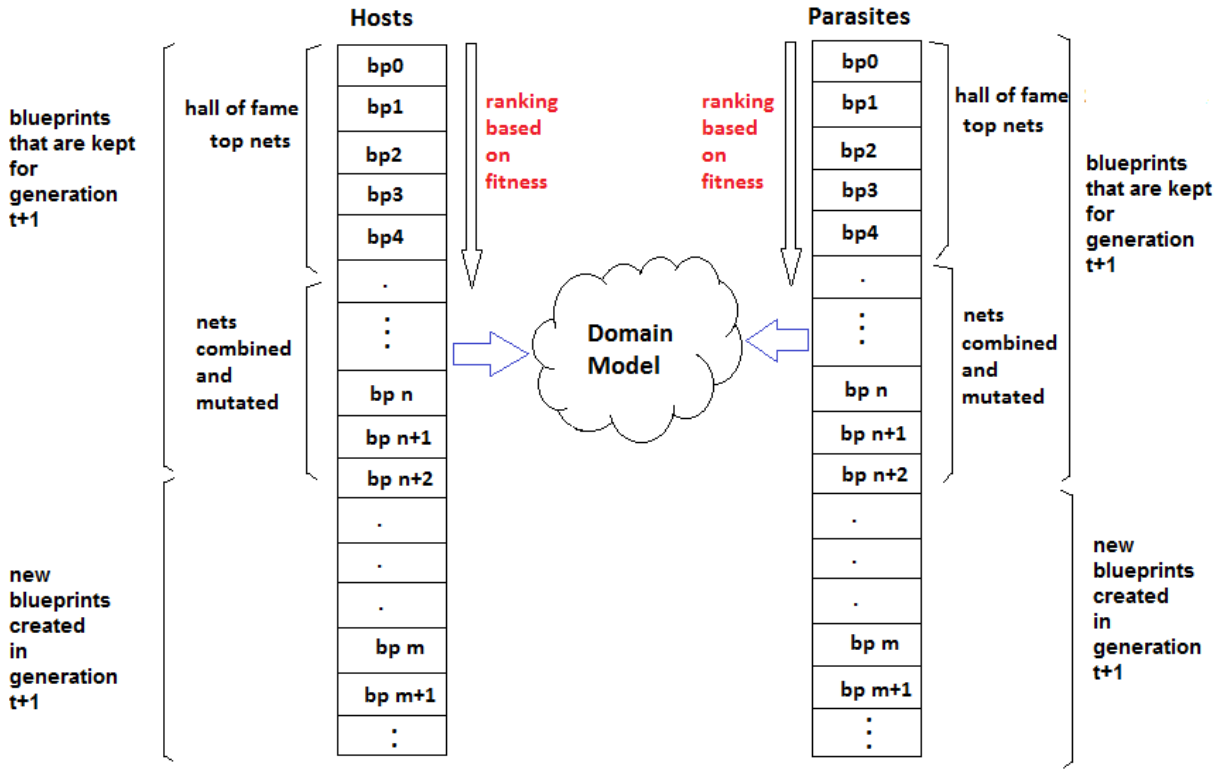


Figure 5.8: Blueprints Hall of Fame and Sampling

use fitness values calculated by a competitive fitness sharing function, which is, selecting the parasites that beat the more (relative) stronger hosts (host that have lost less times) in the previous generation.

In this thesis the implementation of sampling process for the parasite population described by Rosin & Belew [1997], which is shows in the Algorithm 1, it is the same as implemented for host population using competitive fitness sharing to calculate the fitness of each blueprint. For both populations, hosts and parasites, in every generation are selected the more representative set of blueprint populations which it is used a sample test cases for the next generation.

The number of members in the sample in every generation should be at least one half of the population of the blueprints which are competing as is shows in the Figure 5.8.

5.7.3 Implementing Competitive Fitness Sharing Augmented (CFSA)

In the system coded to test the co-evolutionary process was used competitive fitness sharing (CFS) Rosin & Belew [1997] and new proposed method called competitive fitness sharing augmented (CFSA), which is one of the contributions of this thesis. The intention to introduce this fitness function is to introduce more phenotype diversity in the population that are evolving.

As it was described previously, competitive fitness sharing take advantage of the interaction with the opponents and value more if the opponent lost less times than if the opponent lost more frequently only if the game was won by the player, but this technique does not consider the cases when games where lost by player, which should have an effect in the calculation of the fitness. So, CFSA technique consider all the results after the interaction against the opponents, when the game is won or lost by the players.

Rosin & Belew [1997] defines the competitive fitness sharing (CFS) assigned to a player defeating opponents as $\Sigma_{j \in X} \frac{1}{N_j}$ where N_j is the total number of players in the population defeating opponent j, and X the set of interactions in which the player defeat all opponents.

CFSA is defined as the fitness for a player after the interaction against opponents where X is the set of games in which the player won against opponent j and Y the set of games where player lost against opponents i, so, the fitness value is calculated for the player p as

$$CFSA_p = \Sigma_{j \in X} \frac{1}{N_j} - \Sigma_{i \in Y} \frac{1}{N_i} \tag{5.2}$$

Where N_j is the number of times where the opponent j lost games and N_i the

Table 5.1: Results of matchs in the competition

Team 1	Team 2	Results
RM	BC	1
RM	SV	2
RM	AM	1
BC	AM	1
BC	SV	1
SV	AM	X

number of times where the opponent i won games.

The propose of this new fitness function is consider the interactions (or games) where players lost games, and value more negative if the opponent with whom the player lost has won less times against other opponents (weak opponents) and value less negative if the opponent with whom the player lost has won more times (strong opponents). It means, if the player has lost against strong players it will have less impact in its fitness than if the player lost against weak players.

To explain this method be used a football soccer competition in which there are four teams. The team RM, BC, SV and AM, assuming that the teams RM and BC always are the stronger teams in the competition and SV, AM are weak teams which have very low possibility to beat RM or BC.

The incentive to win to stronger teams by the weaker teams are more important if it is used the CFSA than competitive fitness sharing, and at the same time, the stronger teams have more incentive to always beat the weaker teams and avoid some unfair practices as send not the official team when the match is against the weaker teams.

The conditions in which the match happens are: the match is played in a neutral field, It is played only one match against the opponent, and in case the match is drawn both teams receive zero as fitness. In the Table 5.1 can be observed the results, 1 in case the team 1 win, 2 in case team 2 win, and X in case of drawn.

The followings are calculation of the fitness values for each team using CFSA and CFS

Calculation using CFSA:

Fitness RM= $\frac{1}{1}$ (against BC) - $\frac{1}{1}$ (against SV) +$\frac{1}{2}$ (against AM) = 0.5

Fitness BC= $-\frac{1}{2}$ (against RM) + $\frac{1}{1}$ (against SV) + $\frac{1}{2}$ (against AM) = 1

Fitness SV= $\frac{1}{1}$ (against RM) - $\frac{1}{2}$ (against BC) +0 (against AM) = 0.5

Fitness AM= $-\frac{1}{2}$ (against RM) - $\frac{1}{2}$ (against BC) +0 (against SV) = -1.0

Calculation using CFS:

Fitness RM= $\frac{1}{1}$ (against BC) +$\frac{1}{2}$ (against AM) = 1.5

Table 5.2: Results in a Soccer Competition using CFS and CFSA

Team	Fitness using CFS	Fitness using CFSA	Final Position using CFS	Final Position using CFSA
RM	1.5	0.5	1	2
BC	1.5	1.0	1	1
SV	1.0	0.5	3	2
AM	0.0	-1.0	4	4

Table 5.3: Results of the Questions in the Exam of candidates A, B and C

Candidate	Pass	Fail	Not Answered	Final Result CSF	Final Result CSFA
A	6	2	2	12	10
B	6	4	0	12	8
C	5	0	5	10	10

Fitness BC= $\frac{1}{1}$ (against SV) + $\frac{1}{2}$ (against AM) = 1.5
Fitness SV= $\frac{1}{1}$ (against RM) +0 (against AM) = 1
Fitness AM= 0 (against SV) = 0.0

The Table 5.2 summarize the calculation of the fitness values using these techniques and the positions that the team will receive in the competition.

So, as it can be observed in the Table 5.2 depending on the method used the final positions in the competition will be different. In case of using CFS RM and BC (the usual stronger teams) will finish in first positions and SV (the usual weaker team) will finish in third position, but using the CFSA, BC finish in first position (even BC wons two match the same as RM) and RM and SV finish in second position (even SV won only one match, against RM).

So, it was described previously, CFSA give more incentive to strong and weak player to win the game, and avoid some unfair practices of stronger teams to send the match with weaker teams to not official players (weak players) as sometimes is observed in professional football soccer competitions. In this example, SV, a weaker team, beat RM and because of this obtained a better fitness than if can beat another weaker team, and by contrary RM, a stronger team, was penalized because lost against a weaker team, that is why the final positions in the competition are different.

The other example to explain how accurate can be the results of this fitness function is in a examination to enter a university. Imaging the case that the exam to enter a university has 10 questions. The candidates A, B and C has responded according the following table 5.3.

Assuming that every question count 2 points if pass, -1 if fail and 0 if not

answer to the questions, and the calculation of the fitness for simplicity are:

$\sum N_j P_j$, where N_j is the number of questions answered and P_j are the points for the answers (according to table 5.3).

So, according to the Table 5.3, the candidates that can have right to enter the university depend on the fitness evaluation function used. In case of the use of CFS fitness function, who has rights to enter the university is candidate A and B, this is because is not considered the answers that has failed, in other words, CFS method value the same if the answer has failed or was not answered. In case of the use of CFSA fitness function, candidates A and C should enter the university. So, the differences is that CFSA count negatively the wrong answers, which from the author's point of view, this is the correct way value the answers because even candidate A and B have the same knowledge (reply successfully the same number of questions), candidate B is trying to answer something even he does not know the answer, which is an unfair behavior, because in that try, by lucky can guest the answer.

So, in the thesis will be used CFSA because of promote a fair competition and calculate real fitness values of players and not only the positive impact of the interaction with the opponent when players win.

As CFS, CFSA is promoting the phenotype diversity in the population of players and opponents that are interacting, but in case of CFSA considering the positive and negative effect of the interaction against opponents (win and loss games) have a better evaluation of the interactions during competitions. To work with just positive values obtained from CFSA the results from this function should be normalized.

5.8 Co-evolutionary Algorithm for Two Players competition

The other contribution of Zela & Zato [2011] was the introduction of an algorithm that can facilitate the co-evolution of two populations in a host-parasite interaction.

So, the strategy proposed try to solve the deterministic problem identified previously and evolve better strategies for players and opponents populations ensuring a correct co-evolution. In this thesis were introduced some variations to the algorithm introduced in Zela & Zato [2011] because was identified that it was missing some new techniques as CFSA and RIR.

The steps to follow to co-evolve two populations of host and parasites are described below. The steps 1 and 2 are used to initialize isolated the populations of host and parasites, step 3 describes how the populations compete or are evaluated based on the domain model, which is this case the Go game, from the step 4 to

9 describe how the populations of host and parasites are produced and selected for the next generation.

1. Train isolated two populations of neurons for Black (host) and White (parasite) players against an deterministic opponent. The intention of this step is start the co-evolution these two players with "some" knowledge of the domain. In the experiments were used Gnugo and Wally.

2. When the players are starting to beat the opponent or when the populations are good enough trained with good scores, stop the evolution of these two populations. Use the two populations trained isolated to start the co-evolution.

3. In execution of the experiments × M interactions (or games) are performed, where N and M are the number of players and opponents (blueprints) respectively. In the experiments performed in the thesis were used the same values for N and M.

4. When the competition of the N × M games finishes, in the evaluation phase, the evaluation function to calculate the fitness of hosts and parasites are calculated using competitive fitness sharing (CFS) Rosin & Belew [1997], CFSA, or another evaluation function using more information of the game it is explained below.

   For example, in case the fitness function is calculated using CFSA, the host player fitness is calculated using the equation 5.2 as:

   So, the fitness of player or host is $\mathtt{Fitness}_h = \Sigma_{x \in X} \frac{1}{N_x} - \Sigma_{y \in Y} \frac{1}{N_y}$ where N_x is the number of times opponent x lost games if host h beat opponent x and N_y the number of times opponent y won games if h lost against opponent y.

   and fitness of opponent or parasite is $\mathtt{Fitness}_p = \Sigma_{x \in X} \frac{1}{N_x} - \Sigma_{y \in Y} \frac{1}{N_y}$ where N_x is the number of times player x lost games if opponent p beat player x and N_y the number of times opponent y won games if opponent p lost against player y.

   For more details of evaluation function see the section 5.11.

5. Once all the hosts (and parasites) has obtained their fitness, each populations of blueprints (host and parasites) are ranked based in the fitness obtained. For each neuron in the host and parasite population is calculated their fitness based on the fitness of the blueprints networks were the neuron has participated (the same that is used in SANE), and based on the fitness obtained by the neurons, these are ranking based as in case of blueprints.

6. Based on the raking of blueprints and neurons it is created a Hall of Fame for the best blueprints and neurons. For members of Hall of Fame are applied genetic operators as crossover and mutation. The Hall of Fame of blueprints (and neurons) are applied crossover to create new structures of blueprints (and neurons) replacing the less ranked members. The worse blueprints (and neurons) which are not in Hall of Fame are mutated. For simplicity in the experiments the crossover and mutation rate are fixed values, but can be used any function to calculated.

7. When the application of genetic operators to the current populations finishes, it is applied the RIR mechanism, bringing into neuron's population new members replacing the worse neuron population which are not in Hall of Fame.

8. Finally, it is applied the memory mechanism proposed in this thesis, So, it is selected the best blueprint from previous generation based on the fitness obtained in step 4 of each population (for hosts and parasites) and from this is select game that has obtained the best individual fitness (i.e. using CFSA). Based on the moves of this memory (which should be the best strategy for that player according to our assumption) this is reinforced to the neurons of the current population, impacting other blueprints which share these neurons.

9. The steps 3 to 8 is repeated in every generation.

5.9 Mitigation of Co-evolutionary Pathologies Applying the Techniques Proposed

In this section is going to be discussed how the techniques proposed is mitigating the co-evolutionary pathologies that can be observed during the application of co-evolutionary approach.

5.9.1 Mitigation of Loss of Gradients and Disengagement

As it was described, the loss of gradients or disengagement appears when one population is stronger than the population opponent which not facilitate the co-evolutionary process. This can be observed when one population always win against the other populations, or when there is a perfect player that do not lost any game against all the opponents with which he compete.

In this thesis was discussed the replacement immigration rate (RIR) described in the Equation 5.1 which mitigate this disengagement pathology. Even the main

intention of this rate is to replace the worse performing neurons of the neuron population and introduce new members that can contribute with new strategies, this rate is introducing more diversity and avoiding the creation of superiors populations.

In the execution of the experiments that will be described in the next chapter is observed that the use of this rate is important to mitigate this pathology using different values of β of the Equation 5.1, observing that low values of RIR in some executions do not avoid the situation that always one population has more wins that the other population through many generations. A correct co-evolutionary process should allow that sometimes a population A has better results that second population B, but after some generations the second population B has better results that the population A.

5.9.2 Mitigation of Intransivity and cycling dynamics

As it was discussed previously, the cycling dynamics is observed in the co-evolutionary process when the same strategies jump one to another, not promoting the evolution of these strategies. To validate if this dynamics are happening is to monitor the strategies played during the co-evolutionary learning, one way to do it is to analyze if the moves in the memory of the best players introduced in this thesis are cycling.

To mitigate pathology in this thesis there are some techniques introduced in the co-evolutionary process. The first one is to use in the selection and sampling phase CFS and CSFA mechanism (a variant of CFS). CFS mechanism according some authors introduce more diversity to the population and help to reduce the cycling dynamics.

The other mechanism which introduce more diversity in the population is to replacement immigrant rate (RIR) mechanism which introduce more diversity when the population is reaching superior results against the opponents.

The expectation to the results is to observe the red-queen dynamics or an arm-race during the co-evolution of the populations, and at the same time, observing new and better strategies through the generations.

5.9.3 Mitigation of Forgetting

To mitigate the forgetting pathology in this thesis is used two mechanism, first is the use of the hall of fame Rosin & Belew [1997] and the second, use a memory of blueprints mechanism proposed in this thesis.

The hall of fame mechanism is used with the intention to keep the genotypes that obtained better results and ensure that these genotypes are not forget in the future. The best structures of blueprints are maintained and reproduced.

The intention to use the blueprint's memory mechanism is to ensure that some strategies that gave good results in the previous generation are not forget and maintained in the population, actually, this mechanism reinforce these strategies in the genotypes of the population. As it was commented, the result of reinforcing some start-game strategies in Go players were impressive, and the good start-game strategies as start playing in the center of the board was not forget as soon these were discovered during the evolution.

5.10 Generalization and Diversity in Computer Go

As it was discussed in previous sections maintaining the diversity of the genotypes and phenotypes of the population is main approach to keep the solve the co-evolution pathologies described previously. Different authors has proposed different methods which some of them are to slow genetic convergence by halting evolution in a population, the use of multiple and reproductively isolated populations, various types of fitness sharing as competitive fitness sharing and sample sharing Rosin & Belew [1997], selective method of combination, methods to achieve speciation in the population ESP Perez-Bergquist [2001] and others.

This thesis is proposing two methods to keep the genotype diversity of the population, competitive fitness sharing augmented (CFSA) and replacement immigrant rate (RIR). Apart of these methods, SANE by itself use some genetic operators at the level of neurons and blueprints which promote the diversity as well. The other mechanism that is introducing diversity in the co-evolutionary process is the dynamic size of blueprints during the evolution. This mechanism creates diverse structures neural networks while trying to search for solutions to the problem domain. The use of different populations as it was proposed in ESP by Perez-Bergquist [2001] is another option, but Perez-Bergquist considers that is not necessary because of every neuron of the population in SANE is by itself a different solution to the problem.

The other reason to introduce more diversity into the population based on the techniques proposed, is to create more general strategies that can compete and obtain good results against other opponents which was not used during the co-evolution. In practices there are different approaches to measure the generalization, some of them can be test computer player evolved against known computer Go player performing some randoms moves, or test computer players evolved against Go human players (professionals or amateurs). In this thesis the generalization is measure based on the wins against a random set of test cases created based in the generalization measure proposed by Chong *et al.* [2008], Chong *et al.* [2009], which is applying the Chong's generalization framework in

terms of number of wins.

Using the equation 3.8, For this thesis the game outcome of the two strategies are defined as this equation:

$$G_W(i,j) = \begin{cases} 1, & \text{for} \quad g(i,j) > g(j,i) \\ 0, & \text{for otherwise} \end{cases} \tag{5.3}$$

Where $g(j,i)$ and $g(j,i)$ are the scores of the competition of two strategies i and j.

To test how general are the Go players co-evolved during the co-evolution it was created randomly a set of Go players to be set of strategies for testing against the best Go players evolved.

The following is the process used to create the set of N Go players or N test strategies.

1. Create a randomly I Go players or blueprints for White.
   (a) Create the population of neurons randomly for White players.
   (b) Select the neurons to create the blueprints with different sizes.
2. Create a randomly J Go players or blueprints for Black.
   (a) Create the population of neurons randomly for Black players.
   (b) Select the neurons to create the blueprints with different sizes.
3. Compete the strategies of J Black players against strategies of I White players ($I \times J$ interactions).
4. For White players:
   (a) Calculate the $G_W(i,j)$ of the players i after all the interaction according equation 5.3.
   (b) Select and save the White player i that has won more times against Black players j.
5. For Black players:
   (a) Calculate the $G_W(j,i)$ of the players j after all the interaction according equation 5.3.
   (b) Select and save the Black player j that has won more times against white players i
6. Repeat the steps 1 to 5 N times to create N Black and White players or test strategies to be to test the generalization.

So, to measure if the co-evolution process is searching for more general solutions the best player of every generation (White or Black players) are competed against all N test strategies created randomly, high number of times that computer Go players win against this test strategies means more general are the solutions found in the search space. As it was discussed, diversity should introduce more generalization, the measurement of the diversity in this thesis is discussed in the next section.

5.11 Measurement of Genotype Diversity of Neural Networks Evolved

In the section 3.8.1 is discussed how can be measure the genotype diversity of the chromosomes, in this section is going to describe how is measure the genotype diversity in this thesis.

The Figure 5.9 two neurons which could exist in the neuron population that used to evolve and obtain blueprints with better game strategies. In the SANE architecture selected for this thesis, the number of nodes in the input and output is different for every neuron, but the hidden layer contains only one node which connect input and output layer.

Because this structure, should be more common find uncommon genes and more diverse neurons using the definition of section 3.8.1, actually could be difficult to find two similar neurons in the population, so, in this scenario the neuron population should be diverse, so, the intention to measure the diversity in the experiments to measure how diverse are these populations during the evolution and co-evolution.

So, to measure the edit distance for common genes is used the equation 3.14. As can be observed in the Figure 5.9, for the two neurons N_1 and N_2 the common nodes which connect to hidden neuron are the node-1, node-2, node-3 and node-9, these nodes can be referenced as gene-1, gene-2, gene-3 and gene-9 respectively including their weights to use the definition of diversity discussed previously. Apart of that, in SANE architecture, there is not enable and disable status for the genes, meaning that if a particular gene is not there is because that gene does not exist, so, removing this variable, the edit distance for the two common genes is used:

$$d_{com} = \frac{1}{n} \sum_{i=1}^{n} \frac{[|w(g_i)_{N_1}| - |w(g_j)_{N_2}|]}{max[|w(g_i)_{N_1}|, |w(g_j)_{N_2}|]} \tag{5.4}$$

Where N_1 and N_2 are the chromosomes of the two neurons in comparison, $w(g_i)$ and $w(g_j)$ are the weights of common nodes which are connected to the

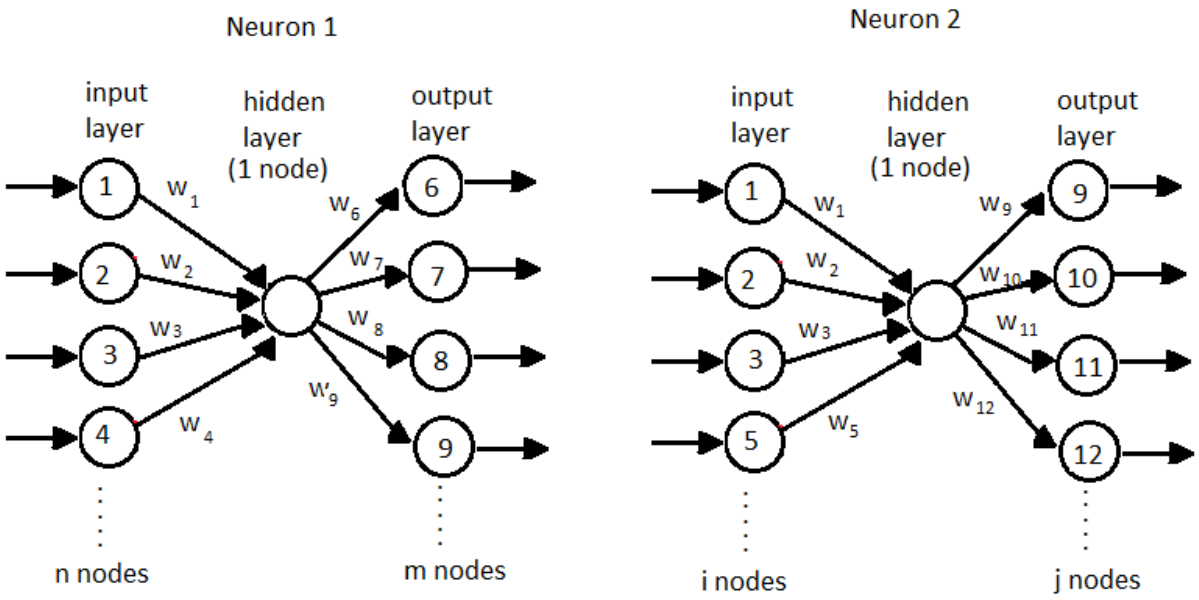


Figure 5.9: Comparison of two neurons with different structures (different number of input and output nodes)

hidden node, and n is the number of common nodes in the neurons compared.

The equation 5.4 describes the distance for which the sign of the weights of the connections are not relevant, in this thesis was implemented an additional distance for which use the sign of the weights is relevant and it is shown in the following equation:

$$d_{com} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n} \frac{[w(g_i)_{N_1} - w(g_j)_{N_2}]}{size}, & \text{if } w(g_i)_{N_1} > w(g_j)_{N_2} \\ \frac{1}{n}\sum_{i=1}^{n} \frac{[w(g_j)_{N_2} - w(g_i)_{N_1}]}{size}, & \text{if } w(g_j)_{N_2} > w(g_i)_{N_1} \end{cases} \quad (5.5)$$

Where N_1 and N_2 are the chromosomes of the two neurons in comparison, $w(g_i)$ and $w(g_j)$ are the weights of common nodes which are connected to the hidden node, n is the number of common nodes in the neurons compared, and $size$ is the max size of $w(g_i) - w(g_j)$, if $w(g_i)$ or $w(g_j) - w(g_i)$, if $w(g_j)$, that can be found in the neurons, this value is always positive.

To measure the edit difference of two uncommon genes are used the equation 3.15. So, as it was discussed previously, in this architecture does not exist enable of disable status of the genes, so, the equation is simplify as is show in the equation

5.6:

$$d_{uncom} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{genes_1} + \frac{1}{m}\sum_{j=1}^{m}\frac{1}{genes_2} \tag{5.6}$$

Where n and m are the number of uncommon genes in the chromosome of neurons, $genes_1$ and $genes_2$ are the total genes from the neurons N_1 and N_2 respectively. In this thesis the neurons has the same size for simplicity, but, should be different length.

Solving this equation, this can be observed in the following equation:

$$d_{uncom} = \frac{2}{s}\sum_{i=1}^{s}\frac{1}{genes_s} \tag{5.7}$$

Where s is the number of uncommon genes for both neuron with similar length. The following condition has to be satisfied: $0 \leq d_{uncom} \leq 2$.

To calculate the edit distance between the chromosomes of neurons N_1 and N_2 can be used the following equation 5.8:

$$d_{NE}(N_1, N_2) = \frac{1}{3}(d_{com} + d_{uncom}) \tag{5.8}$$

Where $0 \leq d_{NE}(N_1, N_2) \leq 1$. So, Finally the genotype diversity of the population is calculated with the following equation 5.9:

$$diversity = \frac{1}{\frac{n(n-1)}{2}}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{NE}(i,j) \tag{5.9}$$

Where n is the total number of neurons in the population, and i, j are the neurons compared.

In the thesis, the nodes are the board positions for input and output layer, and based on the size of the blueprint, the number of hidden nodes can varied. This diversity measurement is implemented and discussed the results in the chapter 6.

5.12 Monitoring the Progress of the Co-evolution of Computer Go players

The intention to monitor the co-evolutionary learning process is the measure the learned strategies during the evolution. As Luke & Wiegand [2003] mentioned, that the correct way to measure the progress of co-evolutions should be to use external progress measure. So, for simplicity in this thesis the approach was to test the best strategies learned during the evolution process against a powerful

opponent. So, the system created will take the best strategies (blueprints) for every generation and compete them against Gnugo (level 10) which is a very strong opponent.

Even the author which have some experience playing Go as amateur has difficulties to beat Gnugo (level 10). The results of this tests are going to be discussed in the next chapter.

According to theory discussed, if the co-evolution is progressing, the best strategies identify later in the evolution should be able to beat earlier strategies. To validate this theory, the other experiment that will be performed are to test the best player of the last generation against the best strategy of the previous generation, but as Ficici mentioned, this is not a reliable condition to identify that co-evolution is really happening.

There is another approach, which is to take the best players for every generation and try to classified them according the Go player levels as kyu, dan after some competitions against some Go human players. This is a more expensive approach, but at the end should be the correct way to value if techniques proposed have been created more general strategies with the capacity to compete against more diverse human Go players.

5.13 Evaluation Functions in Computer Go

As it was discussed, maybe this is the more complex part when is created algorithms for evolutionary or co-evolutionary learning. During the execution of the experiments it was noted that a correct evaluation functions is key issue to allow that the player evolves. A wrong evaluation function can mislead the learning process. In the next sections is described the evaluation function used in this Thesis.

There are two scoring method used widely in Go game, the Chinese and Japanese scoring methods. In the Chinese scoring method the score is determined by the number of the empty intersections completely surrounded by own stone players and the number of alive stones in the board. In the Japanese scoring method the score is determined by counting the captured enemy stones and all empty intersections that are completely surrounded by only stones of that player.

In the programs developed was implemented these two scoring methods, but as because Chinese scoring is more widely used, if was selected for all the experiments. The selection of the scoring methods impact the strategics learned during the evolution process, meaning that one player evolved with Chinese scoring method should not be used in a competition using a Japanese scoring method.

5.13.1 Evaluation Function used in Evolution

In a evolutionary learning process the evaluation function selected was based on the scores that players has obtained after a competition against the opponent.

So, after every competition, the players were evaluated according to the following:

- Calculate the score of every player after the competition.
- If the player has not lost (won or at least drawn) assign the score to the fitness of the layer.
- In other cases (lost), assign zero to the fitness.

5.13.2 Evaluation Function used in Co-evolution

In this thesis was used the host-parasite interaction to represent the co-evolutionary process in computer Go game. It was selected as Host population as Black players and Parasite population as White players.

It was used different fitness functions, considering some features of the game as average score from all competitions or the number of wins that a player obtain during the competitions. So, the following fitness functions was used to calculate the fitness of the player (host or parasite) $\mathsf{Fitness}_p$:

$$\mathsf{Fitness}_p = \chi * \mathsf{cf} + \alpha * \frac{wins_p}{\mathsf{trials}_p} \tag{5.10}$$

Where χ and α are the parameters used during the evolution, and should accomplish this condition: $\chi + \alpha \leq 1$. cf is the competitive fitness sharing used which can be CFS from equation 3.18 or CFSA from equation 5.2. $wins$ is the number of wins that the player has obtained in the competition against the opponent, and trials is the total number of competition of the player

$$\mathsf{Fitness}_p = \chi * \mathsf{cf} + \alpha * \frac{\sum^{\mathsf{trials}_p} \mathsf{score}_p}{\mathsf{ms}_p * \mathsf{trials}_p} \tag{5.11}$$

Where score_p is the score obtained by the player during the competition, and ms_p is the maximum score in the game. In a 9×9 Go board the maximum score is 81 using Chinese scoring.

The parameters χ and α are between 0 and 1, and in some experiments the values are selected manually (0.5 and 0.5) but in other experiments are changing automatically during the evolution. The parameter χ is giving more importance to the competitive fitness used and α is giving more importance to the scores or

the number of wins during the competition. The importance of these parameters are discussed in the chapter 6.

In some experiments were used just CFSA and CFS as fitness function to test how diversity and generalization is introduced by this fitness sharing functions.

So, in a co-evolutionary learning process the evaluation function selected was calculated in the following process:

- Compete all host players (M) against all parasite players (N) in a M $\times$N competitions (For simplicity was selected M = N)
- Calculate the score of every player (host or parasite) after the competitions
- If the Host (black) $\mathtt{h}$ has won the game against a Parasite $\mathtt{p}$:
  - Calculate the Fitness function for each host player using the equation described above: 5.10, or 5.11, or 3.13 (CFS), or 5.4 (CFSA).
- If the Parasite (white) $\mathtt{p}$ has won the game against Host $\mathtt{h}$:
  - Calculate the Fitness function for each parasite player using the equation described above: 5.10, or 5.11, or 3.13 (CFS), or 5.4 (CFSA).

5.13.2.1 Fitness Functions considering the Fitness of Previous Generations

In this thesis is introduced another way to calculate the fitness functions of players after competitions. The previous fitness functions, described above, considers the results of competitions in the current generation and not the results from the previous generations. It was observed in some experiments using co-evolution, as per selection and genetic operators pressure, that some good strategies discovered in specific generation can be lost in the future.

So, the evaluation function should consider previous results as well. The blueprints which belongs to Hall of Fame, or the players with the best strategies, are moving out of this selected group not just based on the results of the competition in the current generation, it is considering as well the results of the previous generations.

In case of the blueprints which not belong to Hall of Fame, should get a very good results to be consider part of this selected group. So, the intention to implement this selection strategy is to keep these good strategies for long time (more generations) and give more time to spread these strategies to other

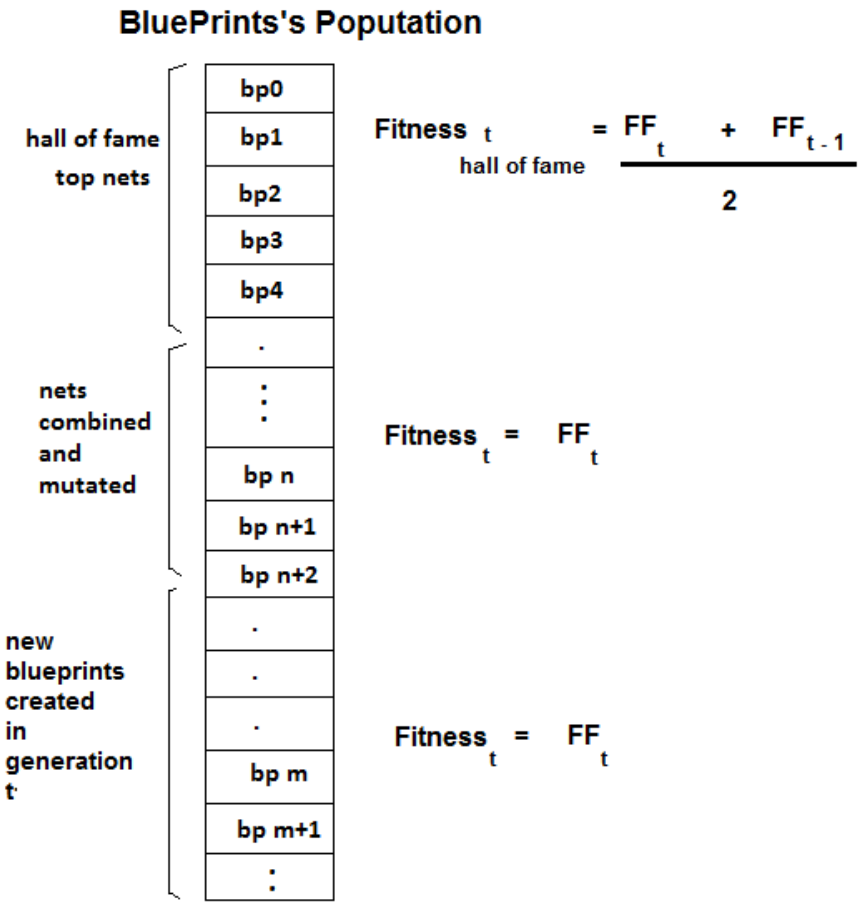


Figure 5.10: Calculation of Fitness Functions (FF) for the blueprints of Hall of Fame and other blueprints

blueprints of the population.So, the evaluation function for members of Hall of Fame is the following:

$$\text{Fitness}_t = \frac{\text{Fitness}_t + \text{Fitness}_{t-1}}{2} \tag{5.12}$$

This can be observed in the Figure 5.10. In the next chapter is discussed the experiments performed using the techniques discussed and proposed by the author in this chapter.

Chapter 6

Application of Co-evolutionary Techniques Proposed in Computer Go Players

In this chapter is presented the results of the experiments performed of evolving and co-evolving Computer Go players in 9×9 Go boards applying the techniques discussed previously.

The chapter starts with the description of the architecture implemented and the results of experiments performed in the evolution of a Computer Go player against Wally. It describes the results of the evolution of black and white population players, analyze the impact of techniques proposed in the evolution of these players using computer programs coded by the author.

In the section 6.2 is described the architecture implemented for the co-evolution of two computer Go players. In the section 6.3 is analyzed the results and main issues observed in the evolution against a known computer Go players.

In the Section 6.4 is described the experiments performed for the co-evolution of two population of computer Go players, discussed about different fitness functions used and the results obtained, analyzed the diversity of the population of neurons obtained during the co-evolution, and measurement of the generalization of the strategies learned in these experiments.

Finally, the chapter ends with the discussion about the global fitness obtained during the co-evolution using an external agent or known computer Go players as Wally and Gnugo, and about the pathologies observed during the experiments.

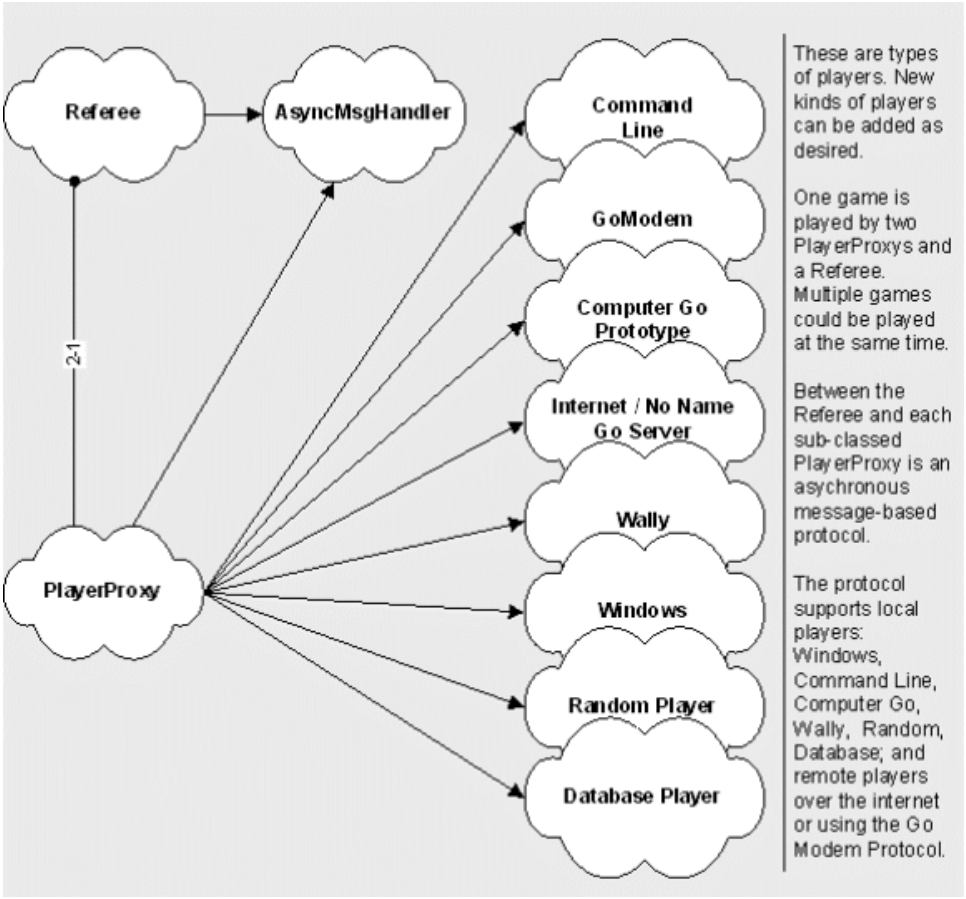


Figure 6.1: Classes in the OpenGo system

6.1 Description of the Architecture using OpenGo

In this thesis to perform the experiments was used the OpenGo system which was available free in Opengo and some routines in C language used in SANE method from SANE-C. The programs for OpenGo are coded in C and available for Windows and Linux operating systems. The Figure 6.1 show the classes available in the OpenGo system. This system was used in the evolutionary and co-evolutionary learning processes performed as a referee for players coded by the author.

As it can be observed in the Figure 6.1, there are some classes that represent the referee, which provide an interface between the classes of players (PlayerProxy class), GameDisplay class which provide a view of the game from the referee's perspective, and DataBoard class which provide information of the game and board. The Referee class create and uses the PlayerProxy classes. There are

only two PlayerProxys at any one time, one for each opponent (black or white players).

When a PlayerProxy class is created, every move from the players is passed to the referee using this PlayerProxy class functions. The game is played asynchronously using difference of sources as players over the Internet. In the Figure 6.1 can be observed some players as Wally, Random Player (which are player which play random moves), Gomodem which is player that support the Go Modem Protocol which is used by many computer Go player in Go competitions. These players are coded as shared libraries in linux (extension .so) and dinamyc load library in windows environment (extension *.dll). For more information of the OpenGo programming environment can be referred at Opengo.

In the experiments were used Wally, a known computer Go player, and Gomodem player as an interface to GnuGo v.3.8, another known computer Go player, and coded two more players by the author which implemented all the techniques discussed in this thesis, these players were called NicoGoW and NicoGoB for white and black stones players respectively. From now till the rest of this thesis, these are called White and Black players.

NicoGoW and NicoGoB were coded in C++ and can be executed in Linux and Windows. In Linux was used for running evolution and co-evolution experiments of population of neurons and blueprints (as .so file), and in Windows (as .dll file) to test the players evolved against another computer Go player or against a human player to observe the strategies evolved.

The structure of these computer Go players and configuration implemented are shown in the Figure 6.2. As it can be see in the Figure 6.2, NicoGoB and NicoGoW has two main parts during the evolution and co-evolution: Evaluation and Production. The evaluation part of these programs coded are based on SANE method:

1. Create a random population of blueprints based on neuron's population (with dynamic sizing or different number of neurons as it was explained in the previous chapter).

2. Compete every blueprint against the opponents (i.e Wally, GnuGo,NicoGoW or NicoGoB).

3. For each blueprint in the population calculate the fitness (i.e score after the competition against opponent) based on the evaluation functions selected and discussed in section 5.13.

4. For each neuron which participate in the blueprint assign the fitness of the blueprint.

5. Divide the fitness of each neuron between the number of blueprints in which have participated.

6. Assign the best blueprints (with best fitness) to Hall of Fame for the next generation and create new blueprints randomly from the neuron population.

7. Repeat step from 2 to 6 till end of the generations or reach maximum fitness possible (i.e. when score of the players reach score 81 in a 9×9 board) .

The Production part implemented in the programs coded are performed after the evaluation part finishes (i.e. when finish all games in the current generation). It contains the following steps:

1. Sort all the blueprints based on the fitness calculated in the competition (create a ranking per neuron).

2. Sort all neurons in the population based on the fitness calculated in the competition (create a ranking per blueprint).

3. The best neurons (based on the ranking) are combined based on crossover rate to replace the worse neurons in the population (i.e. the best half of the population are combined to replace the worse half of the population). This is a sexual reproduction for which is needed two parents.

4. The worse neurons of the population are mutated based on the mutation rate.

5. Replace the worse neurons with new neurons based on the RIR (replacement immigration rate) mechanism. This rate is dynamically calculated as it was discussed in chapter 5.4.

6. The best blueprints (based on the ranking) are combined base on crossover rate to replace the worse blueprints of the population.

7. The worse blueprint are mutated (i.e. replacing some genes of the neurons that belong to the blueprint).

8. Reinforce the best strategies discovered in the competition as it was discussed in chapter 5.5 (adjust the weights of the output genes (moves) that are in the memory of the blueprint).

9. Save the new population of neurons in the neuron population for the next generation.

10. Update the Hall of fame of blueprints for the next generation.

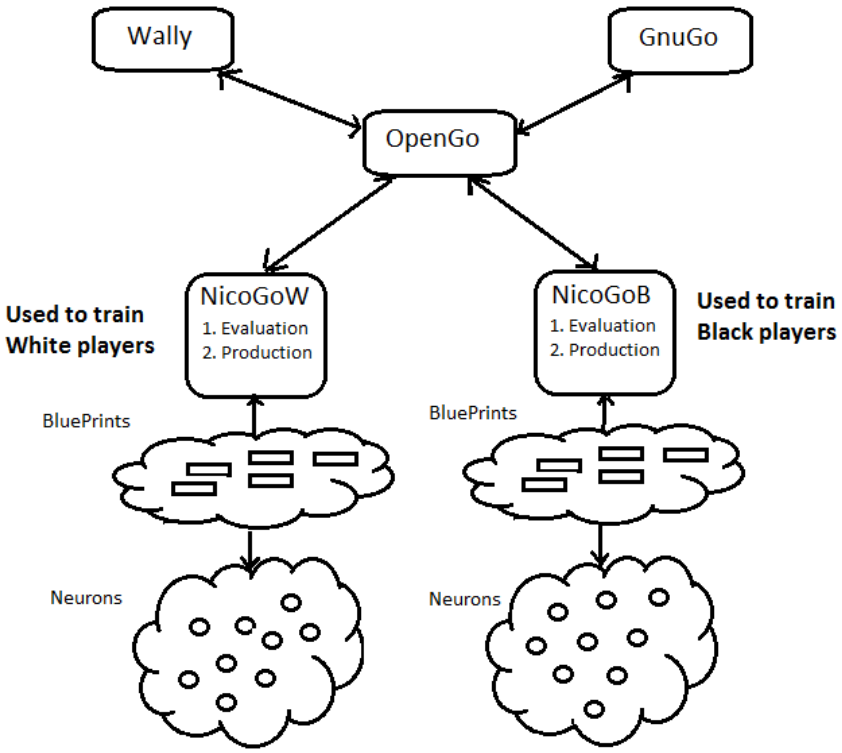


Figure 6.2: Architecture of the Environment used for Experiments in Evolution and Co-evolution

These two parts, Evaluation and Production, were coded in NicoGoW and NicoGoB to evolve neuron populations. Some source code is presented in the Appendix B for reference. In the experiments were used neuron population randomly created. For the co-evolution experiments were used a population previously trained.

In the Windows version of NicoGoW and NicoGoB, these programs were coded to read the best blueprints saved of every generation during the evolution. These blueprints saved are players evolved which have obtained the best fitness compared with other players in each generation. In every generation there is a blueprint saved for NicoGoW, the White player, and NicoGoB, for a Black player.

6.2 Setup of the Experiments in a Go game

During the experiments were used different parameter's values, but, finally it was selected the followings because it produced better results. So, for the evo-

lution and co-evolution of computer Go players against were used the following parameters:

- Neurons: The population size of neurons is 2000. The size of genes of every neuron was 100 (which intent to represent 5 intersection in the input layer and 5 intersections in the output layer).
- Blueprints: The population size of the blueprints is 100 per generation. The Hall of Fame contains 62 blueprints.
- Crossover rate: 50% . Which means that from the best population (i.e. half of the population) 50% of that population are combined.
- Mutation rate: 3%. Which means that 3% of the worse population (i.e. half of the population) are mutated.
- The RIR rate is calculate based in equation 5.1: $\mathsf{RIR} = \frac{\beta}{e^{\frac{-\mathsf{GNL}}{\mathsf{TG}}}}$ The β parameter used was 2.0, but finally it was used 3.0 as it will discussed later.
- The range of sizes for the blueprints created or number of neurons per blueprint were in these range: [24, 100]. Blueprints with size out of this range were not allowed as it was discussed in the chapter 5.6.

The Figure 6.3 shows the Go board positions that were used for the experiments. It starts in the corner 1-A with the position 0, 1-B with the position 1 and 1-C with the position 2 and so on. This nomenclature is used in the next sections when are mentioned the moves played in the board.

So, for these experiments the border positions are the followings: { 0, 1, 2, 3, 4, 5, 6, 7, 8}, { 72, 73, 74, 75, 76, 77, 78, 79, 80}, { 0, 9, 18, 27, 36, 45, 54, 63, 72 } and { 8, 17, 26, 35, 44, 53, 62, 71, 80} .

For the center positions can be considered the corners inside the box that is created by the corners { 20, 24, 56, 60 } including the corner formed by this box. The center of the board is position 40.

6.3 Evolving Computer Go players

In the thesis has been discussed the co-evolutionary learning processes, the pathologies and other issues that are faced when are applied co-evolutionary techniques. The experiments of this thesis start with the evolution of computer Go players against Wally with the intention to evaluate the techniques discussed in this thesis in an evolutionary process against a deterministic player.

In this thesis was introduced new techniques proposed by the author as RIR (replacement immigration rate), CFSA (competitive fitness sharing augmented),

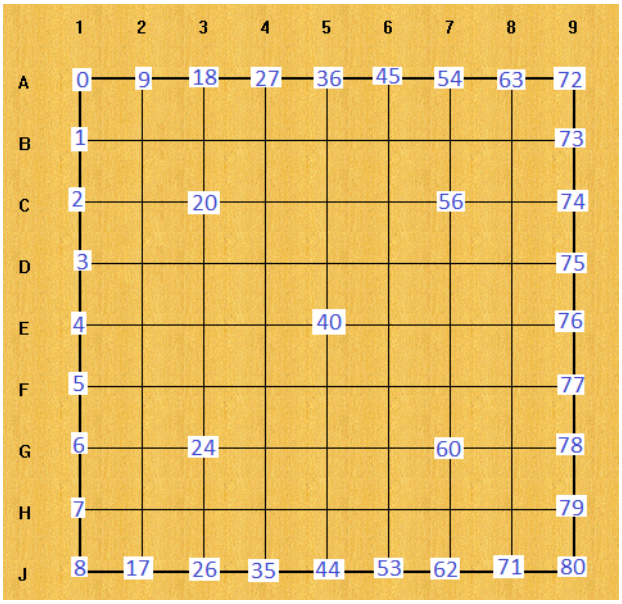


Figure 6.3: Go board positions used in the experiments (for input and output layer)

dynamic size of the blueprint players, memory of the best strategies of the blueprints and other techniques. So, the intention is to observe how this techniques impact in an evolutionary process before are applied in a co-evolutionary environment.

6.3.1 Evolving Computer Go player against Wally in 9x9 Board

In these experiments were evolved computer Go players against Wally. The computer Go players used for the evolution of the population of neurons and blueprints were called NicoGoB for Black players and NicoGoW for White players. Every population of neurons were created randomly initially and evolved against Wally separately. For all experiments were not used handicap neither komi (komi=0), it was used the Chinese scoring method and it was not allowed the suicide moves.

The firsts experiments performed had the intention to evaluate the introduction of some techniques as the memory of blueprints evolved. The Figure 6.4 and Figure 6.5 shows one of the executions for the evolution of NicoGoB and NicoGoW against Wally using non-memory blueprints. In both cases, the initial population of neurons for NicoGoB and NicoGoW were created randomly.

In case of evolution of population of neurons and blueprints using NicoGoB

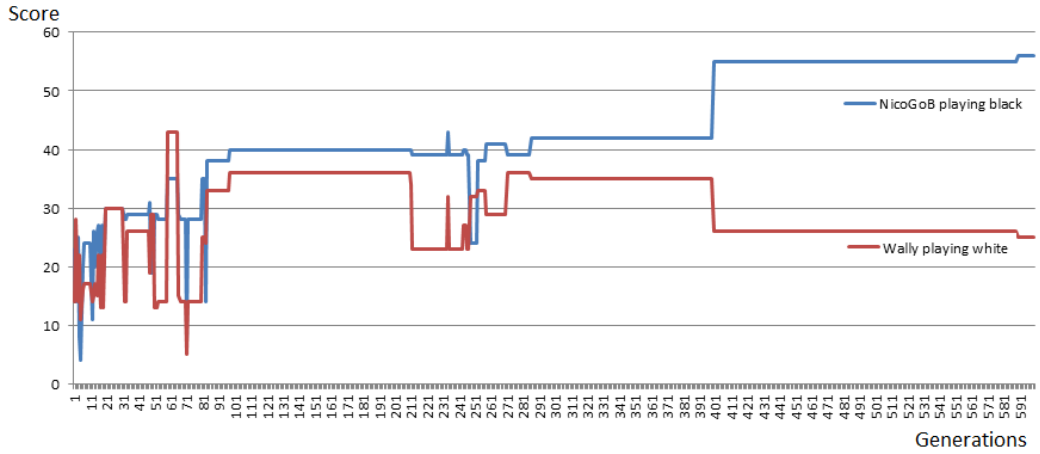


Figure 6.4: Scores of best player NicoGoB (black) and Wally (white) during evolution not using blueprint memory

against Wally, can be observed in the Figure 6.4 that from the generation 83 NicoGoB start to beat Wally more frequently with scores of 38 against Wally with score of 33. From the generation 212 till 242 the difference in the scores are more bigger. From the generation 400 till 600 NicoGoB beat Wally with a greater score: NicoGoB: 55 vs. Wally: 26.

In the Figure 6.5 can be observed one of the executions of the evolution of population of neurons and blueprints using NicoGoW against Wally. In the generation 5 NicoGoW starts beating Wally with the scores: NicoGoW: 36 and Wally:27, and from generation 30 NicoGoW:50 and Wally:26 till the generation 1000. Even NicoGoW starts beating Wally early the difference in the score is not as bigger as in case of NicoGoB vs. Wally. Some of the reasons could be that NicoGoB start playing first against Wally (because of black plays first) which give some advantage and NicoGoW which always plays White is not using Komi (Komi=0).

The Figure 6.6 shows the evolution by number of wins of NicoGoB playing black against Wally playing white using the memory mechanism. It shows that from the generation 50 the number of wins of NicoGoB against Wally it is around 30 games of 100 games per generation. There are some disruptions in the generation 151 and 268 were the number of wins of NicoGoB decrease till 15 and 7 respectively. One of the reasons to this behavior could be that in the previous generations the RIR rate reached peak numbers (values near to 3% of replacement immigration rate) which introduced more diversity in the population impacting the current blueprint structures and performance. In these experiments were used β equal to 2.0. But even these disruptions in some generations, the number of

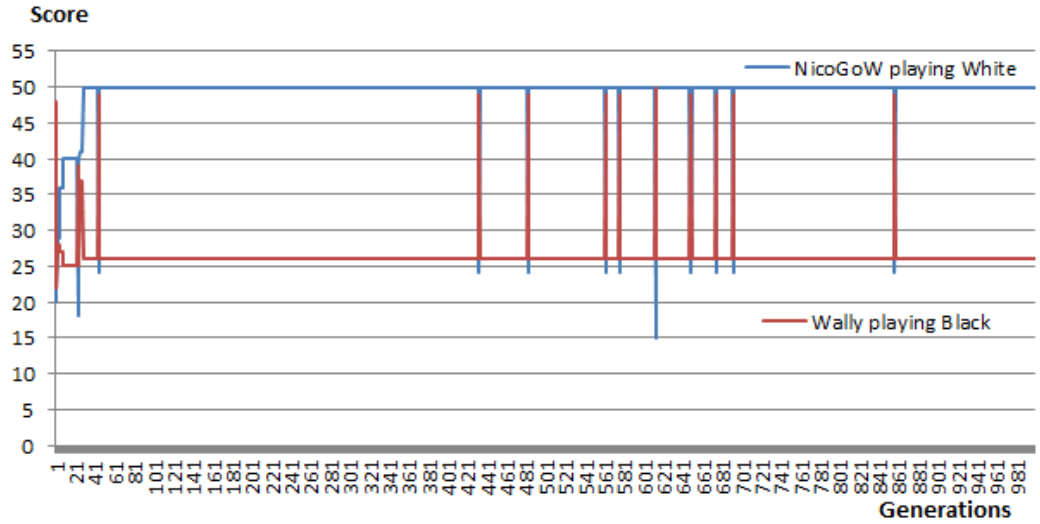


Figure 6.5: Scores of Wally (black) and best player NicoGoW (white) during evolution not using blueprint memory

games won by NicoGoB was around 30 in most of the generations.

The Figure 6.7 shows the evolution of the score of the best player using NicoGoB against Wally in one of the executions. It shows that from the generation 148 the best player of NicoGoB starts beating Wally with the score 81:0.

The Figure 6.8 shows one of the game in which the best players of NicoGoB beat Wally 81:0. The white stones in the position 26 (3-J), 35(4-J), 44(5-J) and 53(6-J) are dead stones. This demonstrates that this configuration can create best strategies to beat Wally 81:0. As it will be discussed later, in both cases, using and not using memory, NicoGoB in some executions beat Wally 81:0. One of the reasons is because Wally is not necessarily a strong computer Go player.

The Figure 6.9 shows one of the evolution of computer Go players of NicoGoW, playing white, against Wally, playing black, using the memory mechanism. It shows that from the generation 241 till generation 425 the number of wins of players of NicoGoW against Wally is around 40 games from 100 games per generation (even much better than NicoGoB). There are some disruptions in the generation 428, 680 and 956 were the number of wins of NicoGoW decrease till 14, 12 and 6 respectively. One of the reason to this behavior could be that the previous generations the RIR rate reached peak numbers (values more than 3% of replacement immigration rate). As in the previous experiments β was equal to 2.0.

The Figure 6.10 shows the evolution of the score of the best player of NicoGoW against Wally in one of the executions performed. It shows that from the generation 51 NicoGoW starts to beat to Wally with the score 48:31 and the

Figure 6.6: Evolution Wally vs. NicoGoB with 100 Trials per Generation - using blueprint memory

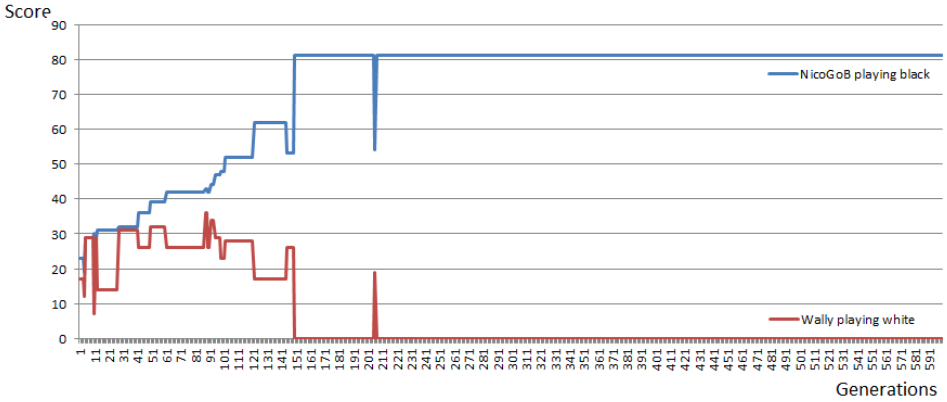


Figure 6.7: Scores of best player NicoGoB (black) and Wally (white) during evolution - using blueprint memory

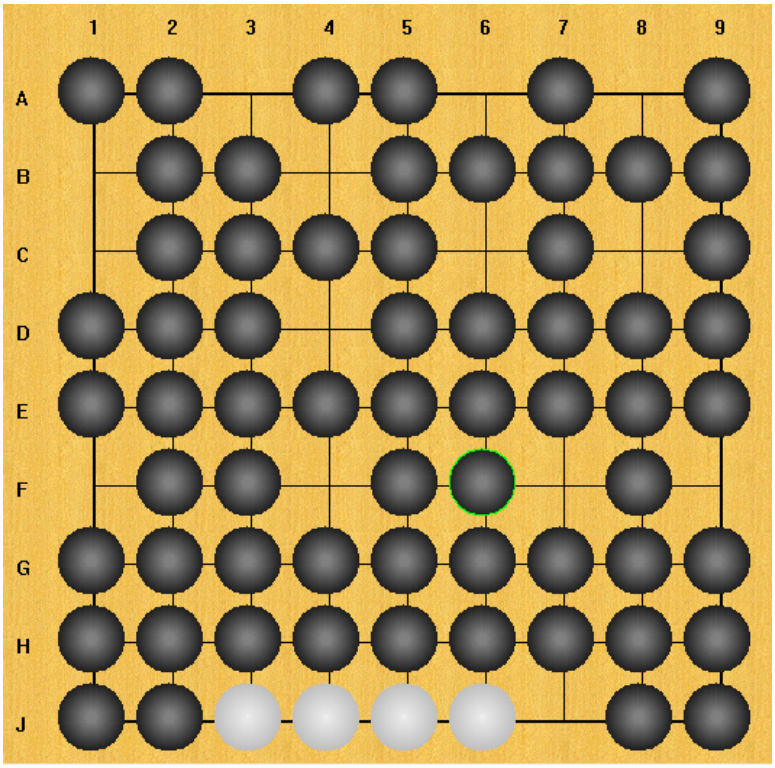


Figure 6.8: Game board of NicoGoB (black) vs. Wally (white) - using blueprint memory

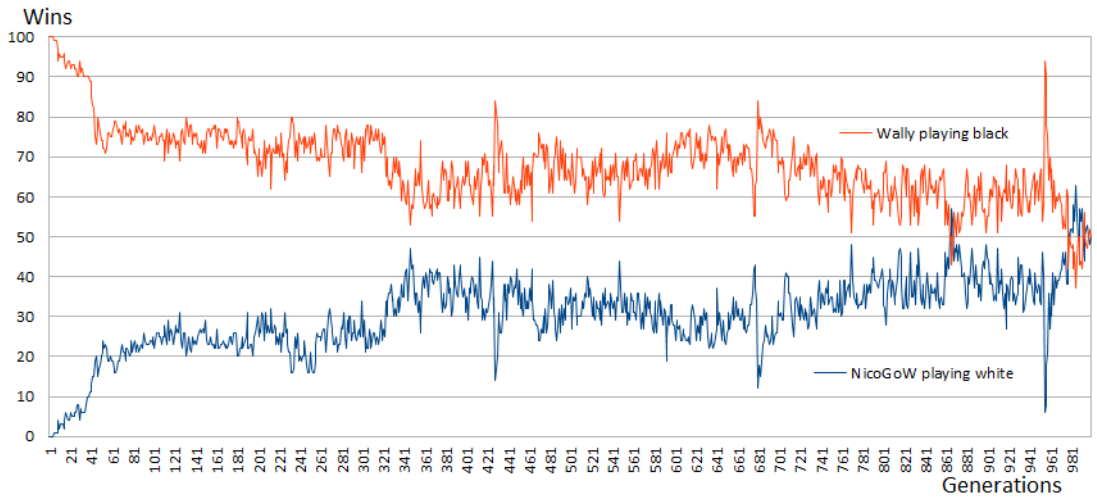


Figure 6.9: Evolution Wally (black) vs. NicoGoB (white) with 100 trials per generation - using blueprint memory

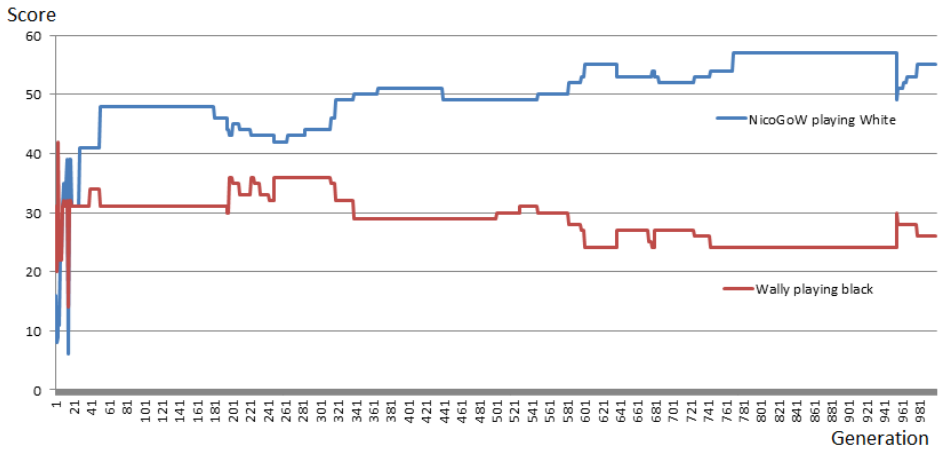


Figure 6.10: Scores of Wally (black) and best player of NicoGoW (white) during evolution - using memory mechanism

difference in the score start to increase from there as it can be show in the figure.

In the Figure 6.11 can be observed the game of one computer Go player evolved using NicoGoW in which beat Wally 26:52 (komi=0). The black stones 1(1-B), 10(2-B), 18(3-A) and 19(3-B) are dead stones, so, can be removed from the board.

The Figure 6.12 shows the comparison of best scores of players evolved by NicoGoB, playing black, using and not using blueprint memory mechanism against Wally, playing white, in 10 different executions (for each memory and not memory) starting with the same initial random population. As it can be observed in the figure, in both cases the best scores of NicoGoB against Wally growth, in some cases beating Wally 81:0 as it was showed in Figure 6.7. So, this indicate that in term of learning both configurations are good enough to discover in every generation better strategies. The executions were called for NicoGoB using memory mechanism BM0 till BM9 and for NicoGoB not using memory mechanism BNM0 till BNM9 as can be observed in the following tables 6.1, 6.3, and 6.4.

The Figure 6.13 shows the best scores of NicoGoW player using and not using blueprint memory against Wally (black) in 10 different executions (for each memory and not memory) starting with the same initial random population. As it can be observed in the figure, in both cases the best scores of NicoGoW against Wally growth, but in case of NicoGoW player, the beats never reached 81:0 as in case of NicoGoB. So, as in previous case, NicoGoW discover in every new generation new strategies even using or not using memory mechanism. The executions were called for NicoGoW using memory mechanism WM0 till WM9 and for NicoGoW not using memory mechanism WNM0 till WNM9 as can be

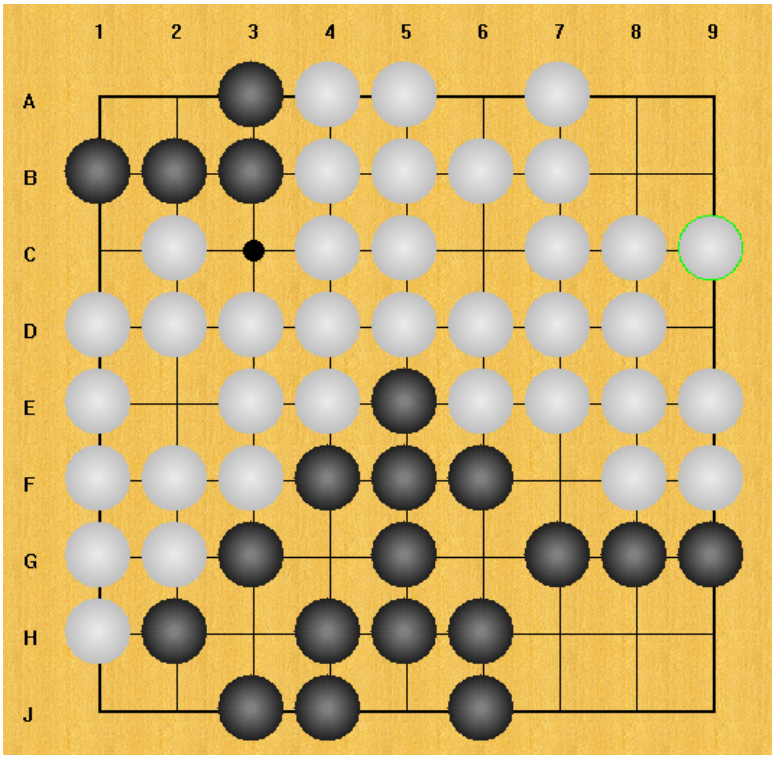


Figure 6.11: Game board of Wally (black) vs. NicoGoW (white) - using memory mechanism

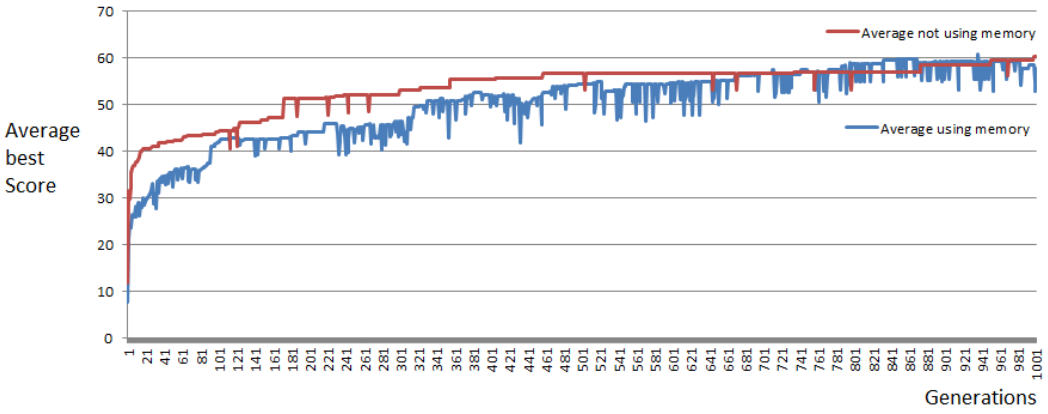


Figure 6.12: Average scores of best scores of player NicoGoB (black) of 10 different executions using and not using memory

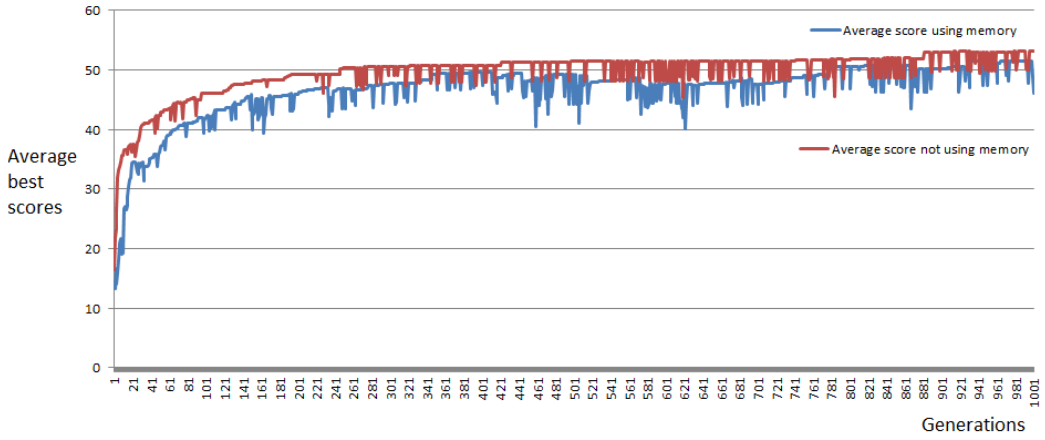


Figure 6.13: Average scores of best scores of player NicoGoW (white) of 10 different executions using and not using memory

observed in the following tables 6.2, 6.5 and 6.6.

In the Table 6.1 can be observed the first seven positions that were reinforced by NicoGoB playing black player in 10 different executions using the same initial random populations as it was discussed in section 5.5. The position reinforced during the evolution against wally are not center positions in the board in some cases. For example execution BM0 (Black Memory execution 0) reinforce positions 13, 14, 15, 42, 45, 25 (in that order) which can be considered not border positions per the previous description. Probably one "good" start-game strategy starting in center in the board is the execution BM8 for the the first seven moves are 14, 23, 44, 78, 13, 15.

The Table 6.2 shows the first seven positions reinforced using the memory mechanism by NicoGoW. As in case of NicoGoB, in some executions it is reinforced some centered positions as in case of WM5 for positions 24, 40 42, 67, 13, 71, 43.

The Table 6.3 shows the first moves by NicoGoB using memory mechanism. As it can be observed, the moves are related to the positions reinforced presented in the table 6.1. For example, the positions reinforced in the execution BM0, are similar to the moves performed by NicoGoB during that execution. In this table can be observed as well the scores and sizes for the best blueprint players at the generation 1000. For example the execution BM5 and BM9 starts with not center positions, but even that NicoGoB beat Wally 81:0. Wally is using Komi=0. Probably the best start-game strategy is the execution BM3.

The size of the blueprint with perfect scores against wally at the generation 1000 was observed that vary, for example in the execution BM5 the size of the perfect strategy has a blueprint with 91 neurons and in the execution BM9 the

Table 6.1: Firsts positions reinforced using the blueprint memory mechanism by NicoGoB

Execution	p1	p2	p3	p4	p5	p6	p7
BM0	13	14	15	42	45	25	61
BM1	26	29	53	67	55	76	15
BM2	9	15	21	29	10	68	55
BM3	25	49	56	14	42	78	12
BM4	2	27	37	48	21	38	70
BM5	2	46	58	70	14	62	4
BM6	9	27	62	60	66	70	61
BM7	13	63	65	52	11	74	43
BM8	14	23	44	78	13	15	18
BM9	10	12	75	19	20	46	40

Table 6.2: Firsts positions reinforced using the blueprint memory mechanism by NicoGoW

Execution	p1	p2	p3	p4	p5	p6	p7
WM0	2	16	63	38	44	46	76
WM1	22	34	36	73	58	16	52
WM2	3	34	66	75	28	19	57
WM3	5	56	74	75	12	34	48
WM4	7	28	75	3	46	34	30
WM5	24	40	42	67	13	71	43
WM6	32	33	21	3	20	7	1
WM7	29	46	57	71	55	10	22
WM8	2	67	70	20	43	76	38
WM9	43	44	48	74	56	68	59

Table 6.3: First moves by NicoGoB using blueprint memory mechanism

Execution	m1	m2	m3	m4	m5	m6	m7	Score	Size
BM0	13	14	15	42	45	25	61	53.00	33
BM1	26	29	53	67	55	76	15	42.00	50
BM2	9	15	21	29	10	68	55	54.00	77
BM3	25	49	56	14	42	78	12	49.00	42
BM4	2	27	37	48	21	38	70	39.00	96
BM5	2	46	58	70	14	62	4	81.00	91
BM6	27	9	62	60	66	70	61	50.00	74
BM7	13	63	65	52	11	74	43	61.00	90
BM8	14	23	44	78	13	15	18	67.00	63
BM9	10	12	75	19	20	46	40	81.00	54

size of the blueprint with the perfect strategy was 54. So, in terms of efficiency can be obtained the same perfect strategy using less number of neurons, so, can be concluded that the use of dynamic sizing of blueprint could find better and more efficient networks structures.

The Table 6.4 shows the first moves not using the memory mechanism. It can be observed that for all of these executions NicoGoB start playing with the same moves 13, 14, 15, 42, 45. The scores and the size of the best blueprints at generation 1000 for each of these executions can be observed in the column score and size respectively.

In the execution with not blueprint memory mechanism it was observed three perfect strategies against wally (execution BNM4, BNM5, BNM9) even the start-game strategy are not necessarily the best starts. In terms of the size, the perfect blueprint strategies have different sizes, for example the blueprint in the execution BNM4 has 86 neurons, in BNM5 has 99 neurons and in BNM9 has 81 neurons.

In the table 6.5 can be observed the best strategies by NicoGoW at the generation 1000 using memory mechanism. As in the case of the NicoGoB, the start-game strategies reinforced are different, in some cases starting in the center of the board as in case of the execution WM5: 24, 40, 42, 67, 13, 71.

Comparing to the NicoGoB,in NicoGoW using memory mechanism there are not perfect strategies. The best score at the generation 1000 against Wally is the execution WM5 where NicoGoW obtained 56 points. As in previous cases, for the executions of NicoGoW was used Komi=0.

In the table 6.6 can be observed the results of the execution of NicoGoW not using the memory mechanism. As in case of NicoGoW using memory mechanism there is not a perfect strategy against Wally, the best score at generation 1000 by NicoGoW is in the execution WNM2 where NicoGoW obtained 63 points.

Table 6.4: First moves by NicoGoB Not using blueprint memory mechanism

Execution	m1	m2	m3	m4	m5	m6	m7	Score	Size
BNM0	13	14	15	42	45	51	25	48.00	52
BNM1	13	14	15	42	45	2	61	57.00	52
BNM2	13	14	42	15	45	51	25	50.00	49
BNM3	13	15	42	14	25	45	71	43.00	58
BNM4	13	14	15	42	45	51	61	81.00	86
BNM5	13	14	42	15	45	51	32	81.00	99
BNM6	13	14	15	42	8	25	33	60.00	71
BNM7	14	15	45	42	76	51	55	54.00	41
BNM8	13	14	42	15	45	25	40	47.00	28
BNM9	13	14	15	42	45	25	7	81.00	81

Table 6.5: First moves by NicoGoW using blueprint memory mechanism

Execution	m1	m2	m3	m4	m5	m6	m7	Score	Size
WM0	2	16	63	38	44	46	76	50.00	71
WM1	22	34	36	73	58	16	40	47.00	68
WM2	3	34	66	75	28	19	57	53.00	49
WM3	5	56	74	75	12	34	3	49.00	53
WM4	7	28	75	3	46	34	30	52.00	75
WM5	24	40	42	67	13	71	43	56.00	81
WM6	32	33	21	3	20	7	1	54.00	54
WM7	29	46	57	71	10	22	55	51.00	69
WM8	2	67	20	43	76	38	68	48.00	89
WM9	43	44	48	74	56	68	43	54.00	59

Table 6.6: First moves by NicoGoW Not using blueprint memory mechanism

Execution	m1	m2	m3	m4	m5	m6	m7	Score	Size
WNM0	2	16	63	38	44	46	77	53.00	68
WNM1	2	16	63	38	46	44	12	52.00	55
WNM2	2	16	38	42	63	44	46	63.00	92
WNM3	2	16	63	38	44	46	12	55.00	79
WNM4	2	16	63	38	46	44	12	53.00	47
WNM5	2	16	63	38	44	46	77	52.00	75
WNM6	2	16	63	38	44	46	12	53.00	84
WNM7	16	38	63	58	2	9	44	57.00	65
WNM8	2	16	63	38	44	46	76	43.00	63
WNM9	2	16	63	38	44	46	52	50.00	34

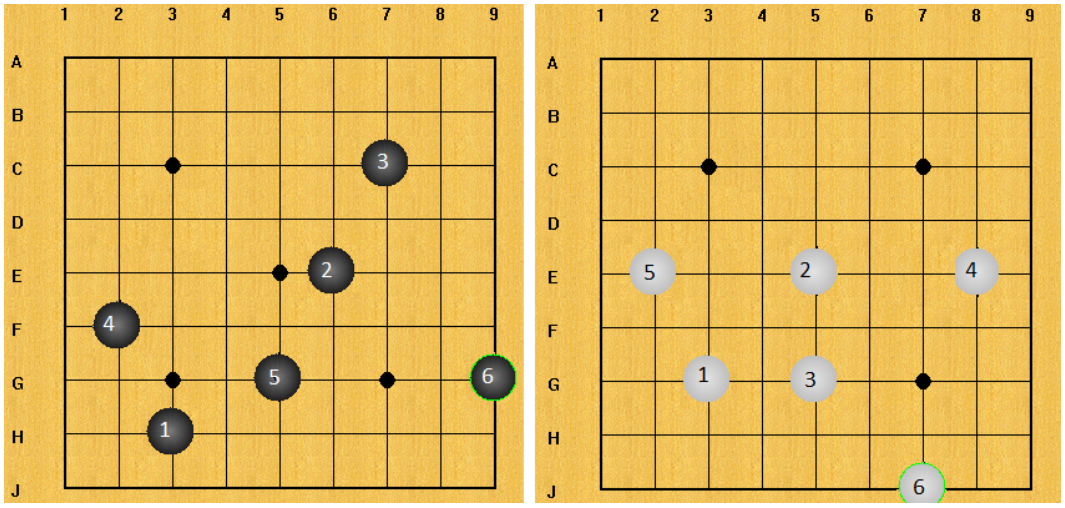


Figure 6.14: First moves evolved non-memory: NicoGoB (black) from execution BM3 - NicoGoW (white) from execution WM5

In general it was observed that the use of blueprint's memory help to the blueprints to keep the best strategies during the evolution, but it was not observed a more superior results comparing to the same computer player not using memory mechanism. What was observed is that the strategies evolved by the computer are different because in every generation it is reinforce different strategies if they produced good results during the competitions. By contrary, not using the memory mechanism evolved the same start-game strategies (even these are not good start) obtaining similar or a little bit better results than using of memory mechanism.

So, can be conclude that the use of non-memory mechanism forget some strategies that in the previous generation produced good results, which is one of the pathologies discussed in an co-evolutionary learning process.

As it can be observed in the Figures 6.14 the memory mechanism tend to reinforce the players (black or white) to start playing in the center of the board. The same behavior was observed in the other executions. In fact, the memory mechanism reinforce with more high values the first moves during the evolution as it was discussed in section 5.5.

The importance to reinforce start strategies is important as it was discussed previously, because according to some Go players start playing in the center of the board is a good start-game strategy .

For simplicity was used Wally for evolving these computer Go player using NicoGoW and NicoGoB, which is pending is to observe what could be the result evolving these players against a more strong computer Go player as GnuGo or

another, the author believe that the results probably could be similar, but, taking more generations to beat these stronger players.

6.3.2 Analysis of the Techniques proposed in the Evolution

In this section is going to be analyzed two techniques introduced in this thesis, the use of memory mechanism and the dynamic sizing of the blueprints, and it discussed the measurement of the diversity produced by these mechanisms in the population.

As it can be observed in the Tables 6.3, 6.4, 6.5 and 6.6 the best players after the evolution of 1000 generations has different sizes (column size). It is not possible to conclude which size is the one that can produce better strategies, in some cases the best strategy evolved will need less neurons that in other evolutions. The author believe that this is a good indication that does not exist the unique size for the blueprint structure, and this should be discovered during the evolution.

The Figure 6.15 shows the first 7 moves evolved by NicoGoB playing black against Wally at the generation 1000 using and not using blueprint memory mechanism described above. The stones with more dark color around it is because more stones has been placed in that position in different executions. These positions are circled. So, in this figure can be observed that in both cases the first moves are not in the center of the board. In case of not using the memory the first moves are similar in 10 different generations. In case of using the memory mechanism the first 7 moves are more diverse, but, even this, the first moves are not in the center of the board.

The Figure 6.16 shows the first 7 moves evolved by NicoGoW playing white against Wally at the generation 1000 using and not using blueprint memory mechanism. In this figure can be observed the similar behavior as in case of black; in case of not using the memory the first moves are similar in 10 different generations, and in case of using the memory mechanism the first 7 moves are more diverse and but in both cases not in the center of the board.

In the Figure 6.17 can be observed the measurement of the population's diversity and the best player's diversity in every generation during the evolution of NicoGoB playing black against Wally. To calculate the diversity of the population was used the equation 5.9. In this figure can be observed that in both cases the diversity grows during the evolution.

In conclusion the techniques applied to the evolution against Wally have show good results in the sense that after some generations the players evolved using NicoGoW and NicoGoB can beat Wally easily, and in some cases even the start

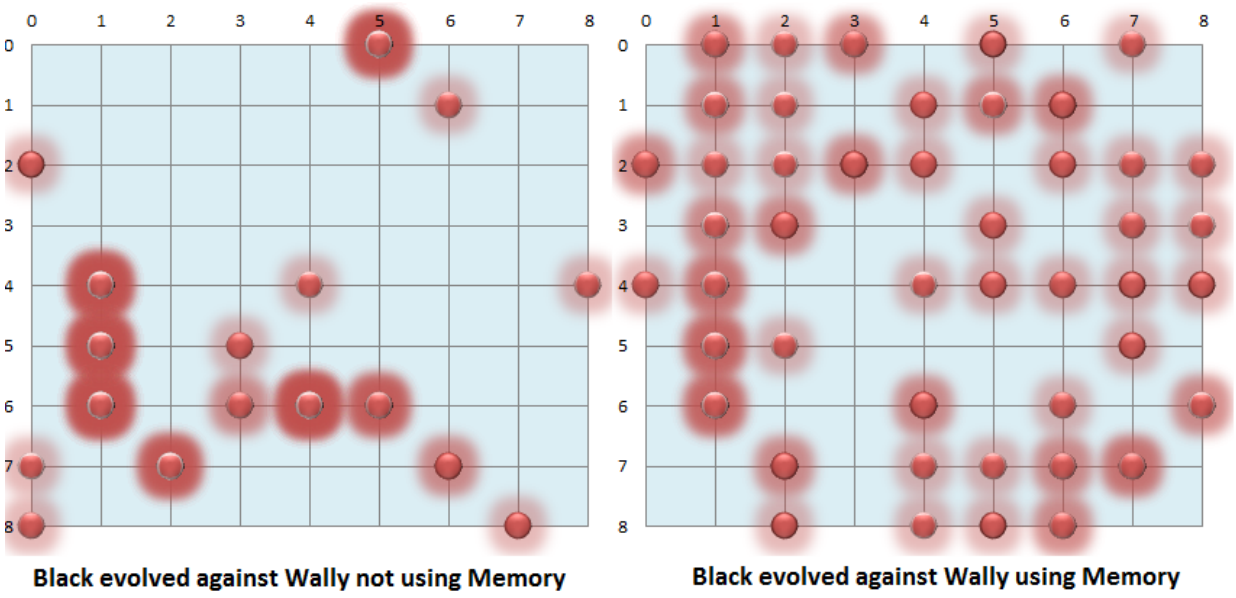


Figure 6.15: First 7 moves of Black evolved against Wally for 10 executions

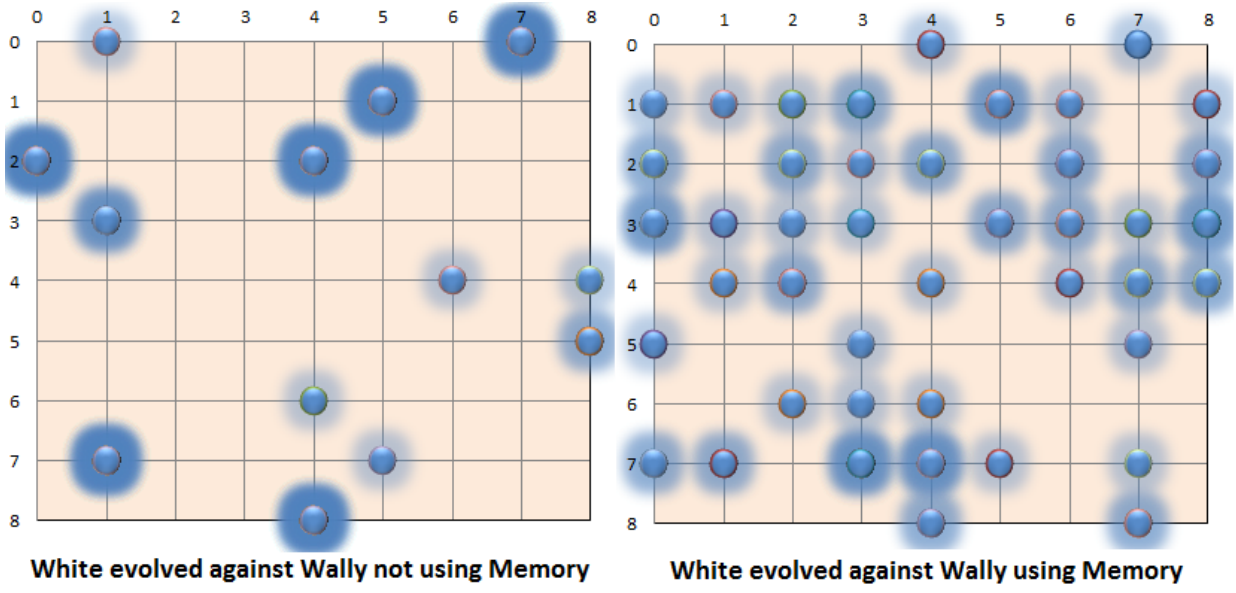


Figure 6.16: First 7 moves of White evolved against Wally for 10 executions

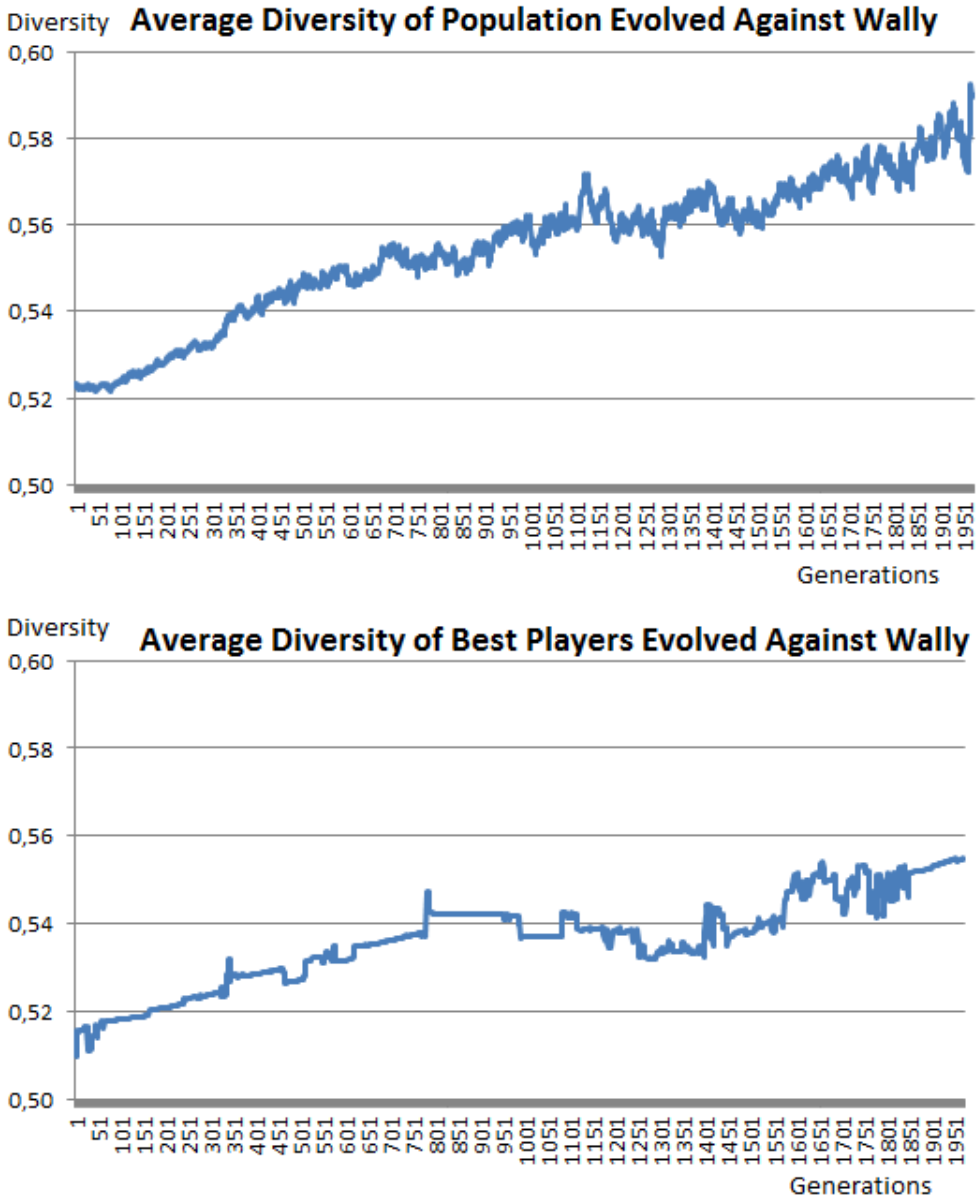


Figure 6.17: Diversity of the Population and Best Players playing Black (using memory) Evolved Against Wally

strategies are not necessarily good start-game strategies as can be observed in the Figures 6.16 and 61.7. Using the memory mechanism the players try to evolved strategies that intent to start in the center of the board, but as it was show in the Figures 6.16 and 6.17 this is not finally achieved. In some evolutions was needed less neurons that in other evolutions to create good strategies which indicate that this mechanism create networks with less redundancy and more effective.

6.4 Co-evolution of Two Computer Go players

As it was discussed in the previous sections, in this work is used co-evolution to solve the Go game. It the previous chapter was presented the solution concept to solve this problem, in which the best player of the last generation should be the solution to this problem, and it was discussed as well, the main issue is to know what is the last generation till these solutions will evolve. In this thesis were evolved computer Go players till different number of generations. It was evolved till generation 1000 and later till generation 2000 and more. It was not evolved in this phase till more number generation because of limited resources, but, the results obtained gave us an idea about whether co-evolution was progressing and whether good solutions were discovered. It was got access to Magerit and as part of future actions is evolve these players till more number of generations and observe the results.

It was explained advantage of using co-evolution and pathologies that can be observed using this technique. In next sections is explained the experiment performed using co-evolution to solve this problem, co-evolving two computer Go players NicoGoW and NicoGoB coded in C++ by the author. The machine used to run these experiments has 8 processors running a Linux Ubuntu. Using this configuration, co-evolve these two players is taking usually 5 days for every 1000 generations.

6.4.1 Setup of the Experiments

The figure 6.2 shows how the computer Go players NicoGoW and NicoGoB are implemented. These two players which are used for co-evolution of two neuron populations interact through the Referee implemented by the OpenGo. The OpenGo framework was modified to allow multiple games to run in Linux and Windows environments.

The game between these two players ends when three times the players do not have more valid moves to play or when consecutive pass happens. The Referee validate if these moves are valid. i.e. suicide moves.

The initial neuron populations used in these experiments are the same (one

white and other black players) for all the experiments to evaluate the techniques discussed in this thesis, and avoid noise because of many neuron populations.

The neuron population size for White and Black players is 2000 neurons, the size of the Gene is 100 for all neurons (for White and Black players) which contains the inputs/outputs and the weights which connect to the hidden neurons. The number of trials per generation was setup to 100, and the number of Go players (or blueprints) created for White and Black players is 100, so, in the competition of every generation there is 10,000 interactions or games between White and Black players.

The number of best players to keep in the Hall of Fame in every generation was 62, and from there the 20 first best players was used for reproduction.

The rate for crossover rate used was 30% and the mutation rates was setup to 3%. The size of the blueprints were setup to [24,100] and applied the dynamic sizing for blueprints mechanism described above not allowing blueprints with less than 24 neurons neither greater than 100 neurons during the crossover of blueprints.

The immigration rate was calculated were calculated using the equation 5.1 where β parameter was setup to 2.

For every corner in the board there is a input/output position. For the player the value of the input position is 1, and for the opponent -1, and +10 and -10 for more recent moves respectively as it was discussed previously. The Chinese scoring method was used, and not suicide moves were allowed and used Komi=0 for all the experiments.

6.4.2 Co-evolution using Different Fitness Functions

As it was discussed in the previous chapter, in this thesis was used different fitness functions to identify which ones are the best fitness function which can co-evolve better strategies. As it was analyzed in the previous sections, the use of CFSA and CFS, including the Blueprint memory method was good to create good start strategies but it was not good enough to discover good strategies and be maintained during the co-evolution process.

It was observed that in competition of these players co-evolved against Wally the results are not good enough, even these players which have a good start-game strategy, these produce bad results against Wally. So, from these section it is going to be experimented with different fitness functions to discover and maintain these good strategies during the co-evolution process.

Different evaluation functions has been used, the first ones described in the section 5.13.2 in the equation 5.11 which consider the average score of the player after the competition against all opponents, and the second described in the equation 5.10 which consider the number of wins of the player in a competition

against all opponents. The results of these evaluation function is discussed in the next sections.

The other evaluation function used in these experiments were the one explained in section 5.13.2.1, in which for players which belongs to Hall of Fame, the fitness calculated for these players take in account the fitness obtained in the previous generations. In the results observed applying this technique it was observed that best players of Hall of Fame were maintained for more generations in this selected group. It was observed that not using this method the best players in every generation were different players, but using this method the same best player was the best after all competitions for two or three consecutive generations, which is an indication that the best strategy of a generation could be the best for the next generations (at least for the next two or three generations).

6.4.3 Experiments Performed and Discussion of Results

In this section is presented the experiments performed and discussed their results for the co-evolution of Black and White populations using NicoGoB and NicoGoW. The experiments performed were group five groups.

- Test the fitness functions proposed in this thesis as explained above, fitness functions as CFS and CFSA, and the ones defined in the equations 5.10 and 5.11, with the intention to measure the diversity and generalization created using these evaluation functions. In this section will be discussed the start game strategies co-evolved.

- Observe the start-game strategies co-evolved. As it was discussed, start game strategies are good when the player start playing in the center of the board. For this it was analyzed the first seven move obtained at the end of every experiment (generation 1000).

- Test the best strategies co-evolved in the experiments with the evaluation functions described above against Wally to observed whether strategies evolved are good enough to beat Wally.

- Measure the diversity of the populations during the co-evolution. It is measured the diversity with the intention to find a relationship with the generalization, and whether is searched more complex strategies during co-evolutionary learning processes. For measurement of genotype diversity is used the equation 5.9 described in the section 5.12.1.

- Evaluate how general are the strategies co-evolved. The generalization is measure based on the number of wins of the players co-evolved against a test cases created randomly.

6.4.3.1 Co-evolution using CFS and CFSA as Fitness Functions

In this section discussed about the experiments using application of CFS Rosin & Belew [1997] as fitness function and compared with the CFSA method proposed by the author. The intention to use the CFSA method is introduce more phenotype diversity to the population that are evolving.

In the Figure 6.18 can be observed the number of wins by White and Black players during 10 executions of co-evolution using the same initial populations for White and Black players. The initial populations used for all co-evolution experiments were trained previously with Wally.

The total number of games per generation was 10000 which is obtained to compete 100 Black players against 100 White players. In this figure can be observed that during the co-evolution process that number of times that Black or White wins the game are oscillating during this process which is a good indication that there is not a dominant population during all this process and the co-evolution can happen.

The experiments were executed till generation 1000 to observe the evolution of these populations. In some experiments as CS1 and CS8 during many generations White players obtained more number of games won against Black players. In other experiments as CS3 and CS5 the Black population has more games won against White population, but this not indicate that one population dominate to the other because the difference in the number of games won is not big and even in many generations one population won more games, there are some generations in which the opponent won some games as well. So, we can say that there is not a dominant population in the co-evolution performed using CFS.

The Table 6.7 shows the first 7 moves performed by Black and White players co-evolved using CFS as fitness function at generation 1000. Comparing with the first 7 moves evolved against Wally, can be observed that the start-game strategies start playing around center of the board as can be observed in the the Figure 6.19 and 6.20.

The table 6.7 includes the size of these players at that 1000 generation. It can be observed that the size of the blueprints (number of neurons in the neural network) vary in every experiment. For example, in the experiment CS0 the size of the best player at the generation 1000 is 80 for Black player and 41 for White player. In case of the experiment CS1, the size of the best Black player and best White player at generation 1000 is 91 and 91 respectively. Other sizes can be observed in the other experiments. So, we can say that there is not a unique best structure of the neural network at the generation 1000.

The Figure 6.19 and 6.20 shows the first 7 moves co-evolved in the experiment CS3 and CS9 by Black and White players using CFS respectively. In both figures can be observed that strategies co-evolved is trying to start playing around the

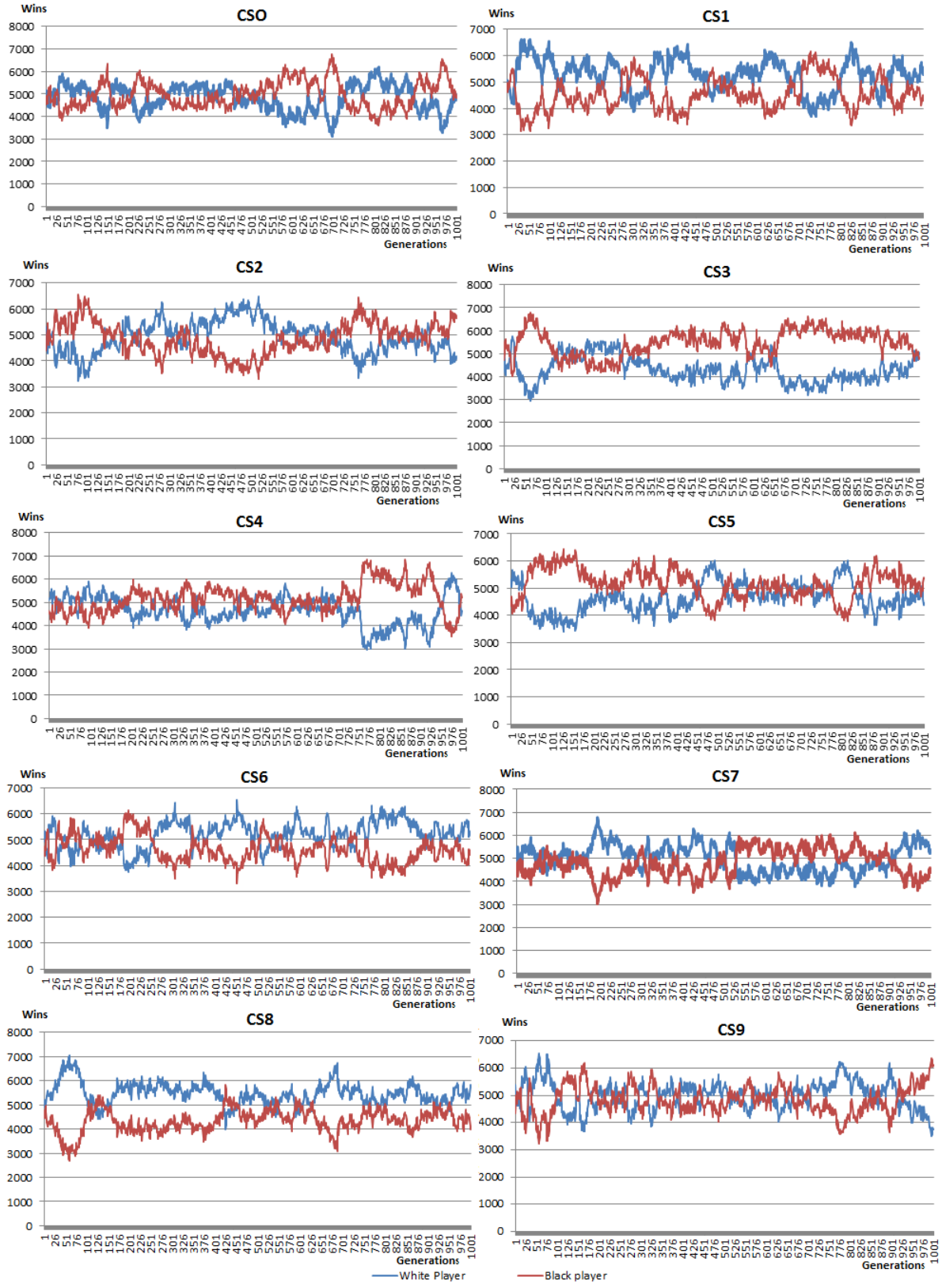


Figure 6.18: Number of games won by White and Black players during co-evolution using CFS in 10000 competitions in every generation

Table 6.7: First 7 moves of White and Black best players using CFS mechanism at generation 1000

Execution	Player	m1	m2	m3	m4	m5	m6	m7	Size
CS0	Black	33	23	13	42	14	56	65	80
	White	32	60	59	49	55	31	40	41
CS1	Black	29	40	38	41	49	20	11	91
	White	39	30	32	12	31	50	56	91
CS2	Black	56	31	32	22	40	24	14	93
	White	30	57	59	24	58	13	23	77
CS3	Black	42	33	50	24	61	49	60	56
	White	51	43	40	52	59	33	41	46
CS4	Black	42	30	34	24	32	35	20	29
	White	43	50	33	68	30	70	52	49
CS5	Black	49	48	42	39	68	41	31	81
	White	19	23	21	10	45	12	25	41
CS6	Black	12	39	10	41	47	56	30	39
	White	33	57	66	32	59	31	30	86
CS7	Black	12	42	24	33	11	37	28	61
	White	38	48	21	57	22	19	47	45
CS8	Black	55	21	12	29	48	68	39	33
	White	33	30	32	31	19	38	16	59
CS9	Black	57	48	30	66	21	39	19	90
	White	50	65	58	31	19	40	49	47

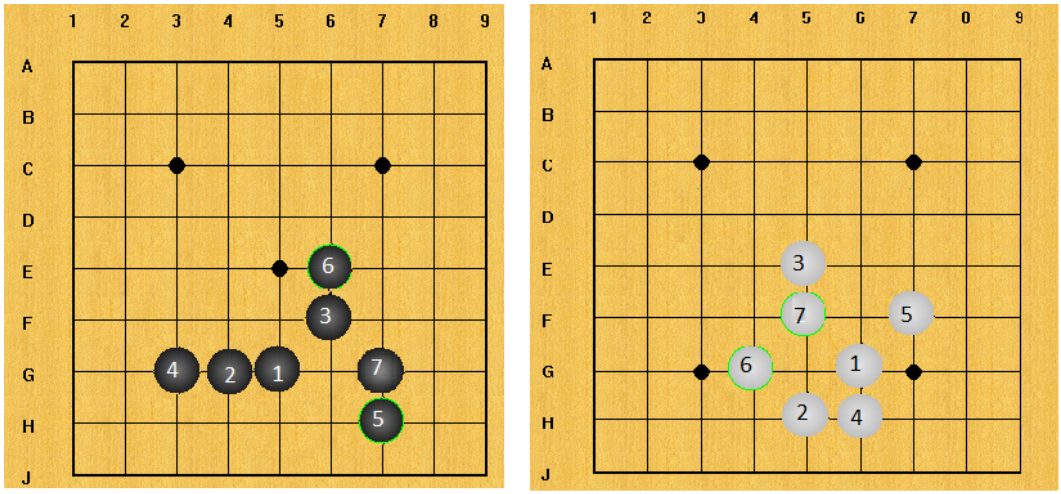


Figure 6.19: First moves of the co-evolved players at generation 1000 - experiment CS3

center of the board, which is a good indication that the co-evolution process reinforce these strategies by itself. Comparing Figure 6.14 which was the evolution against Wally and Figures 6.19 and 6.20 can be observed different initial strategies were evolved and co-evolved by these two approaches, and can be concluded that co-evolution is a better method for searching better start-game strategies.

The reinforcement mechanism using the blueprint's memory is the same in these experiments, in evolution and co-evolution, the only different is the access to the set of test cases, as it was discussed, evolution against a deterministic player approach learn from only one player, in this case Wally, and in co-evolution the players has access to a more diverse set of test cases and the author believe that could be the reason why these start-game strategies are co-evolved.

The Figure 6.21 shows the results of competing the best strategies of every generation co-evolved using CFS against Wally. These 1001 best players from generation 0 till generation 1000 are the same set of experiments CS0 till CS9 discussed previously. In this figure can be observed that even good start-game strategies evolved using co-evolution, in a competition against Wally these strategies were not good enough to beat Wally. In the majority of competitions of these best players in 10 different experiments, the best scores for these players rarely pass 30 points of score (from 81 which is the maximum score in a 9×9 board). Just in few cases, there were players from some generations that were able to beat Wally.

The other thing that can be observed is that even there are few best players that were able to beat Wally, these strategies were not the best in the next generation or not maintained for the future generations, or at least best player

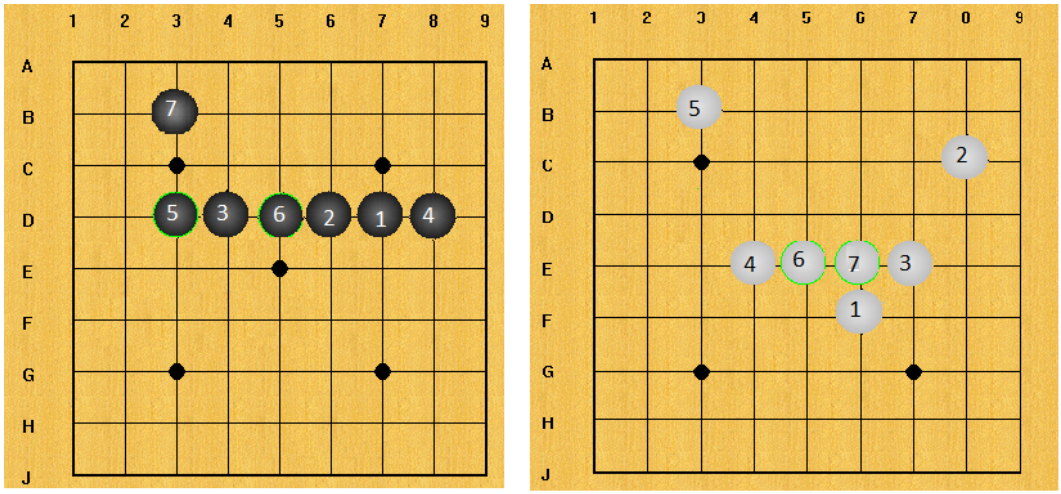


Figure 6.20: First moves of the co-evolved players at generation 1000 - experiment CS9

of this generation were not the best in the next generations. In the Figure 6.21 is marked in red circle the game in which the best player from a generation beat Wally.

The Figure 6.22 shows the results of competing of the best White players co-evolved using CFS against Wally from the experiments discussed above. In this figure can be observed similar results as it was with Black players, there were very few generations in which the best player from some generations were able to beat Wally. So, even the co-evolution using blueprint's memory mechanism is good to find good start-game strategies, is not good enough to beat a weak opponents as Wally, or in the worse case, it is not good to keep the good strategies that were learned during the co-evolution process till generation 1000.

The Figure 6.23 shows the co-evolution of Black and White players using CFSA method proposed by the author. The initial population used is the same initial population used in the co-evolution experiments using CFS with the intention to compare results. As it happened in the previous experiment using CFS, can be observed that during this co-evolution process using CFSA there was not dominant population during all the co-evolution process till generation 1000, even in some generations White players beat more times to Black players, and others Black players beat more time White players. As in the previous experiments the total number of games in every generation was 10000 games.

The Table 6.8 shows the first 7 moves strategies played by the best player during the co-evolution process at generation 1000. As in the case of using CFS, the size of the best players, or blueprints at generation 1000 is different. For example, in the experiment CA0 the best Black and White player at generation

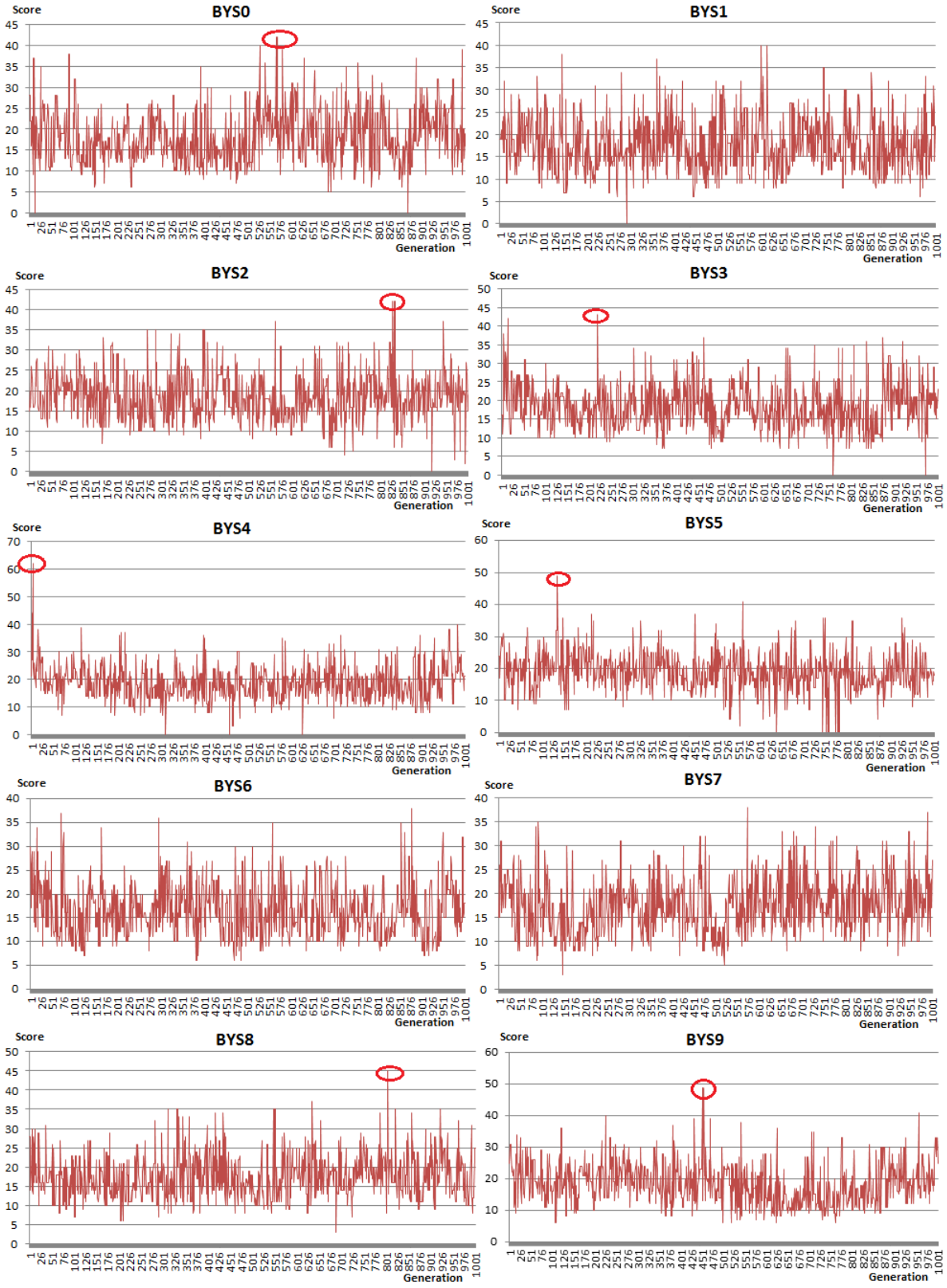


Figure 6.21: Results of competing the best co-evolved Black player against Wally in a board 9x9 using CFS

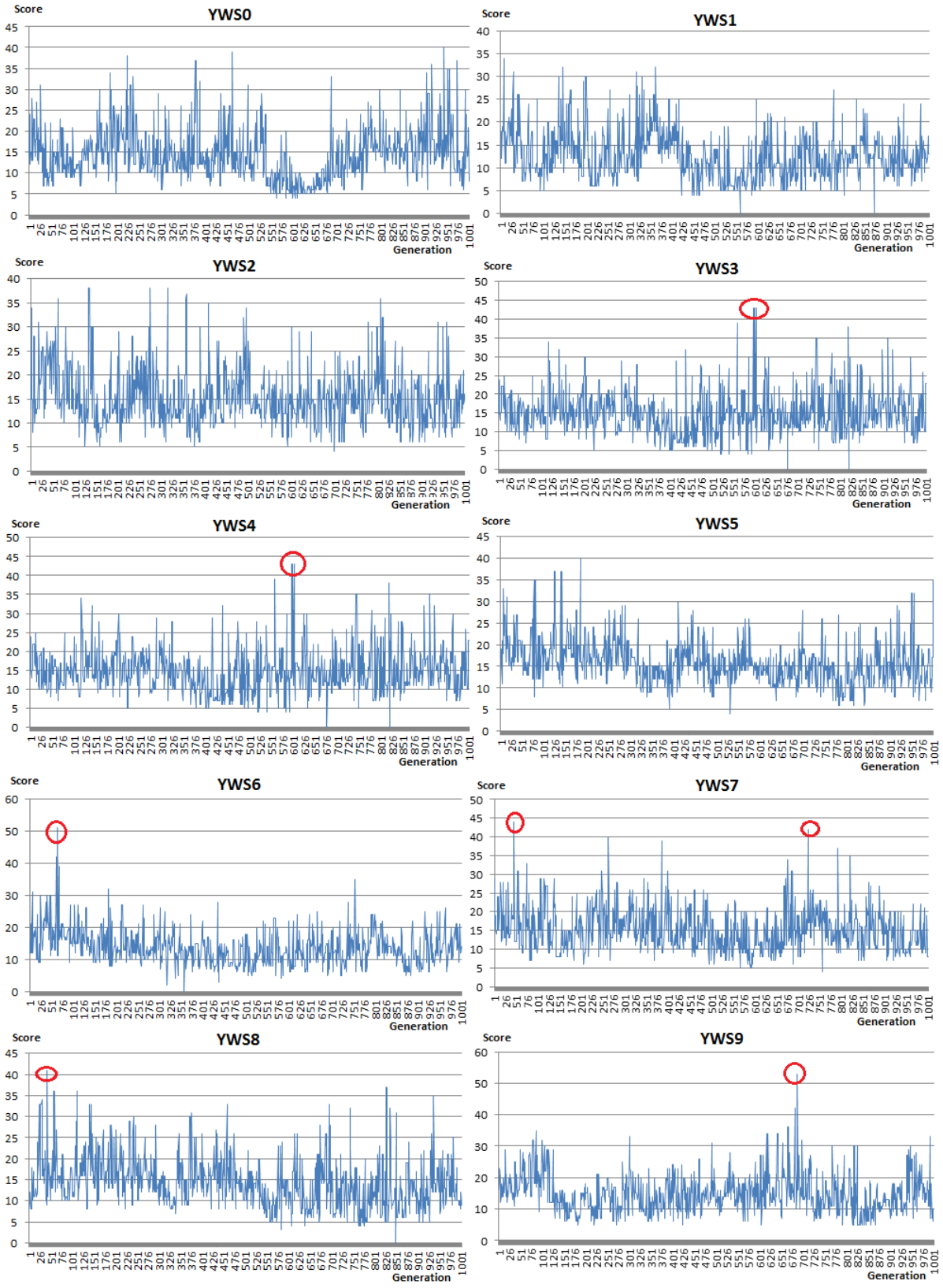


Figure 6.22: Results of competing the best co-evolved White player against Wally in a board 9x9 using CFS

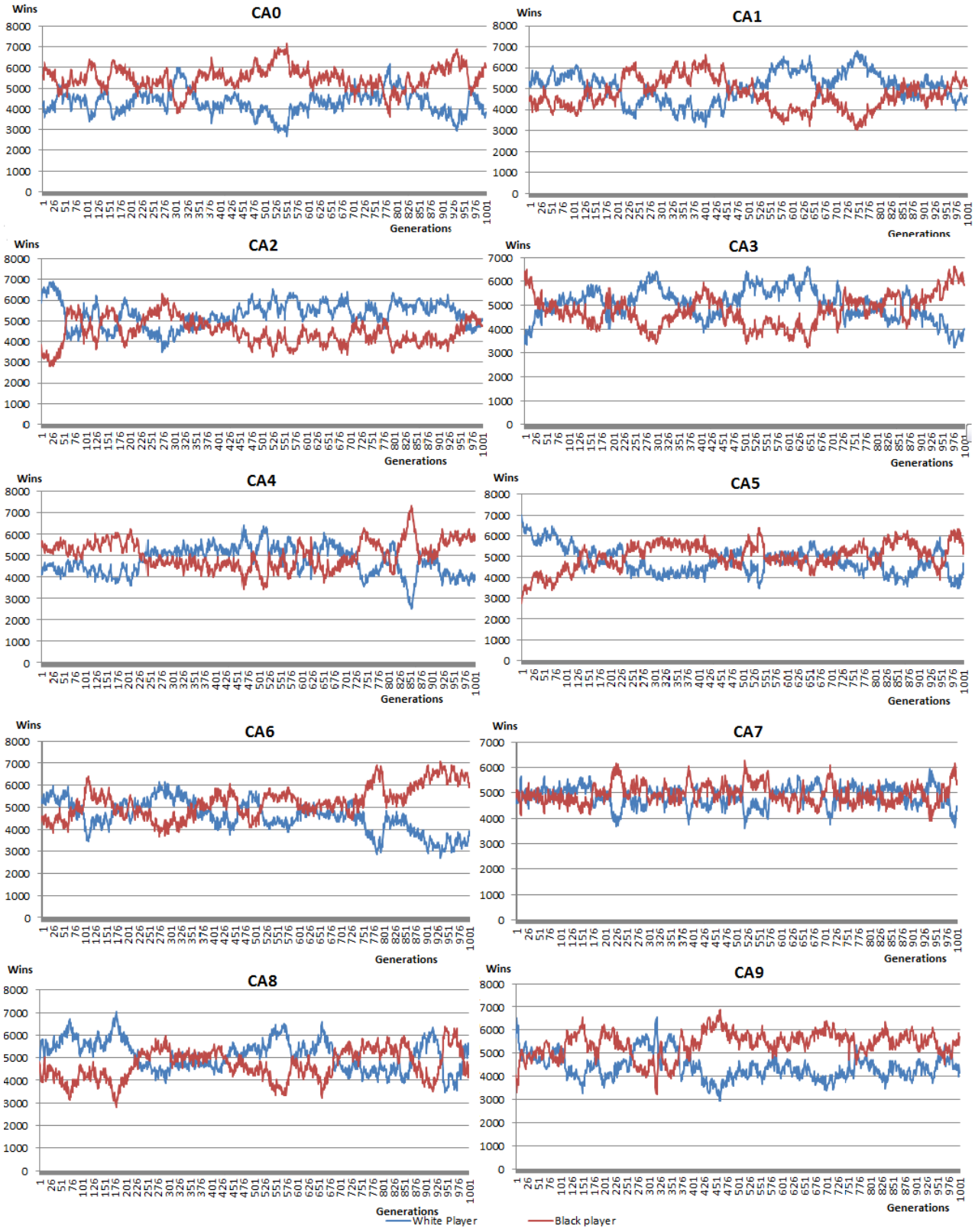


Figure 6.23: Number of games won by White and Black players during co-evolution using CFSA in 10000 competitions in every generation

Table 6.8: First 7 moves of White and Black best players using CFSA mechanism at generation 1000

Execution	Player	m1	m2	m3	m4	m5	m6	m7	Size
CA0	Black	46	42	32	50	40	23	52	96
	White	33	43	31	56	34	59	39	32
CA1	Black	43	41	69	25	58	34	67	25
	White	22	40	69	31	42	13	49	74
CA2	Black	22	75	41	39	28	33	40	73
	White	48	67	13	39	31	19	29	89
CA3	Black	40	49	29	23	24	31	50	76
	White	49	23	41	68	32	37	39	87
CA4	Black	22	21	30	14	43	59	34	77
	White	31	39	12	29	32	50	25	98
CA5	Black	50	33	42	29	69	20	61	94
	White	51	41	49	32	59	46	12	74
CA6	Black	41	65	42	48	57	61	51	92
	White	59	37	58	38	43	11	34	42
CA7	Black	47	29	65	67	55	37	39	66
	White	34	41	32	49	57	51	59	77
CA8	Black	48	65	13	14	22	24	32	99
	White	51	21	14	23	32	42	15	52
CA9	Black	49	21	39	47	57	48	37	73
	White	38	41	40	60	30	48	39	95

1000 has 96 and 32 neurons respectively.

The majority of the first 7 moves are the same in the 10 experiments performed as it was the case of using CFS, just in some cases were observed that these moves vary but mostly in the order. But, when the same experiment were tested not using the memory mechanism was observed that the first 7 moves with this configuration vary more not existing a clear start-game strategy evolved at the generation 1000.

Some of first 7 moves presented in the Table 6.8 is shows in the Figures 6.24 and 6.25. The Figure 6.24 shows the first 7 moves played during the co-evolution from experiment CA0 and Figure 6.25 shows the first 7 moves from experiment CA3. In both cases, as in the other experiments performed, the strategy learned by this process is to start playing in the center of the board.

Based on the results presented so far there is not clear conclusion which method, CFS or CFSA, can create better start-game strategies, in both cases, using any of these methods, the strategies learned are to start playing around the

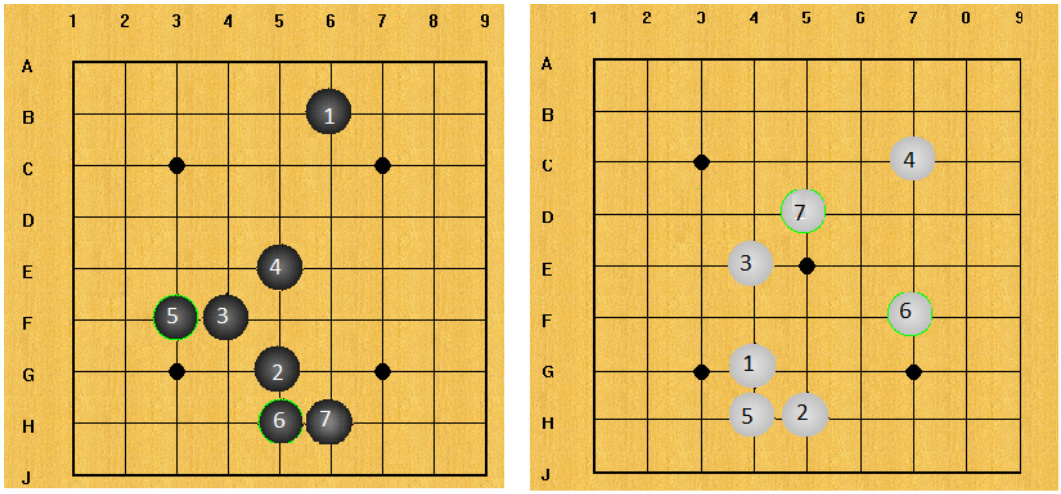


Figure 6.24: First moves of the co-evolved players at generation 1000 - Co-evolution CA0

center of the board.

The Figure 6.26 and 6.27 shows the results of competing the best players co-evolved of every generation using CFSA against Wally. As it was observed in the previous experiments using CFS, even co-evolution using CFSA as evaluation function discovered good start-game strategies, these were not good enough to beat a weak computer Go player as Wally. In the majority of the cases the score of the best White and Black players did not reach 30 points.

As it was observed in the previous experiments using CFS, even in some generations Black or White player were able to beat Wally, these best strategies were not kept for the next generation, or at least these strategies were not the best players of next generations. As it was discussed before, for every generation it is saved as file the best blueprint player of each generation. This blueprint is a file that can be tested against any other computer or human player any time.

The Figure 6.28 shows the average of the scores obtained by the best Black and White players co-evolved from the previous experiments competing against Wally till generation 1000. It can be observed that apparently CFS and CFSA as fitness functions are creating weak best players, or at least not better players, compared against an external weak player as Wally. So, as it was discussed before, using CFS and CFSA as fitness sharing sampling can improve the diversity of the strategies learned using co-evolution, and not necessarily improve the global fitness of the agents evolved, at least till the generation 1000.

The table 6.9 shows the average results obtained by best players of every generation from the experiments discussed previously using CFS and CFSA methods in a competition against Wally. The table shows the results of best players of the

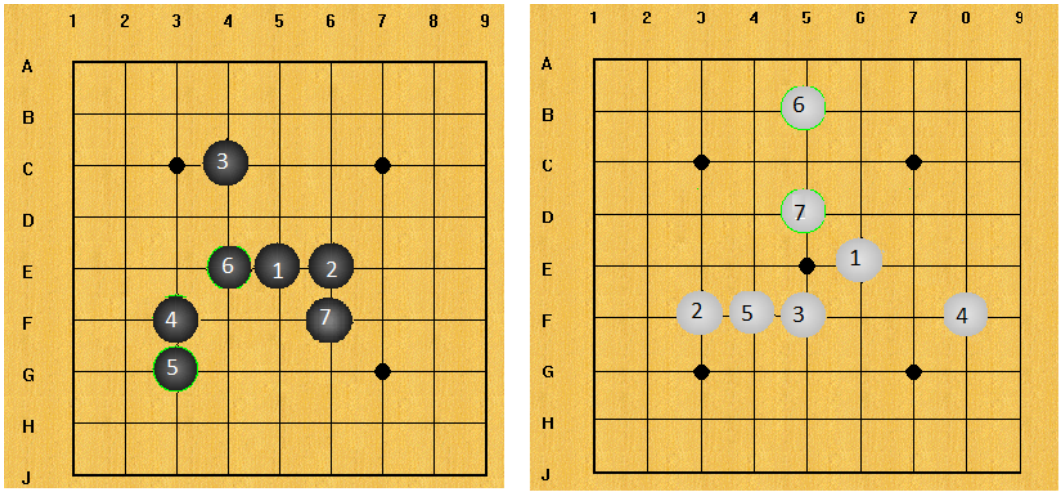


Figure 6.25: First moves of the co-evolved players at generation 1000 - Co-evolution CA3

Table 6.9: Average and Standard Deviation of Scores against Wally of Last 500 best players co-evolved using CFSA and CFS till generation 1000

Method	Player	Average	Standard Deviation
CFS	Black	18.03	1.98
	White	13.75	1.78
CFSA	Black	17.77	1.96
	White	14.31	2.02

last 500 generations (from 1000 executed in these experiments). In this table can be observed that there is not too much different in term of results playing against Wally, probably playing White players and CFSA shows better results than CFS for White players, and Black players using CFS produced a little better results against Wally than Black players using CFSA, but the difference is too small to get some conclusions about which method learned more complex strategies measured using Wally as external agent.

The Figures 6.29 shows the first 7 moves of the Black players from the 10 different executions discussed before using CFS and CFSA. The stones with more dark color around it is because more stones has been placed in that position in different executions. In both cases can be observed that there is a concentration of the first 7 moves in the center of board during these 10 experiments. If this figure is compared against Fig 6.15 (evolution using memory) can be observed a very good improvement in the start games strategies using sharing sampling methods as fitness function.

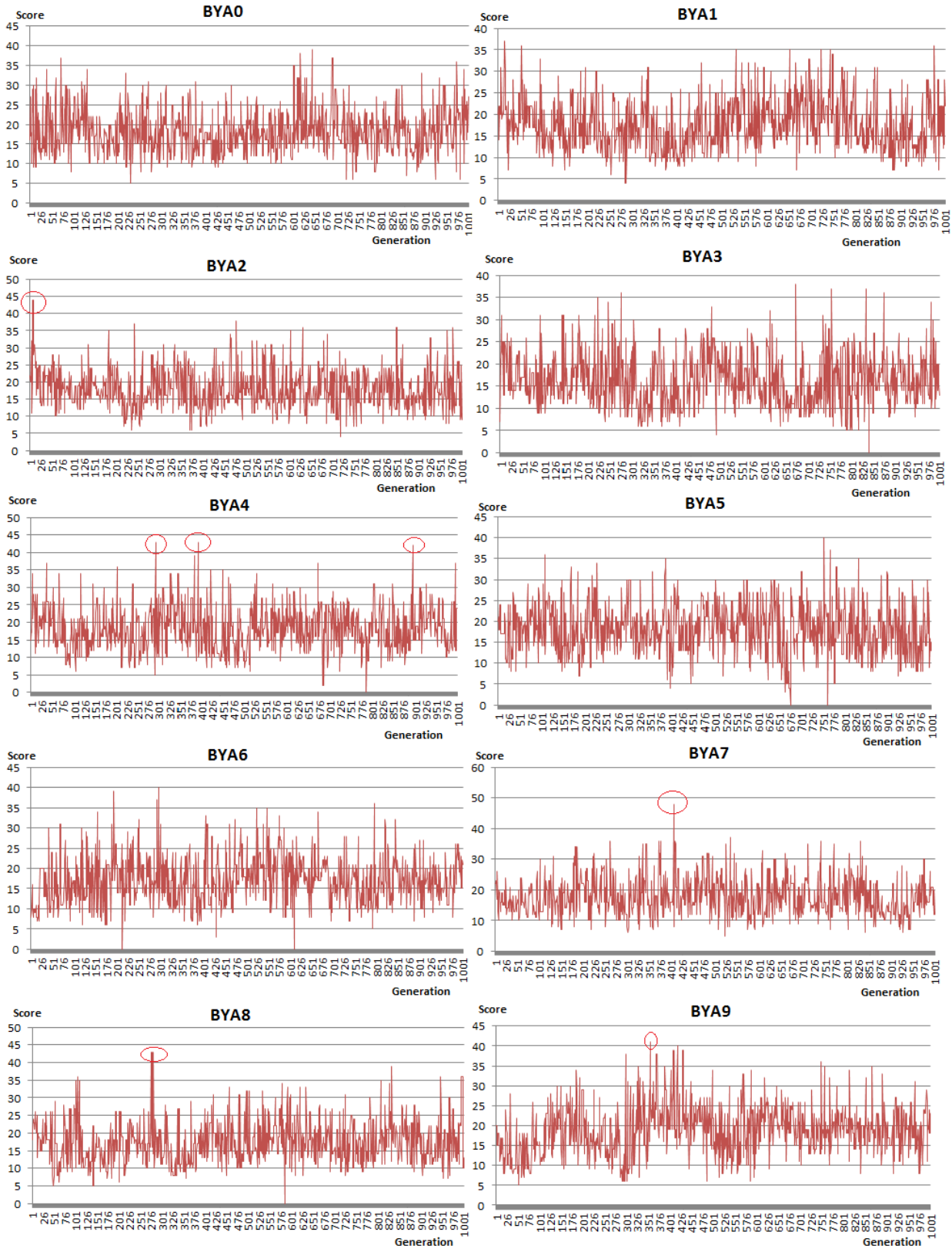


Figure 6.26: Results of competing best Black player of every generation against Wally in a board 9x9 using CFSA

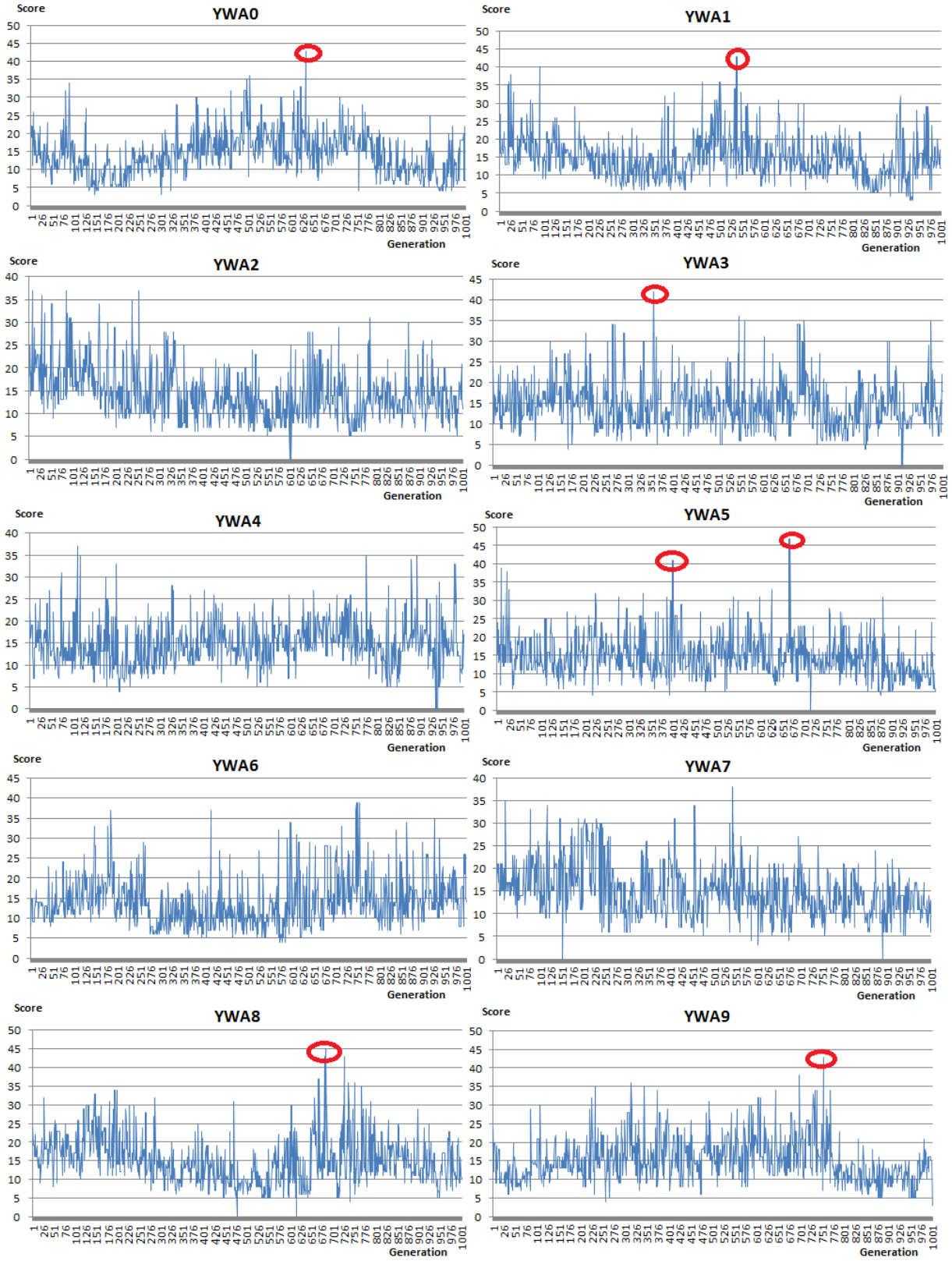


Figure 6.27: Results of competing best White player of every generation against Wally in a board 9x9 using CFSA

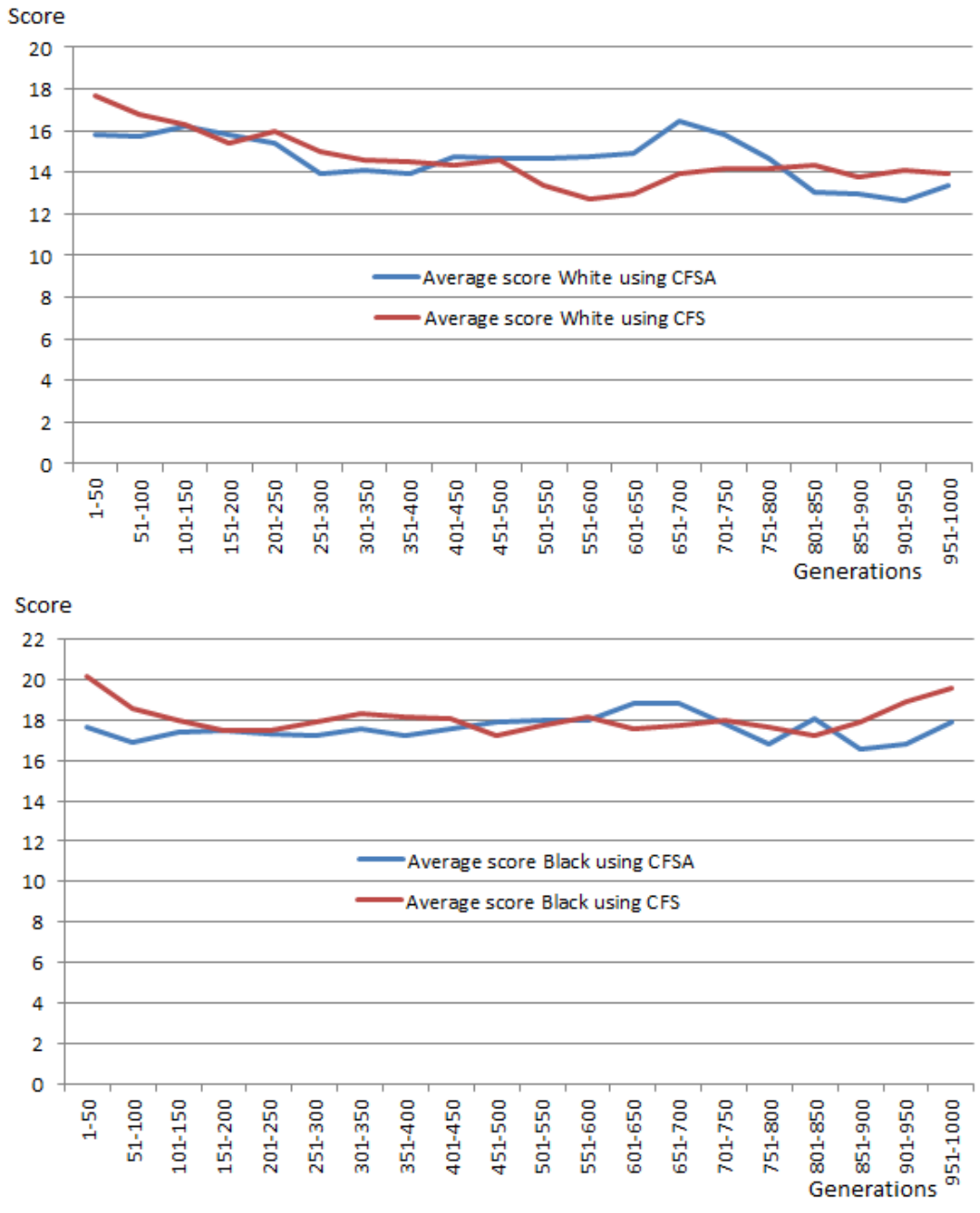


Figure 6.28: Comparison of Average Scores of Black and White players using CSFA and CFS compiting against Wally - Scores group every 50 generations

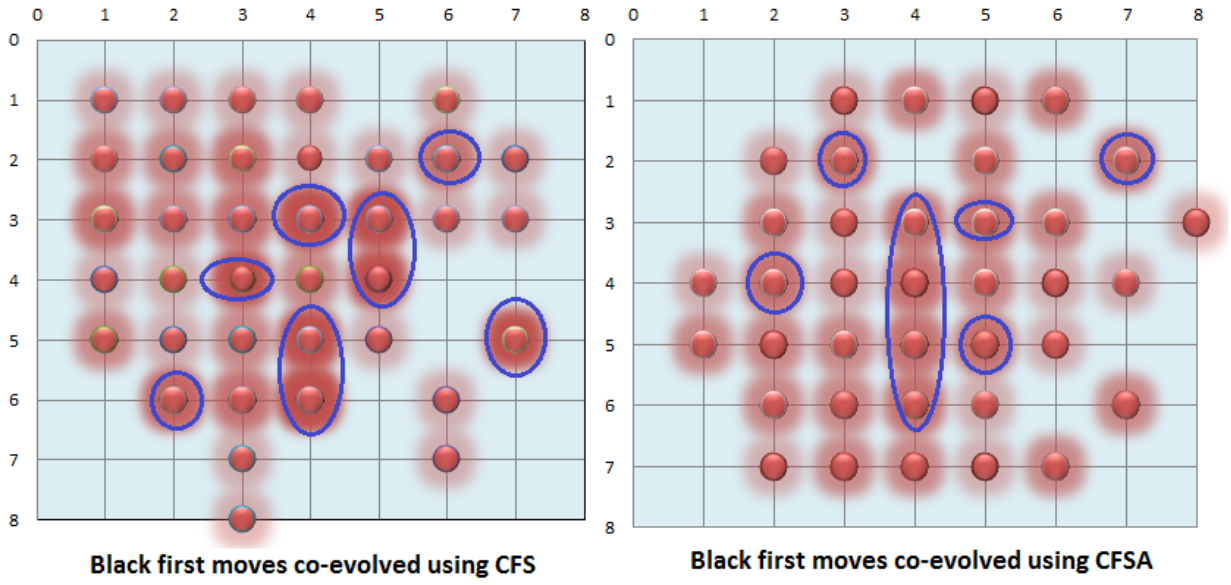


Figure 6.29: First 7 moves co-evolved by black players for 10 executions

The Figure 6.30 shows the first 7 moves of White players from 10 different executions discussed before using CFS and CFSA. The results are similar to the Figure 6.29, but in this figure the start-game strategies co-evolved using CFSA is slightly better than the strategies co-evolved using CFS, because in this figure can be observed more stones around center of the board using CFSA than CFS. It was used the first 7 moves from the Table 6.7 and 6.8 to populate the black and white stones in the Figures 6.29 and 6.30.

The Figure 6.31 shows the first 7 moves for Black and White players in 10 different experiment not using the blueprint's memory mechanism and using CFSA and CFS as evaluation function at the generation 1000. As it was discussed previously the difference using and not using the memory mechanism is that not using the memory mechanism the first 7 moves vary in every generation, even in the same generation for the same best player. So, not using the memory mechanism in blueprints there is not clarity what are the start-game strategies learned.

The Table 6.10 shows the average results and standard deviation of the last 500 competitions of the best players co-evolved using CFSA and not using the blueprint's memory mechanism introduced by the author. It can be observed that in case of Black players comparing to experiments using just CFSA as evaluation function has slightly better average, but the standard deviation is bigger, which can indicate that the strategies in the last 500 generations (from 1000) varied more. In case of White player the average is less than using the memory mechanism, and the standard deviation is slightly less as well.

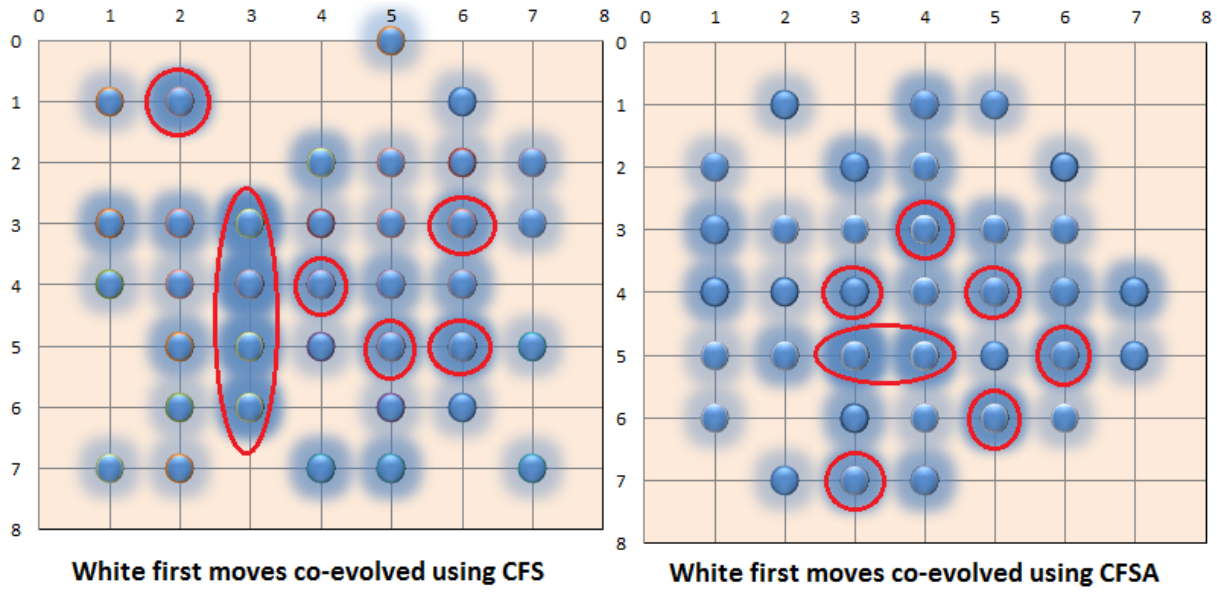


Figure 6.30: First 7 moves co-evolved by white players for 10 executions

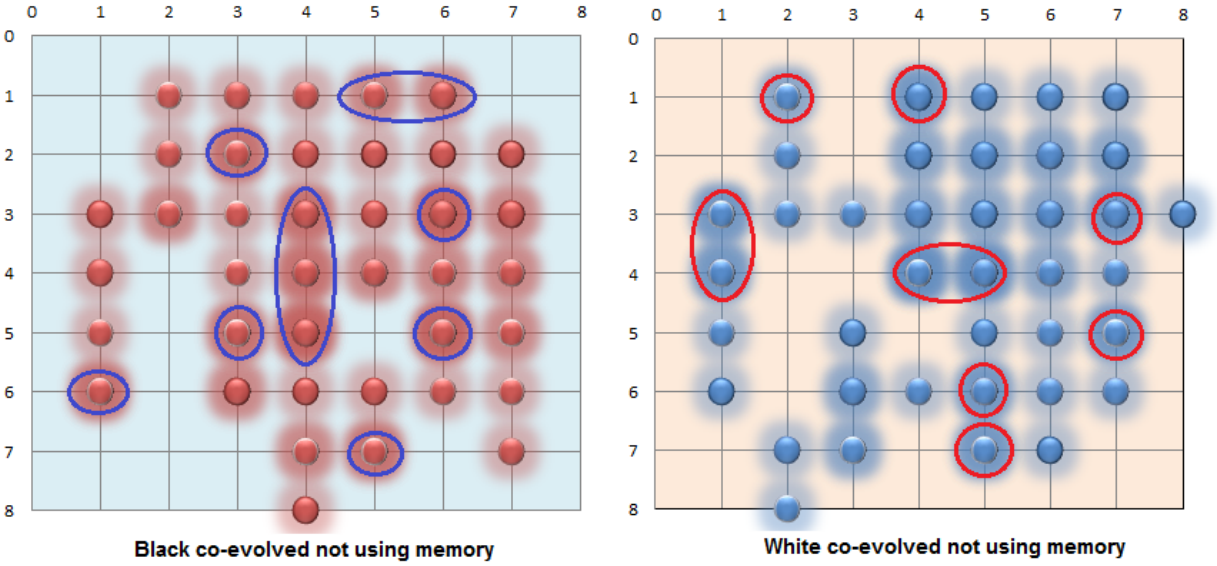


Figure 6.31: First 7 moves co-evolved by Black and White players for 10 executions not using blueprint memory mechanism at generation 1000

Table 6.10: Average and Standard Deviation of Scores against Wally of Last 500 best players co-evolved using CFSA and Non-Memory mechanism

Method	Player	Average	Standard Deviation
CFSA	Black	18.04	2.26
	White	14.29	1.98

So, based on these results the author believe that CFS and CFSA as evaluation functions are not creating by themselves more complex strategies and it is probably needed more complex evaluation functions or at least evaluation functions that can incorporate more information of the game. This is going to be discussed in the next sections.

6.4.4 Measurement of the Diversity of the co-evolved strategies

In this section is discussed the diversity of populations co-evolved using different methods proposed in this thesis, identifying which method produce more genotype diversity in the population measured as it is described in the section 5.11. So, in this section is analyzed the result of some experiments:

- Compare CFSA and CFS methods to identify which one introduce more genotype diversity.
- Analyze the result of increasing the β of RIR to 3.0 using just CFS and CFSA as evaluation function.
- Compare the genotype diversity obtained with the fitness function incorporating more information of the games as average score or number of wins.
- Analyze the results of increasing the β of RIR rate in the co-evolution process using fitness function using Score and Number of Wins.
- Analyze of the results obtained when is introduce neuron immigrants with chromosomes with more diversity with range (0.0, 0.2), (0.0,0.4) and (0.0,0.5).
- Analyze the results when is considered in the fitness functions the previous fitness values obtained by the players that are in Hall of Fame.

The Figure 6.32 shows the genotype diversity of White and Black populations co-evolved using the CFSA and CFS as evaluation functions. In these experiments were executed 4 different experiments for CFSA and CFS. The genotype diversity presented is the average of different four executions. As it can be observed the

initial populations used for both experiments were different, CFSA started with a population more diverse, and CFS with a population less diverse, but at the end of the experiment at the generation 1000 in both cases the diversity decrease to around 0.51. It was executed other similar experiments in all of these cases the genotype diversity of two populations co-evolving, measured as it was discussed in the section 5.11, decrease.

The Figure 6.33 shows the results of co-evolving Black and White player's population using CFSA and CFS but in this case changing the parameter $\beta = 3.0$ for the RIR equation 5.1 with the intention to introduce more new immigrants to both populations. The genotype diversity presented is the average of different four executions. So, it can be observed that moving up this parameter there is a small improvement in the diversity, but still the diversity decrease in this case around 0.52.

The Figure 6.34 shows results of experiments in which the evaluation function used is the one described in the equation 5.10 which consider the number of wins that the players obtained during all competitions in that generation. The genotype diversity is the average of different five executions. In these experiments were used the parameter $\beta = 2.0$ for the RIR rate. It is observed that there is not improvement in the diversity of White and Black player's population using a more complex evaluation function, actually shows a worse result comparing to the previous experiment where $\beta = 3.0$. So, it was decided to use the parameter $\beta = 3.0$ in the rest of the experiments to allow more flow of neuron immigrants to the current population with the intention to improve a little bit the genotype diversity of the population.

The Figure 6.35 shows results of experiments of White and Black's populations co-evolved varying some parameters. The genotype diversity presented is the average of different five executions for each experiment. For example it was experimented using $\beta = 3.0$ and $\beta = 2.0$, and as it was discussed previously the parameter $\beta = 3.0$ produce more genotype diversity.

In this figure is shown the diversity using different fitness functions (FF), using the score or numbers of wins as it is described in the section 5.13.2 and using different values of parameters χ and α. χ which can be used to value more or less the competitive fitness function used (CFS or CFSA) and α which can be used to value more or less the information of the game as number of wins or average score. In some experiments the parameters χ and α were setup initially to (0.1, 0.9) respectively, and were adjusted dynamically moving these values from 0.1 till 10.0 in every 10 generations for the host population (Black) and doing the opposite for the parasite population (White), it means moving the value from 9.0 till 0.0 in every 10 generation. The intention of doing this is to when host (Black) and parasite(White) populations are competing in the same generation the evaluation function will value different these parameters. For example, in

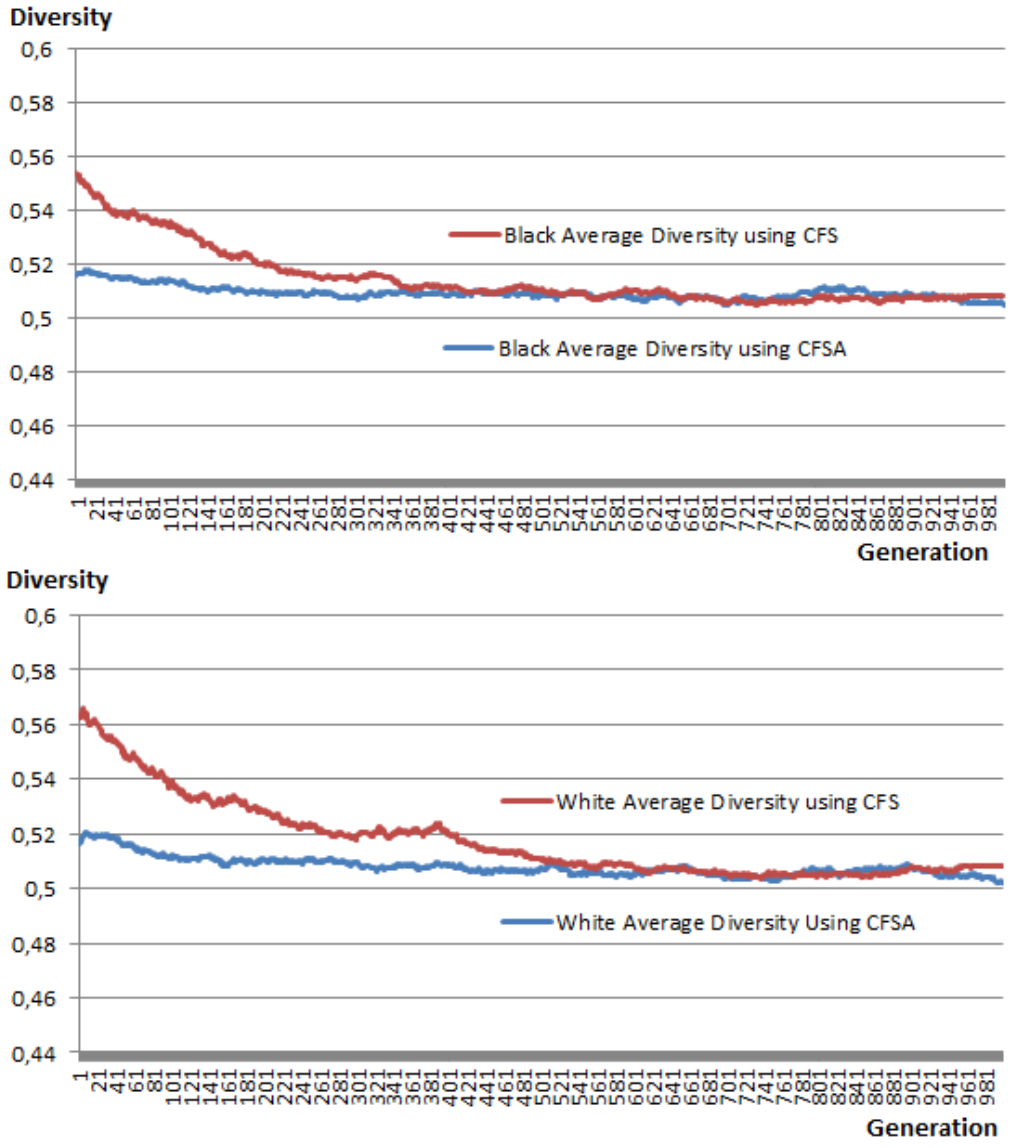


Figure 6.32: Genotype Diversity of the Black and White Populations Co-evolved using CFSA and CFS

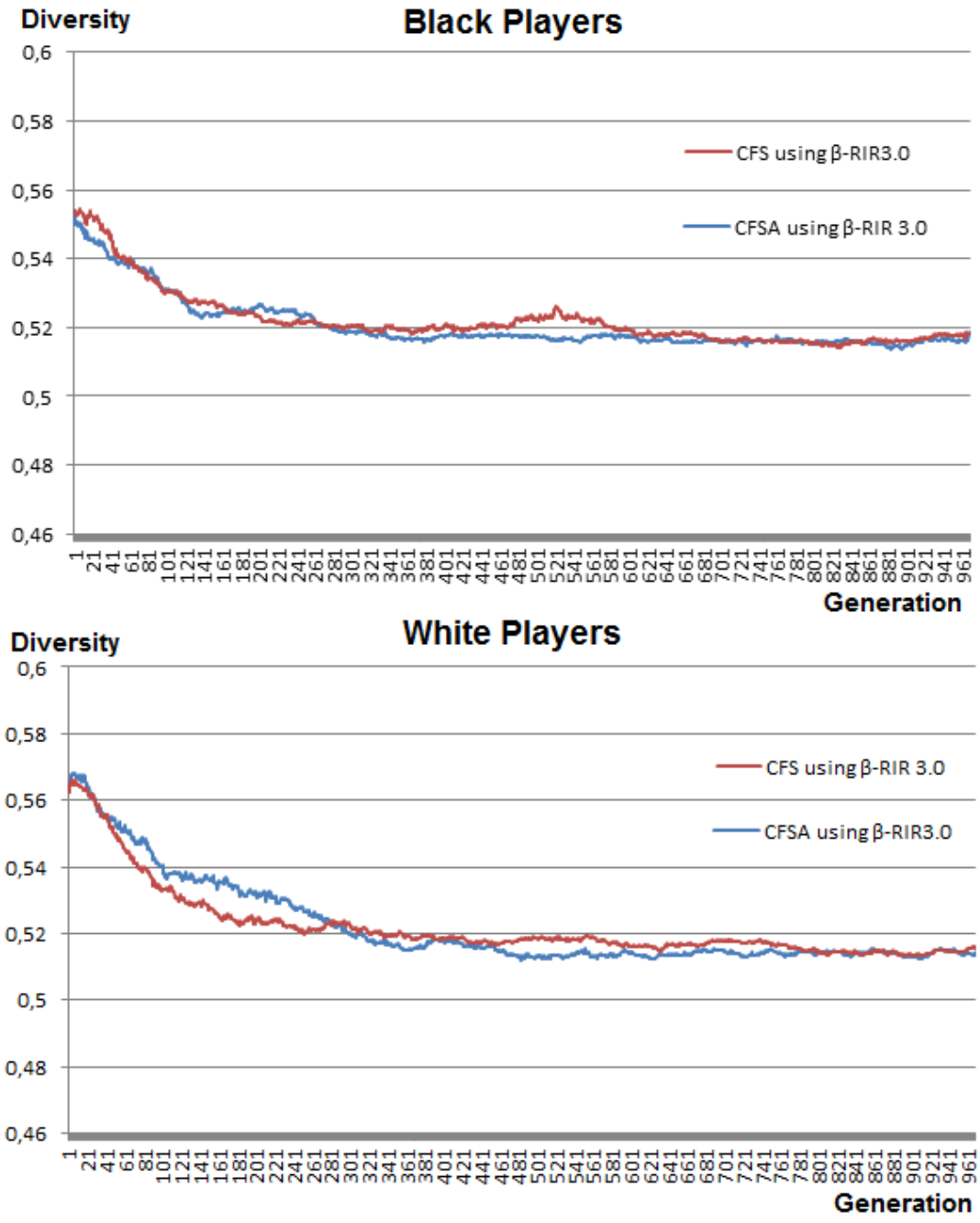


Figure 6.33: Genotype Diversity of the Black and White Populations Co-evolved using CFSA and CFS and B-RIR 3.0

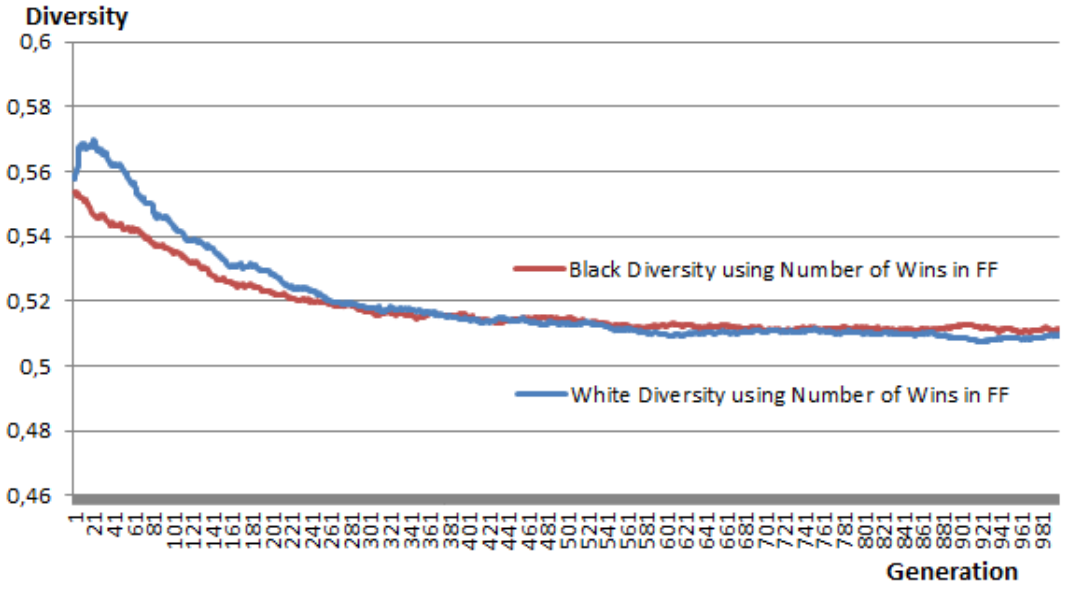


Figure 6.34: Genotype Diversity of the Black and White Populations Co-evolved using Number of Wins in the Fitness Function with B- RIR 2.0

the generation 0, hosts will value for $(\chi, \alpha) = (0.1,0.9)$, meaning valuing more the information of the game, and for parasites $(\chi, \alpha) = (0.9,0.1)$ valuing more competitive fitness sharing used, and so on. So, in the every generation host and parasite will be value the opposite.

In case of experiments where these parameters used in the evaluation functions are (0.5,0.5) means that the these parameters χ and α value the same in all the generations 0.5 and 0.5 respectively for host and parasite players.

The equations containing the Score or the number of Wins can be observed as a multi-objective problem, so, giving different weights to the every objective in different generations and doing the opposite for the opponent.

So, in the Figure 6.35 can be observed that in all of these experiments that the genotype diversity decreases to around 0.52, showing a better result the one which have the combination of these parameters.

In these first four figures of this section has been show experiments where new neuron immigrants had the same range of weights that connect the inputs to the outputs. As it was discussed in the section 2.6.3.3 (description of SANE structure) the weights are represented in the genes of the neurons. Till now these experiments were executed using the range of these weights (0.0,0.1) similar to the range used for the original populations, meaning that new immigrants can have weights with values from -0,1 till 0,1. So, because these previous results it was decided to increase this range in different experiments and observe what is

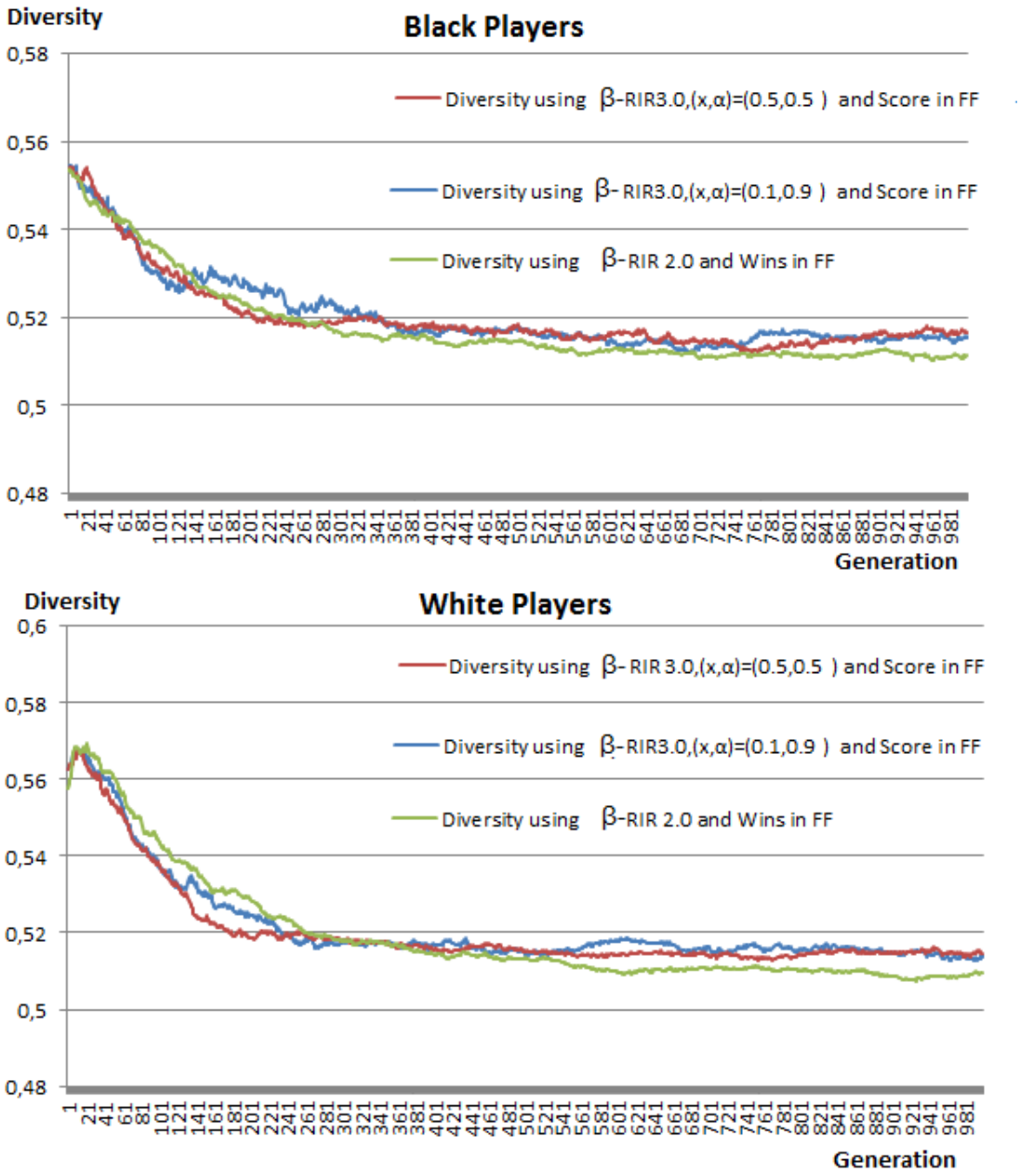


Figure 6.35: Genotype Diversity of the Black and White Populations Co-evolved using Number of Wins and Score in the Fitness Function with B-RIR 2.0 and 3.0

the impact to the genotype diversity of these populations.

The Figure 6.36 shows results of experiments in which it was increased the range of the weights of new neuron immigrants of Black and White neuron populations. So, in these experiments the new neuron immigrants value range of genes of the chromosomes are (0.0,0.2), meaning the weights of neurons connections have values from -0,2 till 0.2. The genotype diversity presented is the average of different four executions for each result.

So, in this figure can be observed that the genotype diversity values trying to growth in the first generations but at the end decrease but in this case to more bigger value 0.55, which it better than 0.52 which were observed when the ranges for the genes were (0.0,0.1). This was a good indication to increase the range of genes in the new neuron chromosomes and observed the results.

The Figure 6.37 show the results of experiments in which were increased the range of these genes of the neuron chromosomes to (0.0, 0.4). The genotype diversity presented is the average of different five executions for each result. In this figure can be observed that it is accomplished the objective to increase the genotype diversity of the neuron population. So, In case of the Black players the genotype diversity grows till generation 400 from there is maintained to around 0.62. In case of White players, the genotype diversity growths more fast till generation 150 and from there is maintained to around 0.62.

In these two experiments for Black and White players were used two different evaluation functions, one considering the Score and the other considering the number of Wins. In both cases the competitive fitness sharing function in the evaluation function was CFSA.

After all of these and other experiments in which some parameters has been changed, the author believe that even introducing more neurons with more diverse genes, there is a point in which the genotype diversity of these neuron populations are stabilized in a particular value. The author believes that one of the reasons why diversity decreases in these experiments could be because of in co-evolution it is used one population for White and Black players and because of genetic operations as crossover are applied apparently to more number of neuron chromosomes of population the genotype diversity decreases. But apparently it looks like that in evolution process against a deterministic player, neurons of players evolved which are part of Hall of Fame are always the same and neuron chromosome replaced applying RIR mechanism are replacing the same worse neuron chromosomes ranked in the bottom, not touching the best chromosomes of Hall of Fame, so for the calculation of genotype diversity are used always new neurons in every generation.

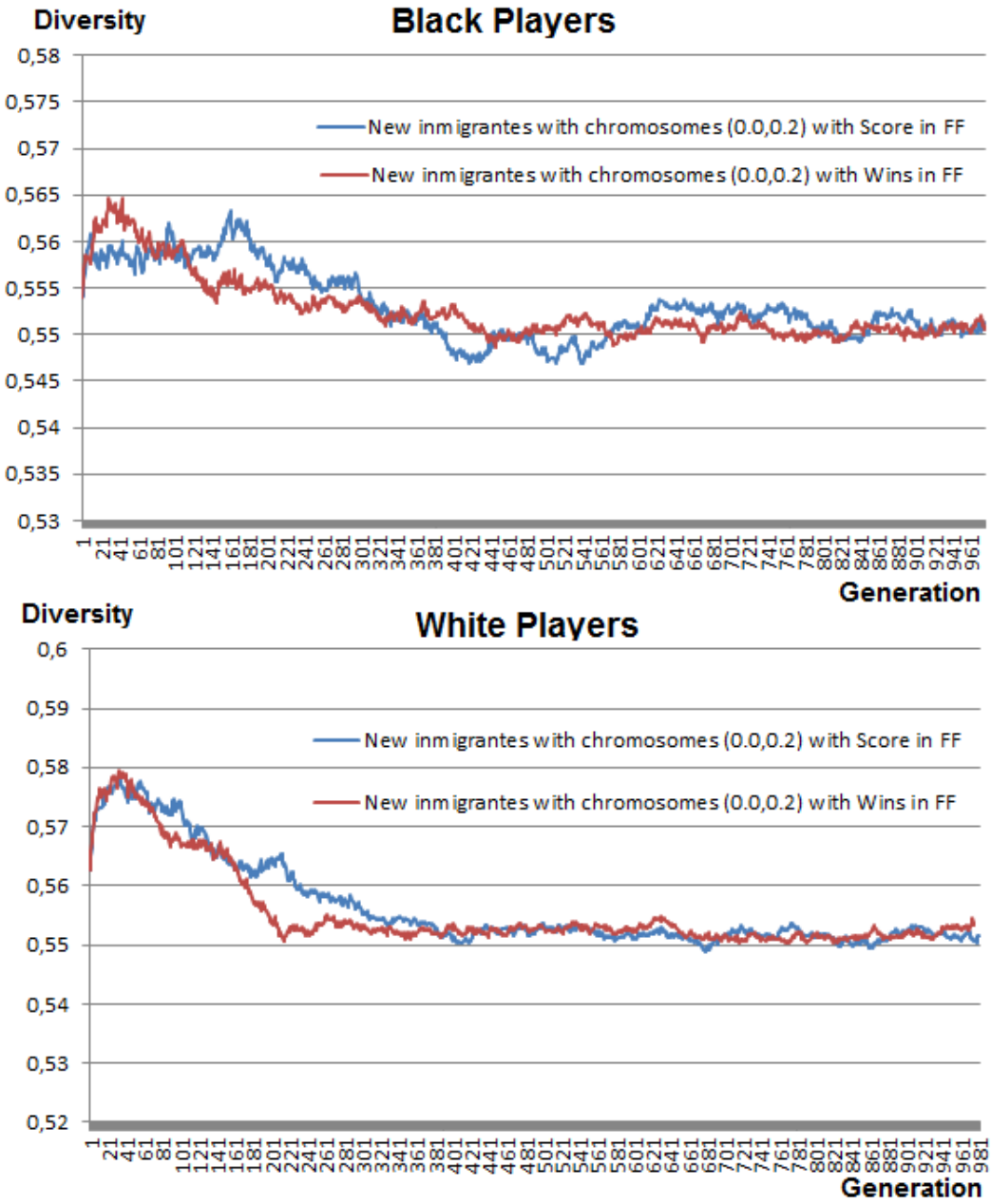


Figure 6.36: Genotype Diversity of the Black and White Populations Co-evolved using Score/Number of Wins in the Fitness Function with B-RIR 3.0 and new immigrants chromosomes in the populations with genes with value range of (0.0,0.2)

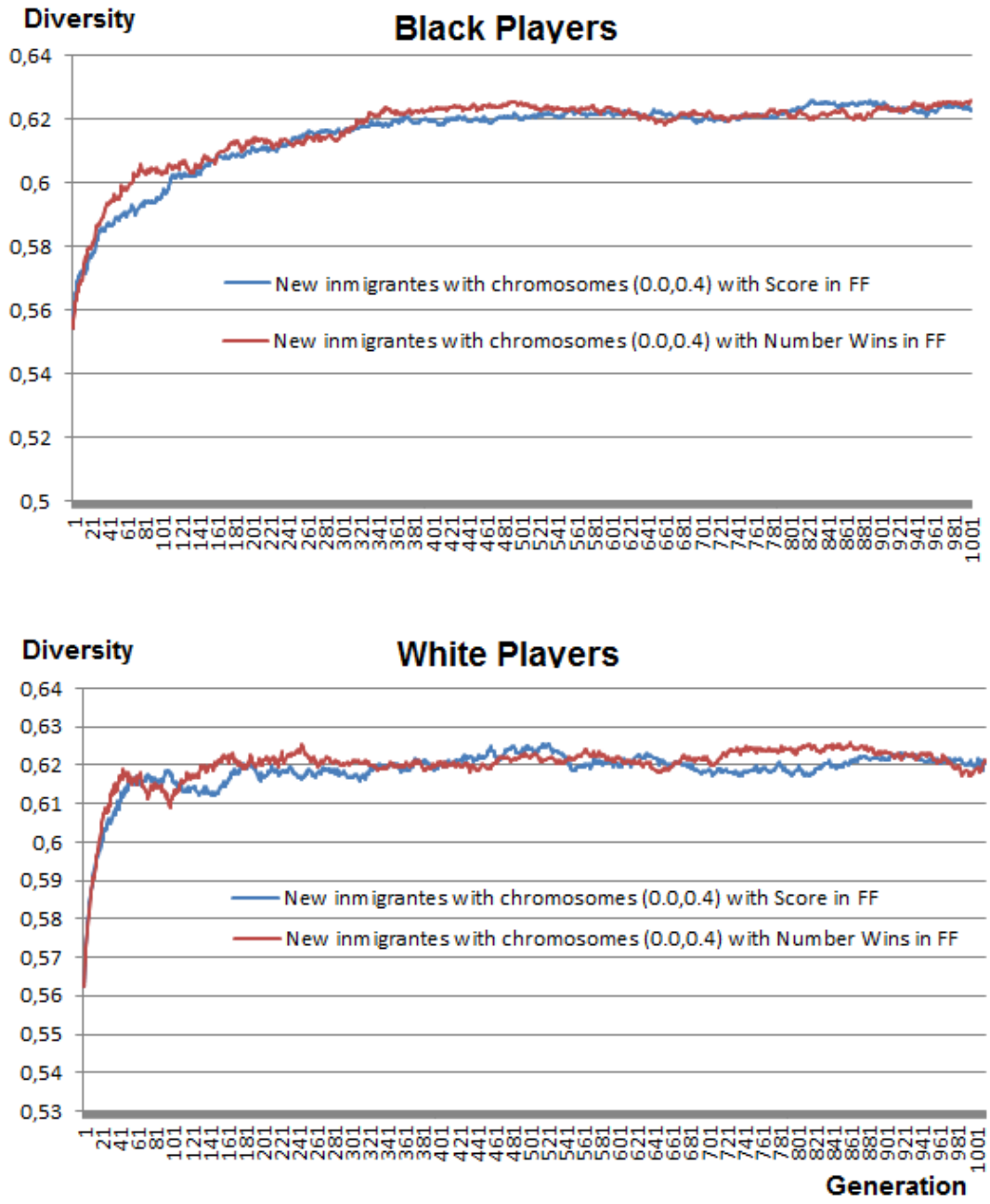


Figure 6.37: Diversity of the Black and White Populations Co-evolved using Score in the Fitness Function with B-RIR 3.0 and new immigrants chromosomes in the populations with genes with value range of (0.0,0.4)

6.4.5 Measurement of the Generalization of the co-evolved strategies

In this section is discussed the generalization of the strategies co-evolved using the methods discussed in this thesis. In this thesis has been introduced different techniques with the intention to increase the diversity and observe how general are results obtained. To measure the generalization were used the best 1800 test samples from 180,000 players created randomly as it was described in the section 5.10.

As it was discussed in the section 3.8.1, using the equation 3.10, it is possible to claim that with 95% confidence and accuracy of 0.023, and because of the absolute difference between the estimate and the true generalization would not exceed 0.002 because of equation 3.6 or 3.7, can be selected 1800 players as test sample.

So, test samples were created randomly for Black and White players which will be used to test how general are the solutions created by these techniques proposed. The followings are measurements of the generalization for players obtained by co-evolution experiments discussed previously:

- Compare CFSA and CFS methods to identify which ones produce more general results.
- Compare the results obtained when is used the score or wins in the fitness function.
- Compare the results when more diversity is introduced with immigrant with chromosomes with genes diverse.
- Compare the results obtained when the players of the hall of fame take in account the result of the previous generation to calculate the current fitness.

The Figure 6.38 shows the generalization of the best Black players of every generation co-evolved using CFSA and CFS. In this figure can be observed the generalization increase from 0.65 till around 0.79 in the generation 350, and from there it looks like it is stabilized around 0.72 and 0.78, and when is reaching the generation 1000 the generalization is around 0.8. In this figure is observed that using CFSA and CFS there is not difference which function create more general strategies.

The Figure 6.39 shows the % generalization in different experiments in which some parameters have been changed for Black players. These experiments were discussed in the previous section, now it is presented their % generalization. So, it can be observed that for any reason the experiment with new chromosome immigrants with gene range (0.0,0.1) which has have less genotype diversity according

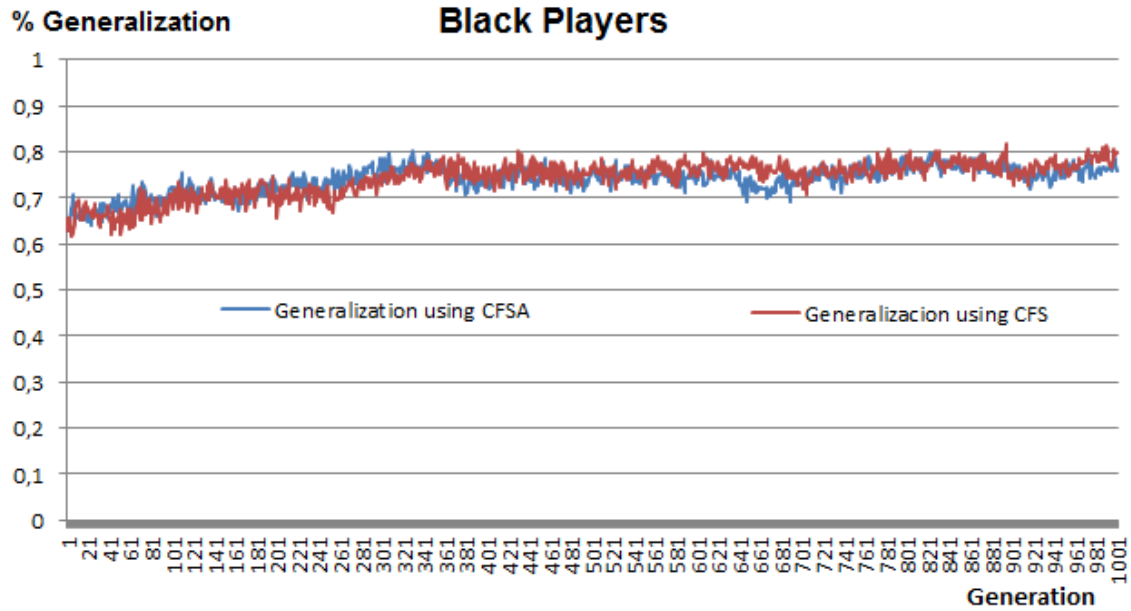


Figure 6.38: Generalization of co-evolved strategies playing black (NicoGoB) using CFSA and CFS

to the results discussed has slightly more % generalization than the other two experiments with gene range (0.0,0.4). So, can be mentioned that more diversity is not necessarily creating more generalization and as it was discussed, this was observed by other authors. But the difference is the generalization is not big enough, so, it is difficult to conclude something in that direction.

Something that probably can be observed from this figure is that in three experiments performed the generalization has a slightly high values when there is an increase in the diversity of the population, but it is not possible distinguish clearly how much this generalization increases. Comparing the generalization using Score and Number of Wins in the Fitness Functions (FF) can be observed that using Score in the FF has a slightly more % generalization.

The Figure 6.40 shows the % generalization for the experiments in which were used new chromosomes with genes range (0.0,0.2) and (0.0,0.4) for Black Players using the Number of Wins in the FF. These experiments were discussed in the previous section but in this sections is analyzed the generalization. So, in this figure can be observed that the experiments with new chromosomes immigrants with gene range (0.0,0.4) has slightly more % generalization than the new chromosomes with gene range (0.0,0.2), actually in some generations, these experiment past 0.80 of % generalization. As in previous figures, the Figure 6.40 shows the average % generalization of the 5 different executions.

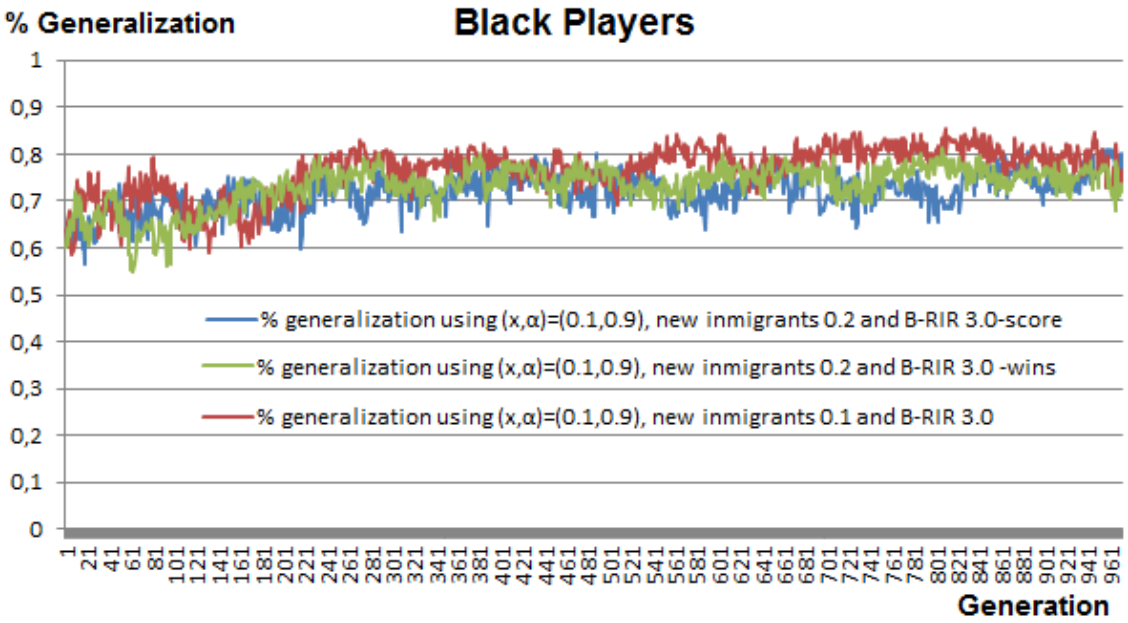


Figure 6.39: Generalization of co-evolved strategies playing black (NicoGoB) using new inmigrantes with chromosomes with genes (0.0,0.1) and (0.0,0.2) and number of Wins and Score in FF

The Figure 6.41 shows the % generalization for the experiments in which were used new chromosomes with genes range (0.0,0.2) and (0.0,0.4) for Black Players using the Score in the FF. As in previous case, this figure shows the average % generalization of the 5 different executions. So, in this figure can be observed that the experiments performed with new chromosomes immigrants with gene range (0.0,0.4) has slightly more % generalization than the new chromosomes with gene range (0.0,0.2) as in the previous case, but in these experiments rarely reach % generalization 0.8.

So, from all of these experiments the author believe that the techniques proposed improve the % generalization during the co-evolution process, but, these are not good enough results because still these results are not consistent, and not crossing in many cases the value 0.8 or 80%.

6.5 Measurement of Global Fitness of Co-evolved Players using an External Agent

As it was discussed previously to measure if co-evolution is progressing, or in other words, measure the global fitness of the strategies evolved using co-evolution, the

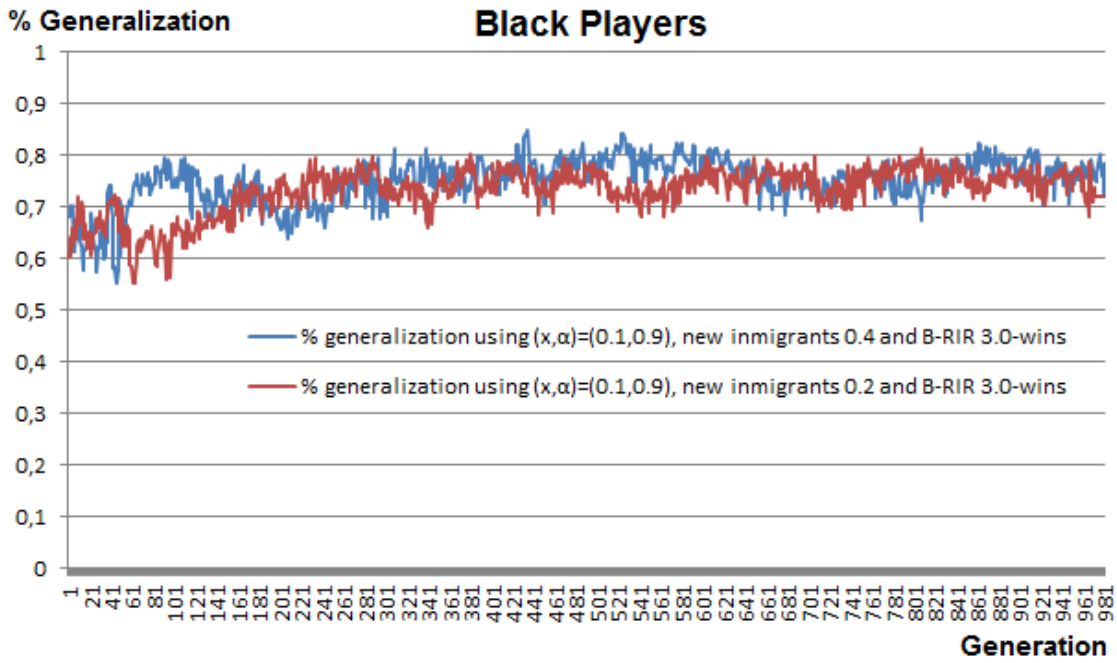


Figure 6.40: Generalization of co-evolved strategies playing black (NicoGoB) using new immigrants with chromosomes with genes (0.0,0.2) and (0.0,0.4) and Number of Wins in FF

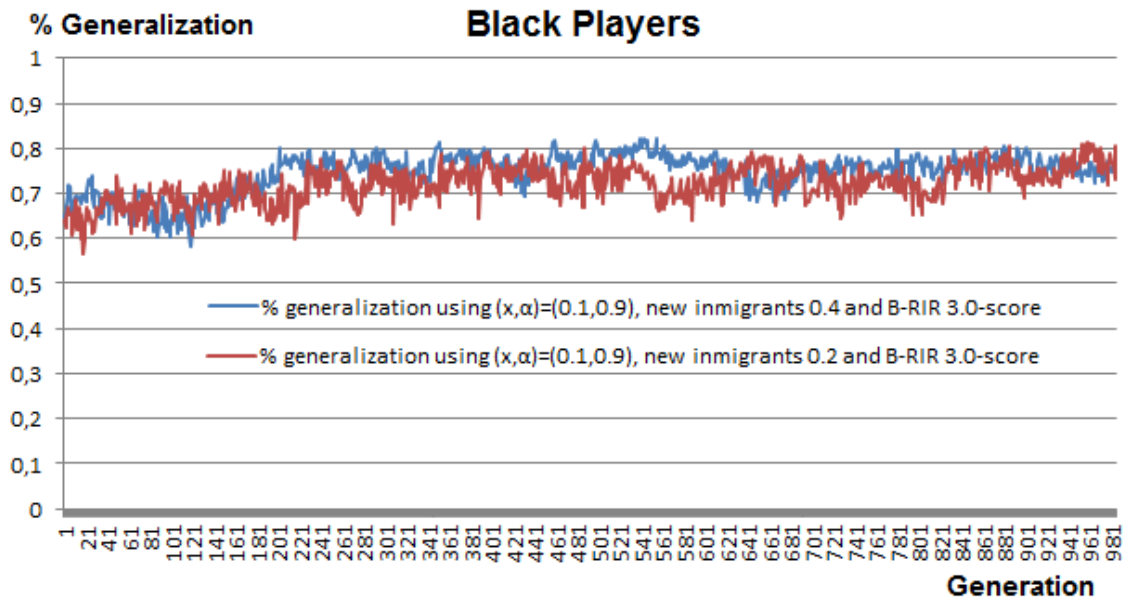


Figure 6.41: Generalization of co-evolved strategies playing black (NicoGoB) using new immigrants with chromosomes with genes (0.0,0.2) and (0.0,0.4) and Score in FF

best players saved from every generation were tested against known computer Go players. In these experiments were used Wally and Gnugo 3.8.

The Figure 6.42 shows the results of competing the best Black players of every generations from the previous experiments competing against Wally varying some parameters of Black players. In this figure can be observed that in general the best players that obtained best results against Wally are the ones which has been calculated more genotype diversity. This is the case of experiments which has the following configuration: immigrant chromosomes with genes with range (0.0, 0.4), using the Score in the FF, with $\beta = 3.0$ from the RIR rate and using the parameters (χ, α) equal to (0.1,0.9).

The Figure 6.43 shows the results of competing the best White players of every generations from the previous experiments competing against Wally varying some parameters of White players. In this figure can be observed,as in the previous case at least till generation 1000, that the best players that obtained best results against Wally are the ones which has been calculated more genotype diversity discussed in the previous section. As in the case of Black players, this is the case of experiments which has the following configuration: immigrant chromosomes with genes with range (0.0, 0.4), using the Score in the FF, with $\beta = 3.0$ from the RIR rate and using the parameters (χ, α) equal to (0.1,0.9).

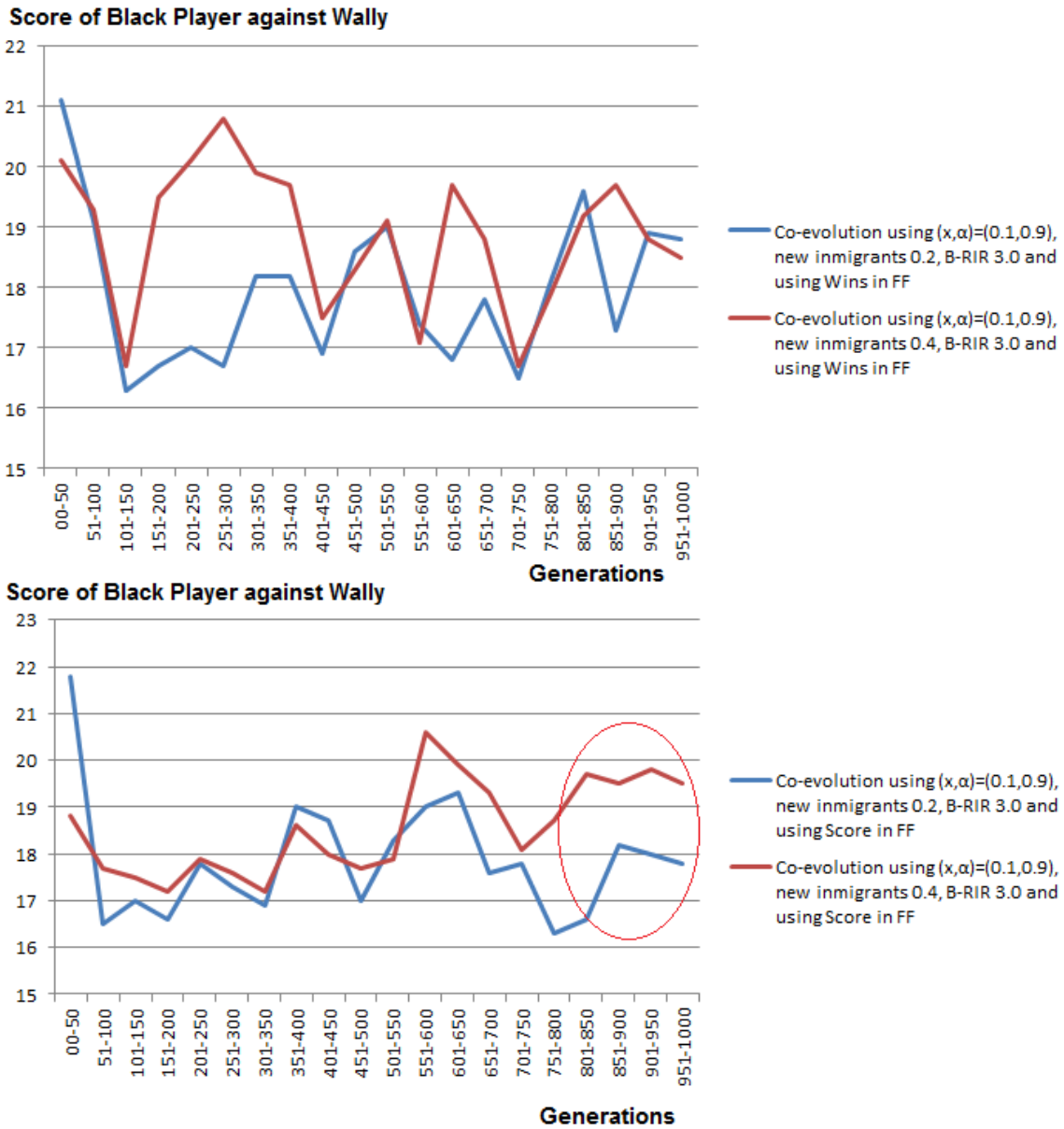


Figure 6.42: Black Players co-evolved introducing neuron chromosomes immigrants with genes (0.0,0.2) and (0.0,0.4) and Score in FF vs. Wally - Average of 5 experiments grouped by 50 generations

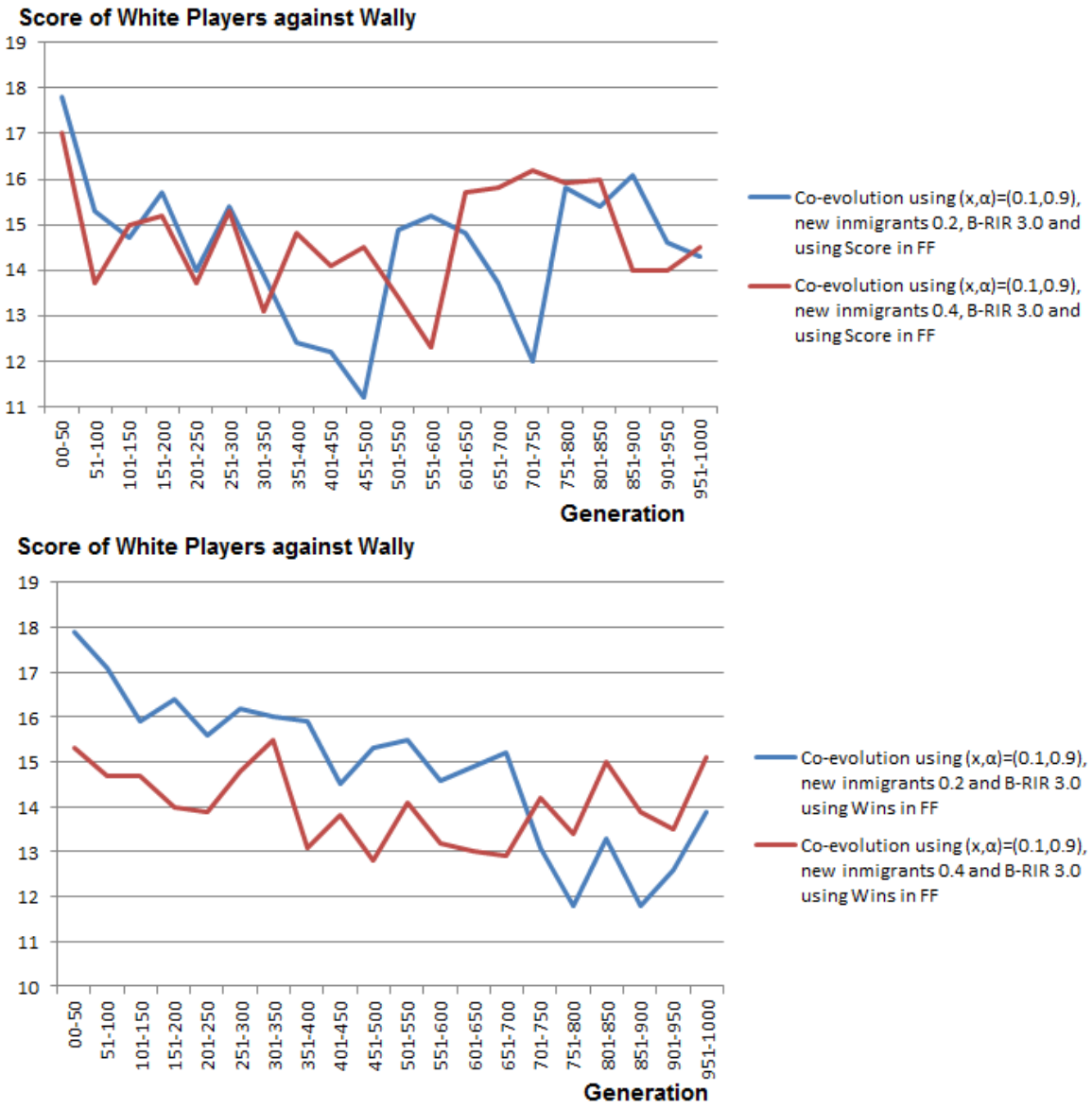


Figure 6.43: White Players co-evolved introducing neuron chromosomes immigrants with genes (0.0,0.2) and (0.0,0.4) and Score in FF vs. Wally - Average of 5 experiments grouped by 50 generations

Table 6.11: Average and Standard Deviation of Scores against Wally of Last 500 best players

Method	Player	Average	Standard Deviation
Chromosomes immigrants with genes (0.0,0.4), Score in FF and $\beta = 3.0$	Black	19.30	2.82
	White	14.65	2.61
Chromosomes immigrants with genes (0.0,0.4), Wins in FF and $\beta = 3.0$	Black	17.89	3.66
	White	14.69	2.61

The Table 6.11 shows the average and standard deviation from the last 500 generations till generation 1000 of the results discussed in the Figure 6.42 and 6.43. From this table can be conclude that the best results against an external agent, in this case Wally, it was obtained by the configuration that had created more genotype diversity in the population at least till generation 1000.

The Figure 6.44 shows the same experiments discussed previously, introducing chromosome immigrants with genes (0.0,0.4), Score in FF and $\beta = 3.0$ but till generation 2100 for Black and White players vs. Wally. This figure, as in previous cases, show the average result of 5 different executions. In this figure can be observed that in both cases the scores of the White and Black players increases till generation 1100 but from there till generation 2100 looks like that score is decreasing in some generations.

Other thing that can be observed is that the global fitness of the strategies evolved by the best White and Black players are decreasing from generation 1100. Apparently which is happening is that the strategies which have been learned by the best White or Black players from the generation 1100 are less complex which is impacting negatively to the learned strategies by the opponent. In other words reduction of the global fitness of the Black (White) players are reducing the global fitness of the White (Black) players. In this figure can be observed that lines of White and Black players from generation 1100 follows similar pattern (marked in red circle).

The Figure 6.45 shows one execution of the experiments discussed in the Figure 6.44. In this figure can be observed that taking in account the fitness function from the previous generation was possible to keep for three generations (34,35,36) a neuron structure that was able to beat to Wally with the score 81:0.

One of the reasons why this happened could be as it was mentioned before, the initial population used for the co-evolution were trained against Wally and by

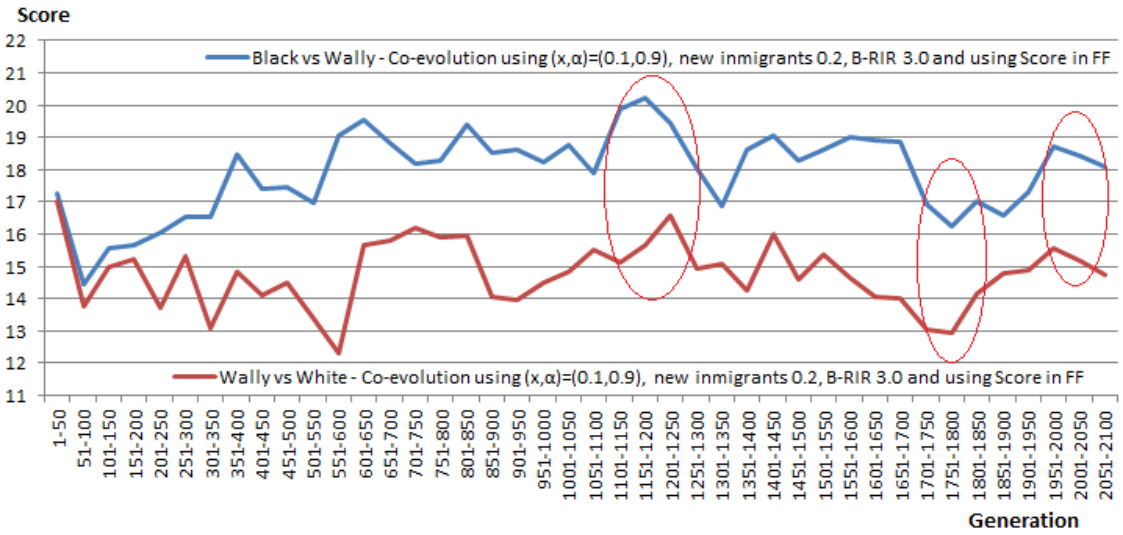


Figure 6.44: Average of Global Fitness - Black and White Players co-evolved introducing chromosome immigrants with genes (0.0,0.4), Score in FF and B-RIR = 3.0 vs. Wally - Average of 5 experiments grouped by 50 generations

coincidence in these generations were selected the combination of neurons that were able to beat Wally 81:0 when was trained.

So, the information how to beat Wally was there in some neurons of the initial population and appears in the generation 34. But, unfortunately this player with this strategy disappear in the next generations from Hall of Fame.

Analyzing the first 7 moves of this player was observed that did not have a good start-game strategy, meaning that started to play in other places different to center of the board, but even that initial strategy this player was able to beat Wally 81:0. The strategy of playing around the center of the boards was observed later in the co-evolution for these experiments.

The Figure 6.46 shows the competition of Gnugo vs best White players co-evolved using NicoGoW using chromosome immigrants with genes (0.0,0.4), Score in the Fitness Function and $\beta = 3.0$. This figure shows the average of 5 executions co-evolved using this configuration. It can be observed that the White players co-evolved are not increasing the global fitness value during the co-evolution till generation 2000. The author believe that there are basically two reasons why it is not showing a progress. The first one is Gnugo is more stronger player than Wally, so, probably it is needed more generations, and second is that probably it is needed to adjust more some parameters as introducing more diversity in the population.

Even these results, it was observed that in some executions there were some

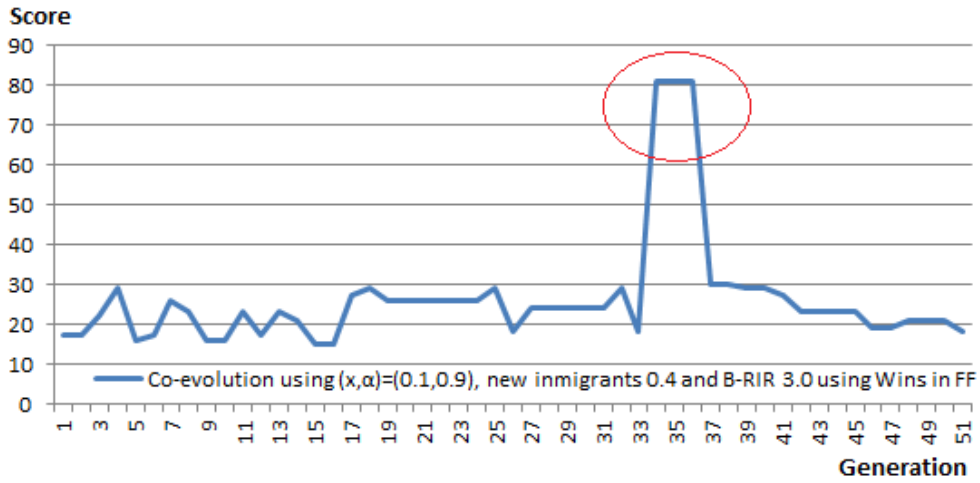


Figure 6.45: Wally vs NicoGoB playing Black - co-evolved with chromosome immigrants with genes (0.0,0.4), Wins in FF and B-RIR = 3.0

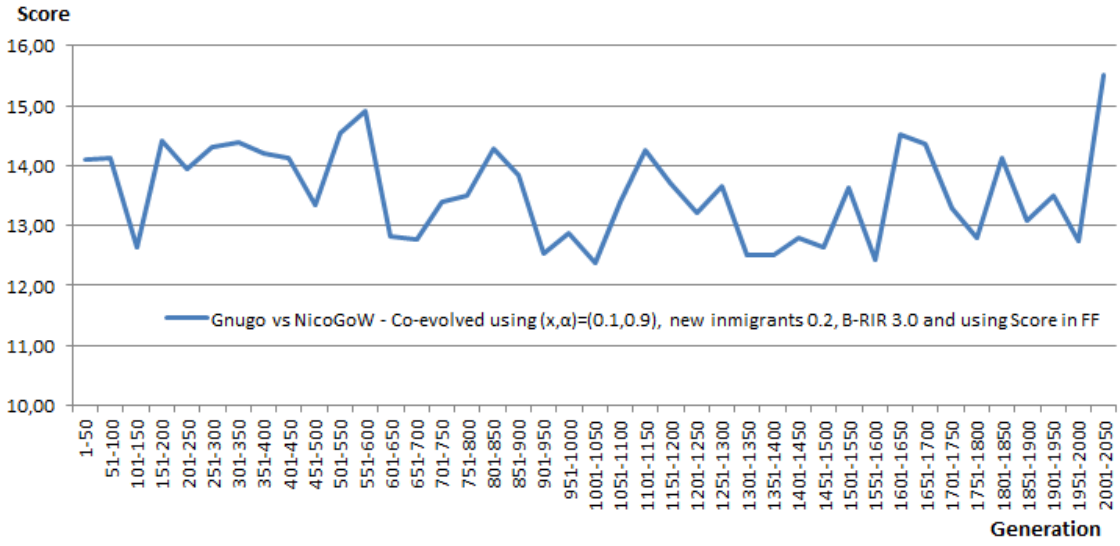


Figure 6.46: Gnugo vs. NicoGoW - co-evolved introducing chromosome immigrants with genes (0.0,0.4), Score in FF and B-RIR = 3.0 - Average of 5 experiments grouped by 50 generations

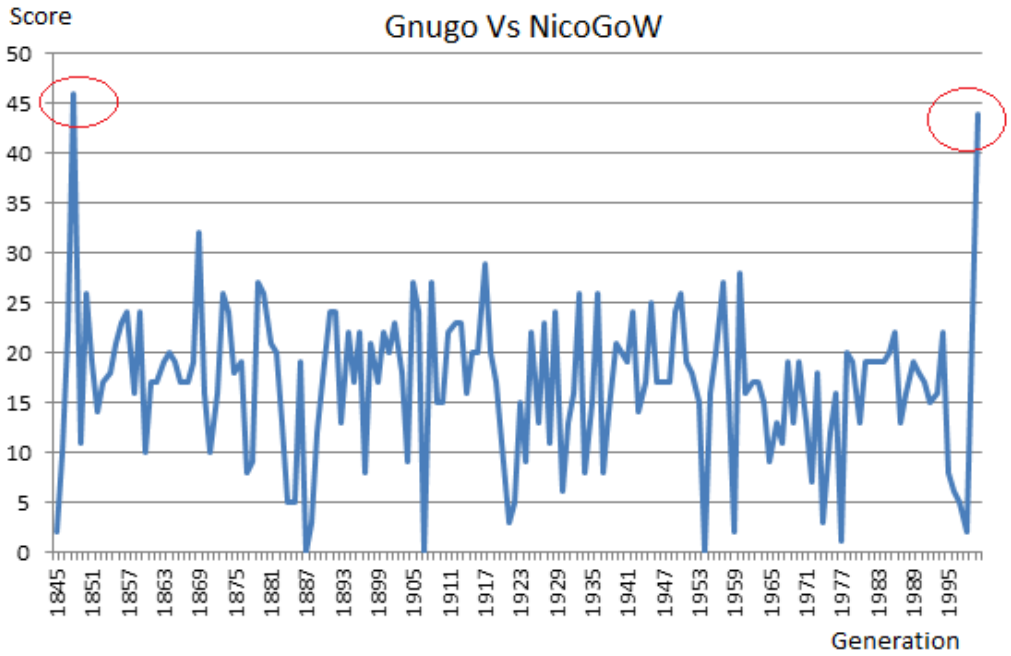


Figure 6.47: Gnugo vs NicoGoW playing White co-evolved with chromosome immigrants with genes (0.0,0.4), Score in FF and B-RIR = 3.0

players of some generations that were able to beat Gnugo as it can be observed in the Figure 6.47 marked in red circle.

In the Appendix A can be observed Figure 8 and Figure 9 from the competition of Wally vs co-evolved players using NicoGoB and NicoGoW.

So, from this section can be concluded that the diversity has impacted positively to the global fitness of the strategies learned, but in case of measurement this against Wally there is point, at least in the experiment performed that from generation 1100 till 2100 (Figure 6.44), that global fitness is not increasing. In case of measuring using Gnugo the global fitness is not increasing at least till generation 2000 performed in these experiments because of Gnugo is a strong player.

6.6 Analysis of Co-evolutionary Pathologies

In this section is discussed whether in all of these experiments using different techniques proposed in this thesis was observed co-evolutionary pathologies described in the previous chapters.

- Loss of Gradient and Disengagement: As it was discussed and presented

some figures in this chapter, using the techniques proposed was not observed Loss of Gradient and Disengagement. There was not a dominant population during the co-evolution process in all the experiments performed. As it was discussed in some figures, in some experiments were observed that in some generations some player (Black or White) obtained more successful results competing against the opponent, but, after some generations the opponent started to obtain better results than the opponent.

This can be observed in Figures 6.18 and 6.23 discussed in the previous sections. In the Appendix A can be observed other experiments with different configurations has shown the same results. So, can be concluded that in these experiments there were not a dominant population and not loss of gradients and disengagements have been occurred.

- Cyclic Dynamics: In the previous chapters has been discussed the red-queen dynamics and cyclic dynamic pathology. In this thesis has been used external agents to measure if the global fitness is increasing while co-evolution is progressing.

  As it was described, the best players of the every generation were saved and were competed against these known computer Go players as Wally and Gnugo. The results were different. In case of the competition against Wally the results discussed previously indicate that there is an increase in the global fitness of the players co-evolved (till generation 1100) but after this generation the global fitness is not progressing. The author believe probably it is needed more generations, i.e. 5000, to observe if the global fitness is increasing.

  In case of the competition against Gnugo the results were different. Gnugo it is more strong player and till the generation 2000 it was not observed an increase of the global fitness of the players co-evolved. What it was observed is that in some generations an specific player was able to beat Gnugo as it can observed in the Figure 6.47.

  So, based on the result of these experiments the author believe that the cyclic dynamics and red-queen dynamics were not observed in these experiments, but still is needed more experiments to conclude that.

- Forgetting: In this thesis was introduced some techniques to avoid that some good strategies are lost because selection methods or by genetic operators. One of this method was the introduction of the blueprint's memory mechanism. Thanks to this method it was able to reinforce some strategies modifying the genes of the best players ensuring that genetic information (strategies) are inherited in the following generations.

The other method introduced to mitigate this pathology was the evaluation functions of players of Hall of Fame can consider the fitness of previous generations. The results of this second method probably has been show less impact than the first method because of as it was discussed before even in some generations were find very good strategies (the ones that beat Gnugo or Wally) but these were not maintained during the co-evolution.

With this section is ending this chapter. In the next chapters is discussed the conclusions and future actions of this dissertation.

Chapter 7

Conclusions

In this chapter is presented the conclusions to this dissertation. The conclusions can be divided in two groups, from the perspective of the techniques proposed and implemented in this thesis, and second from the perspective whether computer Go players evolved have learned very good strategies.

In this thesis was discussed about other approaches to solve games, and Go game. It was discussed different techniques and the state of the art of computer Go game. As it was discussed in the introduction the intention of this thesis was to model some natural process as co-evolution to solve some complex games as Go game.

Co-evolution is a very good technique with many advantage as it was discussed previously but at the same time present some pathologies that need to be solved to ensure that co-evolution is really happening. It was proposed a solution concept to solve this problem and different techniques to ensure co-evolution. The programs coded did not contain any strategies neither were created any set of test cases with Go knowledge to train the Go players, which from other authors perspective this the main advantage compared to other methods discussed in this thesis. So, the followings are the conclusion related to the techniques proposed in this thesis:

- It was show some evidences that cooperative and competitive co-evolution learning process can be produce complex strategies simulating some natural processes. To prove complex strategies has been learned, these strategies evolved were tested against external known computer Go players to measure the global fitness of their evolved strategies which showed good results as discussed. The other proof used to measure if complex strategies were learned was to observed some start game strategies have been created as start playing in the center of the board, which according some professional Go players is very good start game strategy. These start game strategies were obtained using different configurations in experiments performed using

competitive and cooperative co-evolution.

- To ensure that the co-evolution really happens it was needed to applied certain techniques which maintained the diversity of the population as RIR. It was introduced a memory mechanism to keep the good strategies discovered from the past. The co-evolution pathologies were not observed as it was discussed in the previous chapter. The only pathology which were not solved at all was Forgetting. The author believe that other techniques has been introduced to mitigate this pathology.

- Monitor the co-evolution progress or global fitness is very important. In this thesis was used some methods to monitor the co-evolution using an external agent as Wally or Gnugo, and measuring the generalization of the strategies learned. The test against external agents was used Wally and Gnugo and shows good results, beating Gnugo, a strong players, in some generations. To measure the generalization was created some random sample strategies as it was described in the chapter 5. It was observed that the generalization of Go players increased with the evolution, in some experiments reaching 90%.

- It was observed that in co-evolution processes the diversity of the population decreases, and it was needed some mechanism to increase or at least maintain the diversity of the populations co-evolved. In this thesis was introduced some techniques that successful to ensure or at least to maintain some levels of genotype diversity of the evolved populations. The author believe that some reasons why diversity decrease in the initial experiments could be because of we are co-evolving one population for each player and because of genetic operations as crossover are applied to more number neuron chromosomes of population. But apparently in the evolution against a deterministic players, the blueprints evolved which are part of hall of fame are always the same and neuron chromosome replaced applying RIR mechanism, for example, are replacing the same worse chromosomes, not touching the best chromosomes which are ranked always at the top.

- It was found that the generalization of the strategies learned increase during the evolution, but still is not good enough, the author believes that are needed some other techniques to increase the generalization of the strategies learned, some of these techniques are related to maintaining the diversity. As it was discussed, should exist a relationship between diversity and generalization, and whether is not possible to increase or maintain the diversity of the neuron populations is very difficult to create more general solutions.

- It was presented evidences that dynamic sizing of blueprints could find more efficient blueprint networks structures, meaning that networks with less number of neurons can be produce a best strategy beating an opponent with the maximun score 81:0.

The conclusion about whether was obtained a very good computer Go player using these techniques the author believes that this was partially achieved. It is true that was obtained good computer Go players, but, still not good enough to beat a human professional Go player. In some experiments were obtained very good players and strategies, for example, evolved players were able to beat a known strong computer Go player as Gnugo, it was evolved some good strategies as start playing the game in the center of the board, and it was played some games against these players evolved by the the author and other human non-professional players showing good strategies during the games, but even that good results, still there is space for improvement.

The experiments were performed in the majority of the cases till generation 1000, but in the Appendix A can be some results till generation 2000, in which can be observed some improvement as for example % generalization, for instance, in some experiments were reached 90 % of generation which is good result.

The author is leaving the objective to create a very good computer Go player able to beat a professional player for future actions which is discussed in the next chapter.

Chapter 8

Discussion on Future Direction

In this chapter is discussed the future directions to the topics discussed in this thesis. This chapters is divided in three sections. The section 8.1 and 8.2 discuss about the future directions to improve co-evolutionary learning techniques discussed in this thesis to obtain players with more complex strategies, and in the section 8.3 is discussed the application of the techniques discussed in this thesis to computer trading agents.

8.1 Applying Techniques in More Bigger Boards

As it was discussed in the chapter one, in previous works discussed it in which was used the Go game as testbed was used boards less than 9×9, in this thesis the size used was 9×9 which has the same complexity as Chess has. But the same techniques and programs coded for this thesis can be escalated to more bigger board 13×13 or 19×19. The problems to use more bigger boards create some performance in the computing efforts and it is needed a computer with more power.

In the experiments performed in the best machine used, a serve running Linux Ubuntu with 8 processors, it takes the co-evolution of two populations evolved in 1000 generations almost one week, taking in some cases 8 to 10 minutes per generation. Thanks to Juan Pazos the author has obtained access to Magerit, which is the most powerful super-computer in Spain which is located in the UPM university, the intention is to used this supercomputer to continue with the experiments. This computer is formed by a cluster with 245 nodes PS702 each one of them configured with 16 cores in two processors IBM POWER7 (8-core) of 64 bits of 3'0 GHz, 32 GiB of RAM memory and 300 GB of local disk, which is able to reach 103,4 Teraflops of calculus power.

But, before to escalate this board to more bigger size, which still is pending

and can be used the power of Magerit is to run these co-evolution will more bigger generations as for example 10000 or 20000 generations and observe the results. There is some evidence, at least till generation 2000 in which executed some experiments, that can be obtained some good results as % generalization. Actualize the life in the world has been not created in some years, and species that we already knows has evolved some millions of years.

Apart of that, which is pending to test is increase number of members of hall of fame and the number of competitions per generation using Magerit and observe the results if this is impacting positively to the co-evolution of more complex strategies.

8.2 Improve the Techniques Proposed

In this thesis was proposed different techniques to ensure that co-evolution is progressing and increase global fitness of strategies searched. As it was discussed previously, even co-evolution pathologies were not observed, while testing strategies learned against external agents, it was concluded that still there some improvements pending to perform to these techniques.

One of the improvements is to ensure that it is possible to increase the generalization of the strategies learned, introducing more diversity or applying other techniques.

The memory mechanism introduced some improvements to evolve players in terms of maintaining some strategies , but, still it was observed some good strategies has been forgotten. This is not a contradiction to the previous statements saying that not co-evolution pathologies as forgetting was observed, but the point is that even some strategies were maintained probably can be created another blueprint structure in which some strategies can be maintained forever and avoid to be forgot because of the pressure of some selection methods or genetic operations in the population. The big issue in this approach is to identify what is good strategies or which good strategies should be kept. In a co-evolution process "good" is relative because depend on the level of the opponents which are evolving.

Another improvement to the structure used is to add another level of blueprint, called blueprint of blueprints, which can collect and be used in any generation the knowledge of more than one player (i.e. best players of Hall of Fame) and not only from one player as it is done in this thesis. Another improvement to the neural networks structures used in this thesis is the implementation of these networks using two levels of hidden neurons which can capture another level of abstraction of the game.

8.3 Other Applications to the Co-evolutionary Techniques Proposed

The application of techniques proposed in this dissertation is diverse as it was discussed in the thesis. The author has already started to apply these techniques in the trading of securities in the financial sector, but still some issues than should be solved as to increase of % generalization of the strategies learned.

Apart of that, these techniques can be applied in other board games, and general in different computer games were it is difficult and very expensive to code the knowledge from experts. Even these techniques can be applied in security industry, for example in cyberwars or cyberwarfare where some organizations as nations, companies and others can create any piece of software as virus, worms,etc and their antidotes can evolve interacting with their opponents. Cyberwarfare was defined as actions by a nation-state to penetrate another nation's computers or networks for the purposes of causing damage or disruption Clarke [2010].

So, the application of these co-evolutionary learning process as it was discussed in this thesis with some examples from other authors is very diverse and applicable to any real-world problem in which traditional approaches are not feasible because it is almost impossible to create absolute quality measure of solutions that are required for these search algorithms.

8.3.1 Application to Security Trading

In the chapter 4 was presented the application of co-evolution to a real-world problem as security trading, this section is introducing some ideas how to implement the techniques discussed in the thesis.

In the Figure 8.1 can be observed the architecture proposed by the author to evolve trading agents that compete in a fictitious capital market. The inputs for the agents are the indexes that can be captured from any sources as for example information that come from Blomberg or others. The Agent transform these indexes to technical market indicators (TMI) which are the inputs to the neural network structure.

The neural network has many inputs as many TMI used to evolve, and one output which is the trading action (BUY, SELL, CUT, Not action). The Agent will be formed by one blueprint which has many neurons from a neuron population which will evolve. Every agent has their own neuron population.

The trading agents will be a compete in a fictitious market in which different agents will perform some actions, at the end of every time during competition (time depending on the times series used, i.e. minutes, hours, days, weeks) and will calculate the fitness based on the benefit that can obtain every agent. The

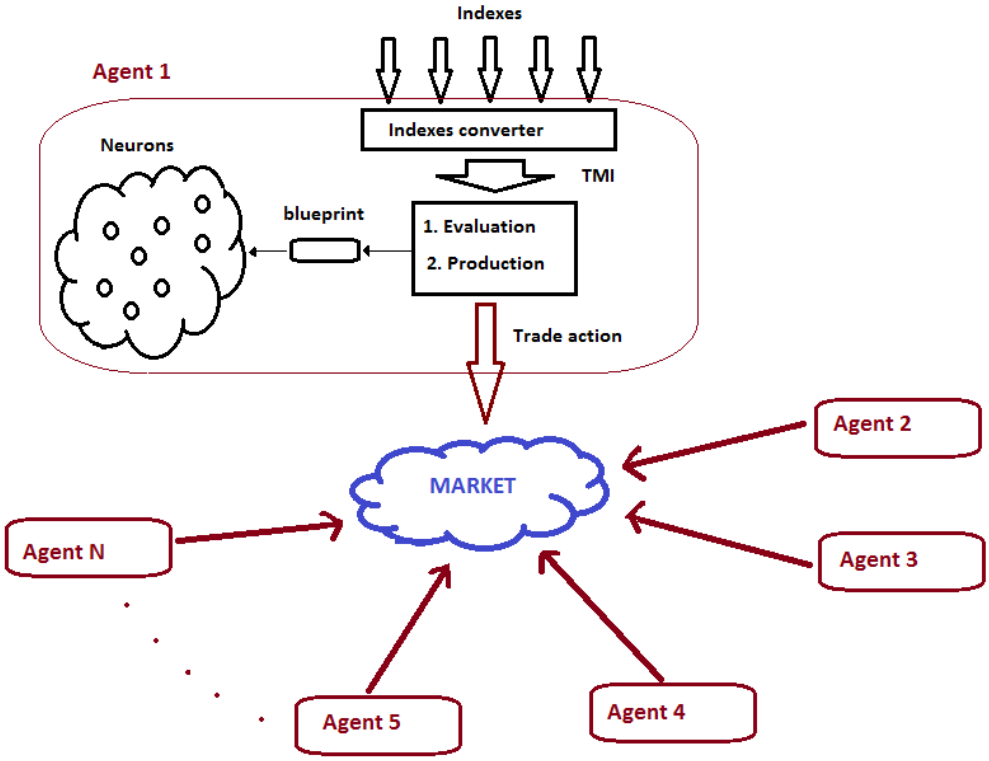


Figure 8.1: Overview of the architecture proposed to evolve Trading Agents in Capital Market

fitness function of every agent which can be used is the equation 4.13 described in the chapter 4 based on P&L (Profit and Loss) and SR (Sharpe Ratio).

Every generation will contain different time windows. For example if there are time series for 5 years and these are daily data, for training can be used data for first 4 years, and the last year to evaluate the agents trained. So, during the training using the first 4 years data, the first generation for example can start in day 1 and end at day 200 (first 200 days), the second generation will start in day 2 and end at day 201 (next 200 days) and so on. Finally, not data used for training can be used to evaluate which best agent are doing more profits.

In the Appendix B is the other TMIs which can be used to transform the indexes into the inputs to trading agents.

Appendix A

The following figures shows some results to the experiments discussed in the previous chapters.

.1 Results of Computer Go Players Co-evolved from Some Experiments

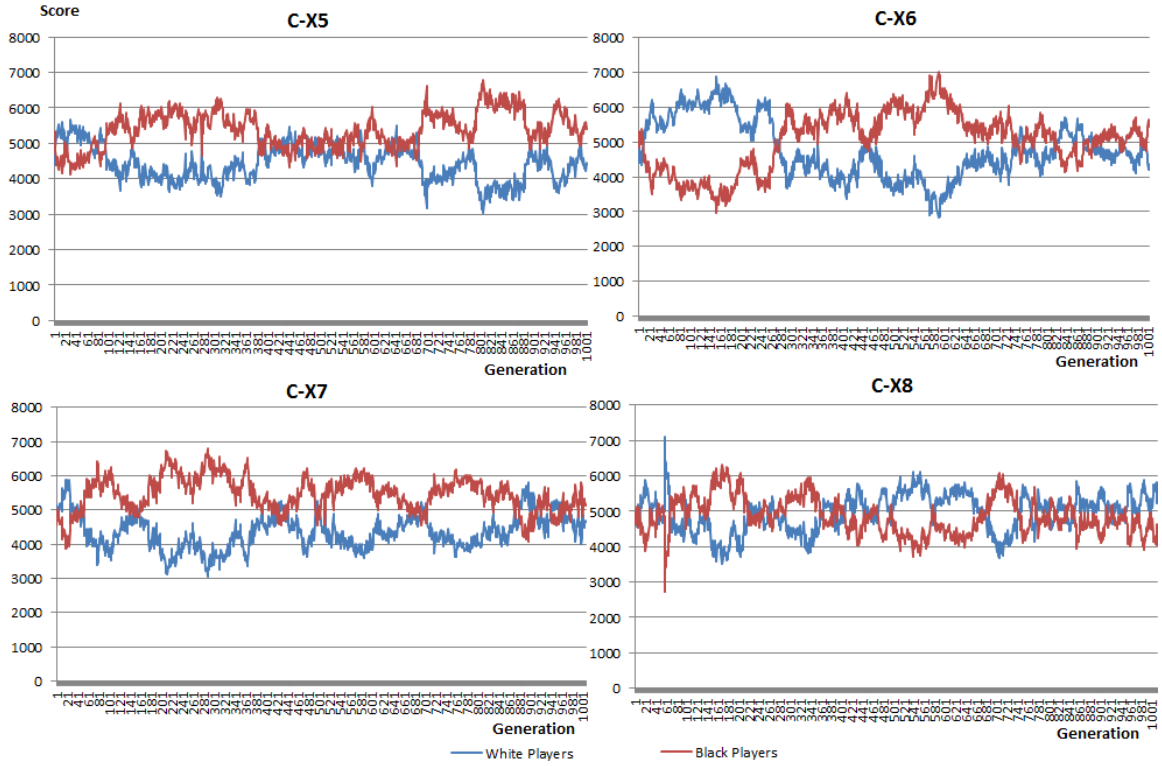


Figure 2: Number of games won by White and Black players during co-evolution in 10000 competitions in every generation using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Score in FF and (x,a) = (0.1,0.9)

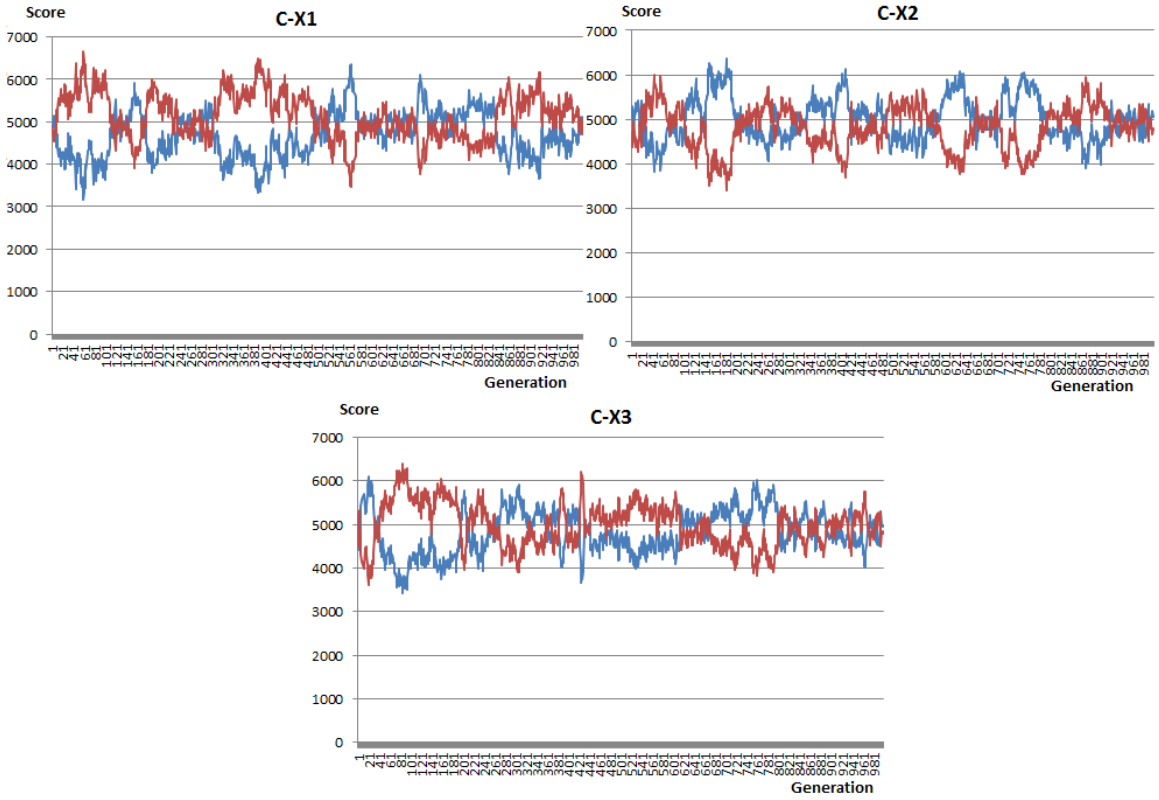


Figure 3: Number of games won by White and Black players during co-evolution in 10000 competitions in every generation using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Number of Wins in FF and (x,a) = (0.1,0.9)

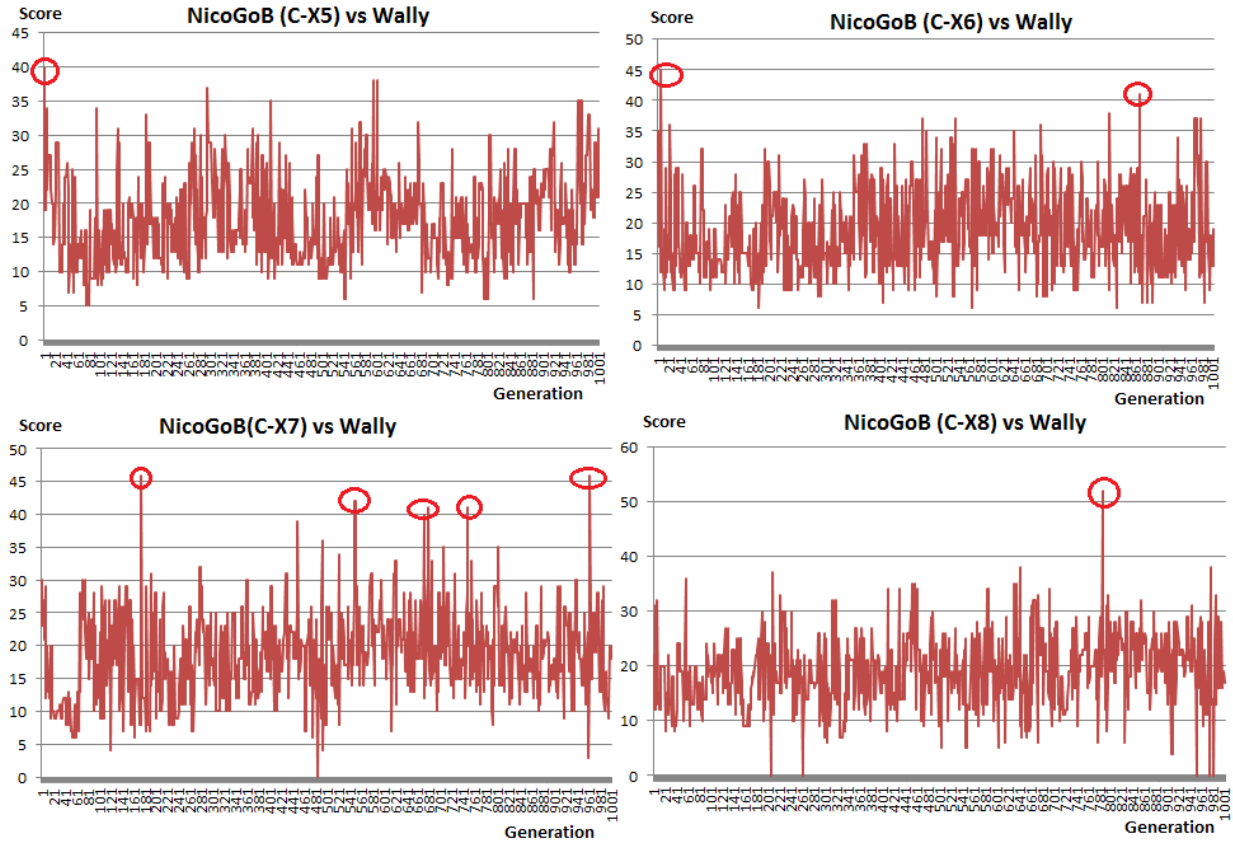


Figure 4: Results of competing Wally vs best Black player of every generation co-evolved using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Score in FF and (x,a) = (0.1,0.9) - Circled in red best Black players beat Wally

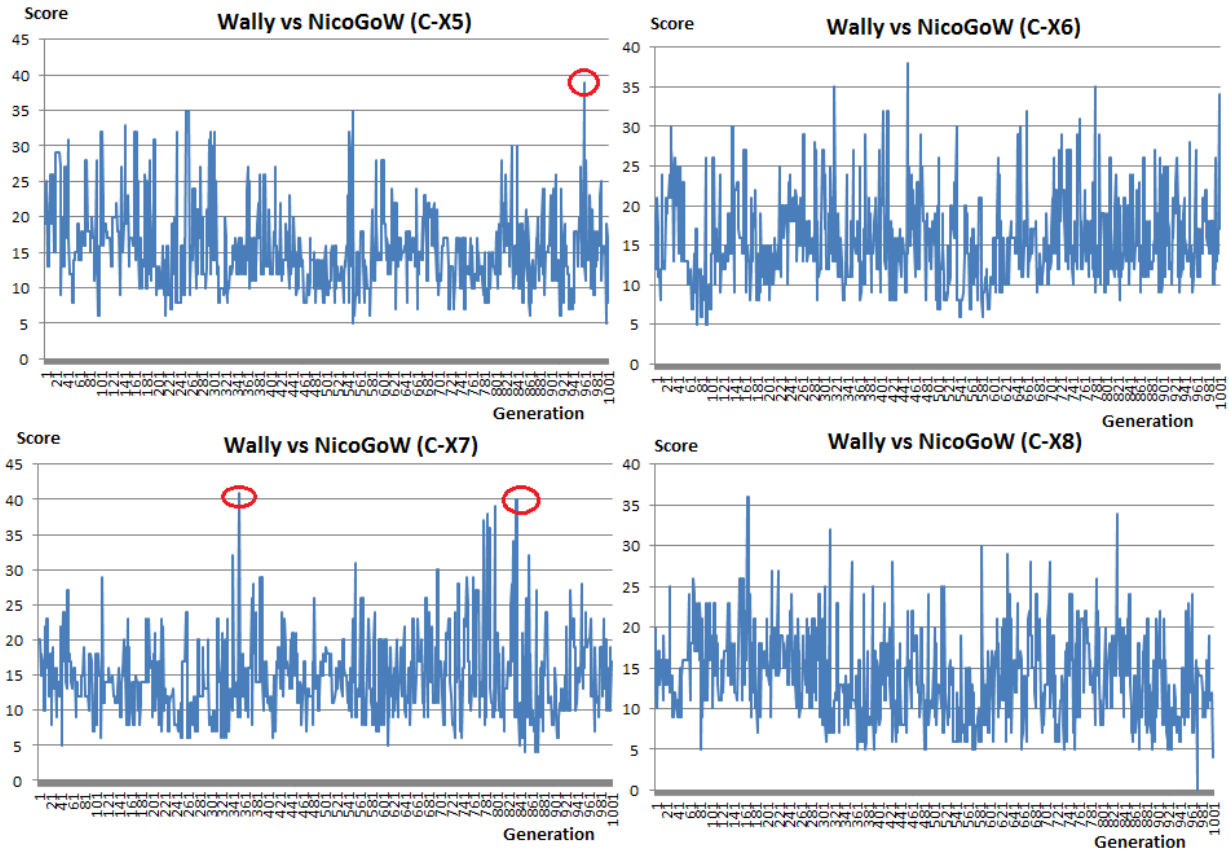


Figure 5: Results of competing Wally vs best White player of every generation co-evolved using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Score in FF and (x,a) = (0.1,0.9) - Circled in red best White players beat Wally

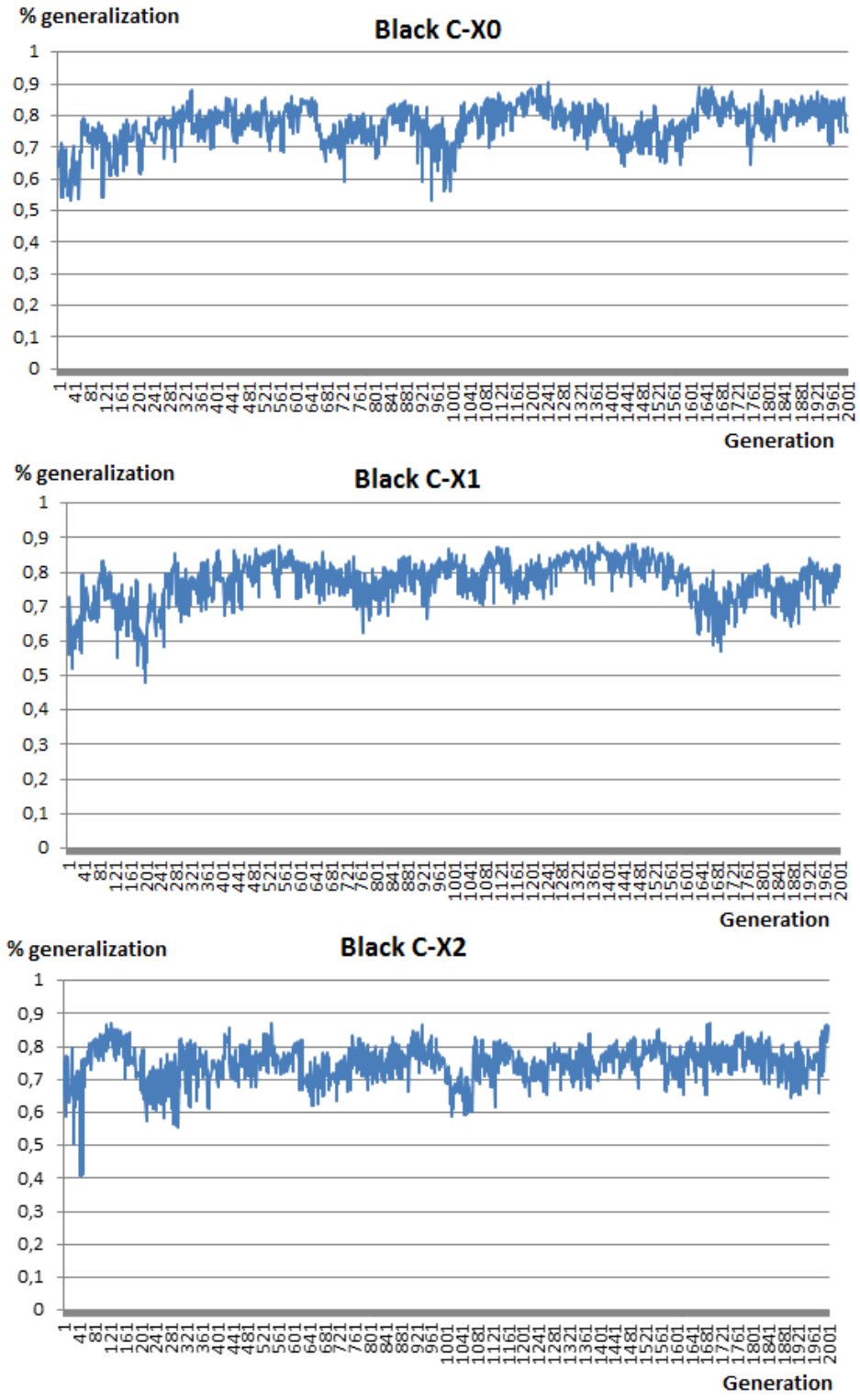


Figure 6: % Generalization of Black players from experiments C-X0, C-X1, C-X2. Best Black players were co-evolved using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Number of Wins in FF and (x,a) = (0.1,0.9)

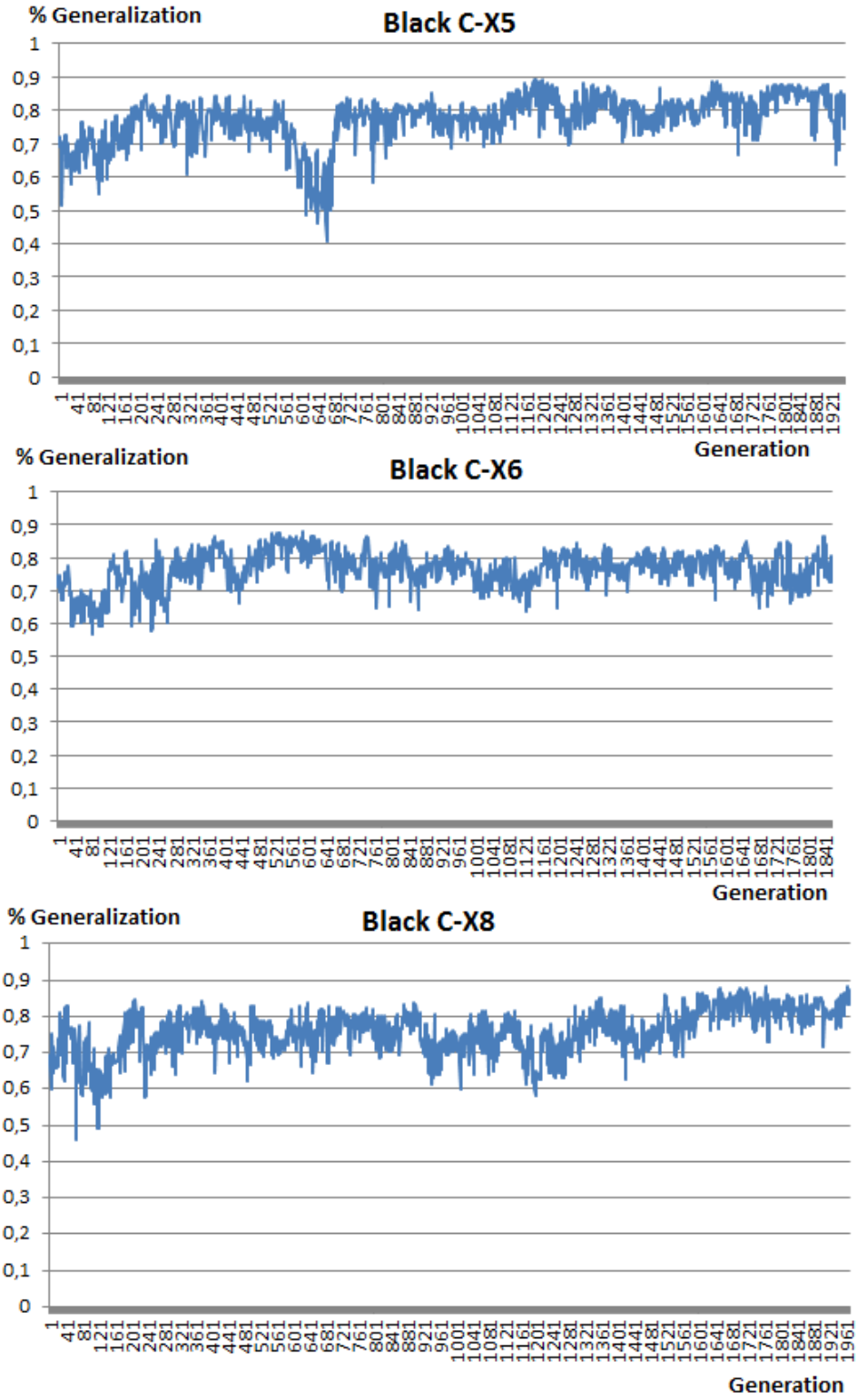


Figure 7: % Generalization of Black players from experiments C-X5, C-X6, C-X8. Best Black players were co-evolved using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Score in FF and (x,a) = (0.1,0.9)

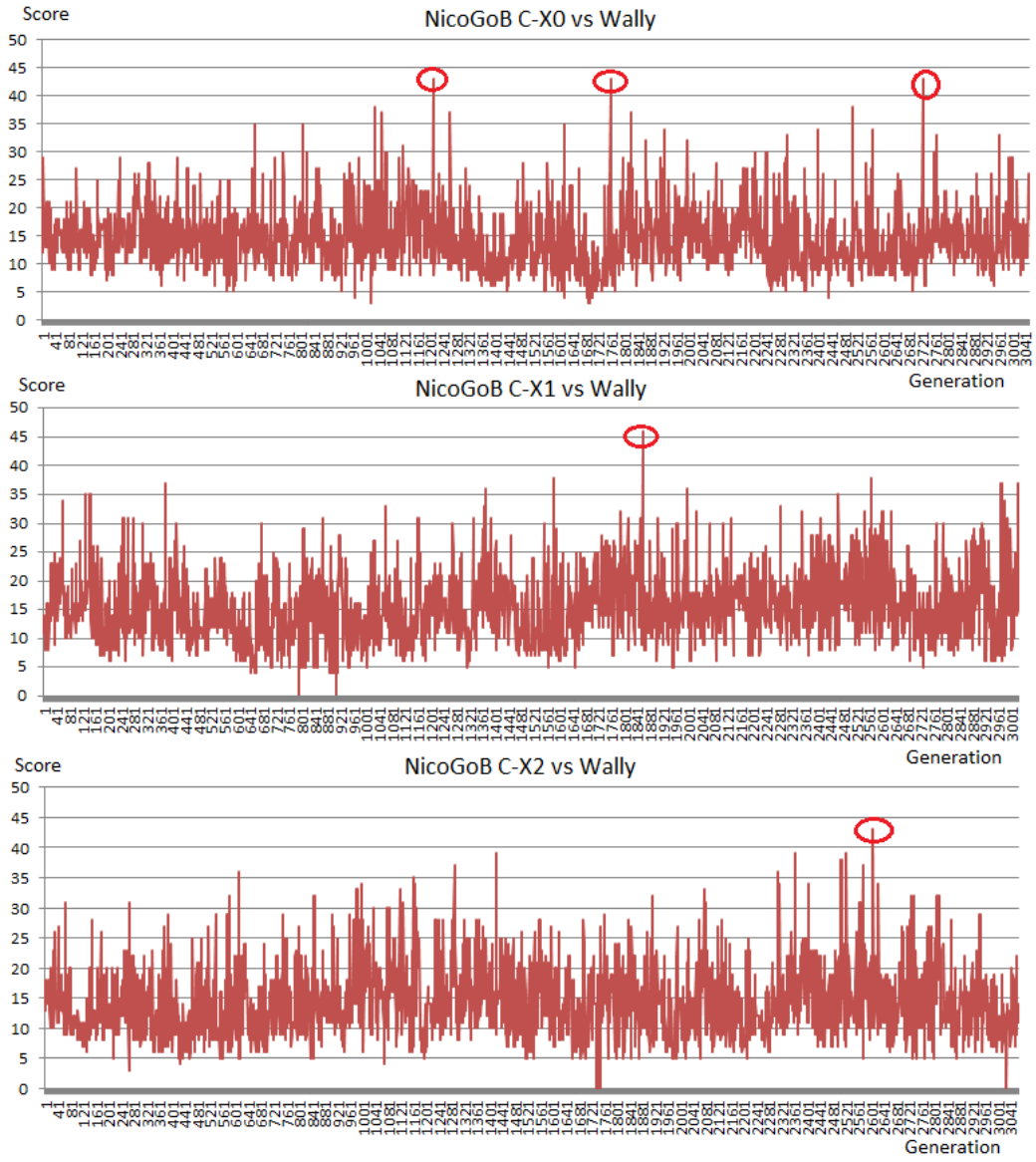


Figure 8: Competition of NicoGoB vs Wally from experiments C-X0, C-X1, C-X2 till generation 3000. Best Black players were co-evolved using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Number of Wins in FF and (x,a) = (0.1,0.9)

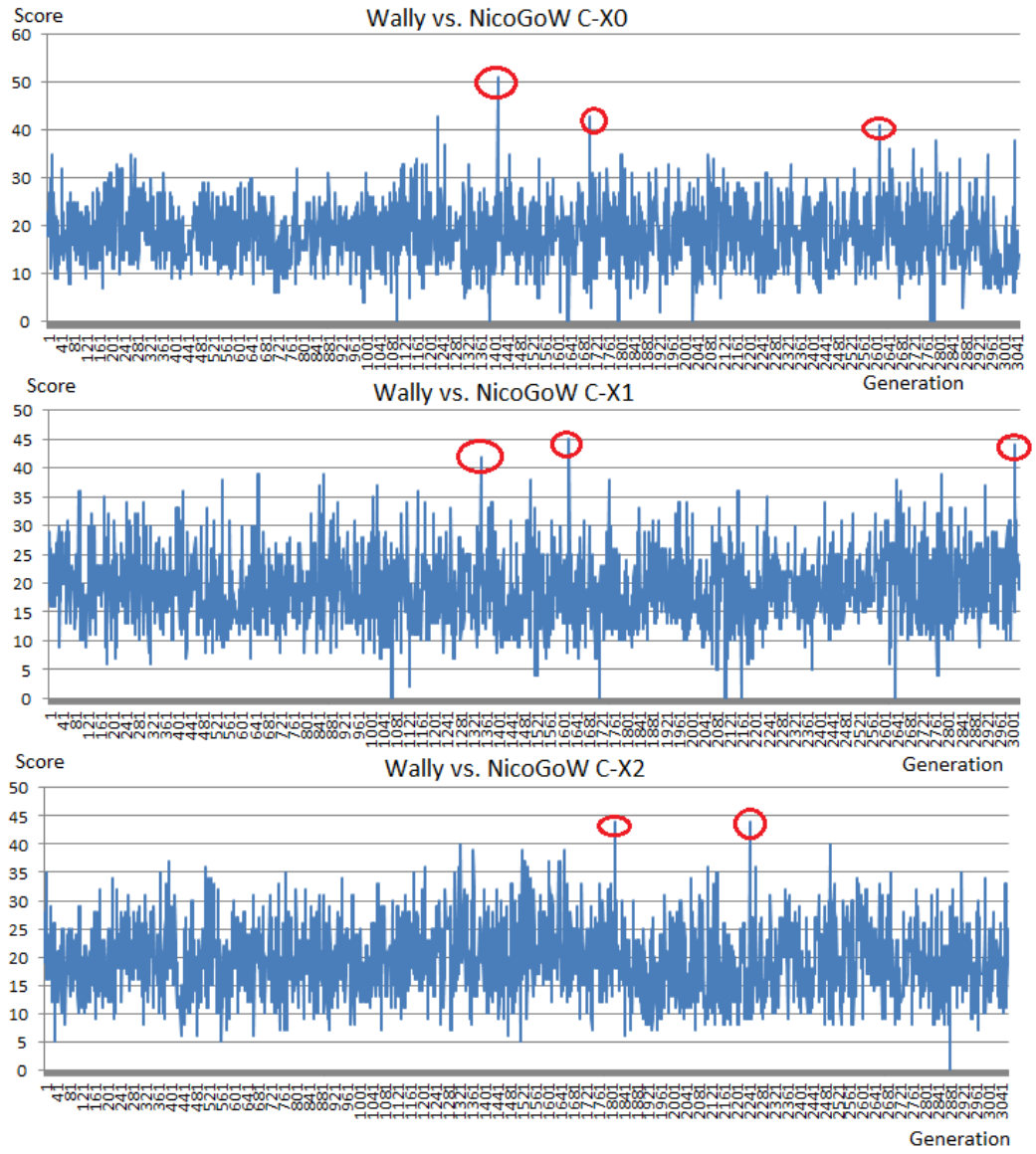


Figure 9: Competition of NicoGoW vs Wally from experiments C-X0, C-X1, C-X2 till generation 3000. Best White players were co-evolved using the following configuration: neuron chromosomes immigrants with range (0.0,0.4), B-RIR 3.0, Score in FF and (x,a) = (0.1,0.9)

Appendix B

This appendix is not showing all source code, just some sections relevant to understand the some techniques discussed in this thesis.

- Definition of neuron structure used in this thesis

```
typedef struct {
   float gene[GENE_SIZE];     //neuron's chromsome
   int in_conn[GENE_SIZE/2];  //inputs to network
   int out_conn[GENE_SIZE/2];  //outputs to network
   int numin ;
   int numout;
   float in_weight[GENE_SIZE/2];  //connection from inputs to
   hidden layer
   float out_weight[GENE_SIZE/2]; //connection from outputs
   from hidden layer
   float in_delta[GENE_SIZE/2];
   float out_delta[GENE_SIZE/2];
   char decoded;
   float fitness;     //neuron's fitness value
   int tests;
   int ranking;      //where neuron ranks in population
   int type;
   double sum;
   float sigout;
   char output;
   float error;
} neuron;
```

- Definition of network structure or blueprint. This structure will keep pointers to the neurons of the network used in SANE.

```
typedef struct {
 int nhidden;       // hidden number dynamic
 float input[NUM_INPUTS];
 neuron *hidden[NUM_HIDDEN];   //pointers to neurons
 float sigout[NUM_OUTPUTS];
```

```
  float sum[NUM_OUTPUTS];
  int winner;  /*which output unit was highest*/
//memory of the sequence of moves per player
  int out_moved_player[NUM_OUTPUTS*2];
  int out_moved_opponent[NUM_OUTPUTS*2];
  int nmov;  //number of movements in the game
//used for the memory in coevolution
  int gamewin; //if the game was won
} network;
```

- Definition of best nets structures. This structure will keep pointers to the best networks of the previous generation. This is the genetic description for the network level evolution.

```
typedef struct {
  int nbhidden; //hidden number dynamic
 neuron *hidden[NUM_HIDDEN]; // pointers to neurons
  float fitness;
//sequence of moves per player
  int o1_moved_player[NUM_OUTPUTS*2];
  int o1_moved_opponent[NUM_OUTPUTS*2];
  int n1mov;  //number of movements by player in the game
//For the use of memory in coevolution
  int o1_moved_cplayer[NUM_TRIALS][NUM_OUTPUTS*2];
  int o1_moved_copponent[NUM_TRIALS][NUM_OUTPUTS*2];
  int n1movc[NUM_TRIALS]; //number of movements by player in
    the game
  int gamewin[NUM_TRIALS]; //save which games not lost
  float ifitness[NUM_TRIALS]; //individual fitness for each
    game
} best_net_structure;
```

- Create blueprints with dynamic size (or different number of neurons)

```
 srand ( time(NULL));
 net.nhidden=randint(NUM_HIDDEN_MIN,NUM_HIDDEN_MAX);
 //select nhidden neurons from population to form blueprint
 for(i=0;i<net.nhidden;++i)
 {
  // it takes a neuron from the neuron population randomly
    j = randint(0, POP_SIZE-1);
    net.hidden[i] = popu[j];
    if (popu[j]->decoded == 0)
    gene_to_weights(popu[j]);
}
```

- This section show how the NicoGoB (or NicoGoW) is providing a move to OpenGo framework

```
ErrId Sane::cbGetMove( Move &OppMv, const DataBoard *pBd )
{
   ErrId err = ERR_NONE;

   // Check if move supplied is from the opponent, the first
   move of the game, or if
   //  last move was rejected.
   if( OppMv.GetMoveType() == MT_None )
   {
      // rejected or first move!
        theSane.AddRejectedMoves(r_saved,c_saved);
     // reduce the counter of the memory
          theSane.updateMemory(i_output_player,
   i_output_opponent,0);
       int x = 1; // catch for debugging
       ++RepeatedRejects;
       //if( RepeatedRejects > 2 ) {
     if( RepeatedRejects > MOVES_REJECTED ) {
               // resign
                _SaneMv.Set( MT_Resign, GetPlayerColor(), 0, 0)
   ;
               // post results back...
               Post_MoveRsp( _SaneMv );
               return ERR_NONE; // no error
          }
   }
   else {
       RepeatedRejects = 0; // clear
       theSane.ClearRejectedMoves();
       // Give Sane the opponent's move...
       if( (err=SetMove( OppMv)) )
         return err;
     else
    i_output_opponent = theSane.map_boardposition_output( OppMv
   ._row , OppMv._col);

         i_output_player = theSane.map_boardposition_output(
   r_saved , c_saved);
   }

   // Now get the response...
```

```
38      _SaneMv.Set( MT_None, PC_None, 0, 0);
         int SaneAction=0, r=0, c=0;
40
        // map the databoard to inputs
42      theSane.map_databoard_inputs(pBd, inputs);

44      ColorOfPlayer = GetPlayerColor();
        if( GetPlayerColor() == PC_Black )
46          SaneAction = theSane.SaneHandleMyMove( BLACK, WHITE,
        inputs, &r, &c);
        else
48          SaneAction = theSane.SaneHandleMyMove( WHITE, BLACK,
        inputs, &r, &c);

50      //save these values
        r_saved=r;
52      c_saved=c;

54  //update the memory with the last movements i_output_player and
         i_output_movement
        theSane.updateMemory(i_output_player,i_output_opponent,1);
56
        switch( SaneAction )
58      {
        case RESIGN:
60          _SaneMv.Set( MT_Resign, GetPlayerColor(), 0, 0);//
        resign
            break;
62      case PASS:
            _SaneMv.Set( MT_Pass, GetPlayerColor(), 0, 0);// pass
64          break;
        case BOTHPASS:
66          _SaneMv.Set( MT_Pass, GetPlayerColor(), 0, 0);// pass
            break;
68      default:    // move
            _SaneMv.Set( MT_Move, GetPlayerColor(), r, c); // move
70          break;
        }
72      // post results back...
        Post_MoveRsp( _SaneMv );
74      return ERR_NONE; // no error
    }
```

- Network activation function used in SANE method.

```
1
    void activate_net(network *net)
```

```
{
    int i,j,hd;
    hd=net->nhidden; // number of neurons in the blueprint
    double sum,max;
    neuron *h;
    max = -999999.0;
    /*reset output layer*/
    for(i=0;i<NUM_OUTPUTS;++i)
      net->sum[i] = 0.0;
    for (i=0;i<hd;++i)
    {  h = net->hidden[i];
       sum = 0.0;
       for(j=0;j<h->numin;++j)
           sum += h->in_weight[j] * net->input[h->in_conn[j]];
       h->sigout =   1/(1+exp(-sum));
       for(j=0;j<h->numout;++j)
            net->sum[h->out_conn[j]] += h->out_weight[j]*h->
    sigout;
    }
    for(i=0;i<NUM_OUTPUTS;++i)
    { net->sigout[i] = 1/(1+exp(-net->sum[i]));
      if (net->sum[i] >= max) {
        net->winner = i;
        max = net->sum[i];
      }
    }
}
```

- Application of genetic operators to neurons used in SANE

```
//crossover of neuron's population
for (j=0;j<NUM_BREED;++j)
   {   if ( randint(0,100) < CROSS_RATE)    // crossover rate
             one_pt_crossover(popu[j%ELITE],popu[find_mate(j%
    ELITE)],popu[POP_SIZE-(1+j*2)],popu[POP_SIZE-(2+j*2)]);
    }

//mutation of neuron's population
for (j=NUM_BREED;j<POP_SIZE;++j)
 { // all worst hidden neurons after NUM_BREED have to be
   MUTTED
     if (randint(0,100) < MUT_RATE)
     { for(i=0;i<GENE_SIZE;++i)
          { if (randint(0,100) < MUT_RATE) //mutation rate
     { if (i%2)
           popu[j]->gene[i] = -popu[j]->gene[i];
       else
```

```
    popu[j]->gene[i] = randint(0,(NUM_INPUTS+NUM_OUTPUTS));

     }
          }
    }
}
```

- Application of genetic operators to blueprints networks used in SANE

```
// Crossover of blueprint's population
  for (j=0;j<TOP_NETS_BREED;++j)
    { if (randint(0,100) < CROSS_RATE)  //crossover rate
              new_network_one_pt_crossover(j,popu);
    }

//mutation of blueprint's population
for(j=TOP_NETS_BREED;j<TOP_NETS;++j)
 { if (randint(0,100) < MUT_RATE)
       {for(i=0;i<best_nets[j]->nbhidden;++i)
  if (randint(0,100) < MUT_RATE)
     best_nets[j]->hidden[i] = popu[randint(0,POP_SIZE)];
       }
 }
```

- Introducing more diversity using RIR rate (inmigration_rate). gm_not_lost_host is the number of games not lost by the player. TOP_NETs is the total number games played.

```
  inmigration_rate=exp((gm_not_lost_host/(TOP_NETS-1)));
  for (i=(POP_SIZE/2);i<POP_SIZE;++i)
  { // replace the worst perfomed neurons with new population
       if ((randint(0,100)*1.0) < inmigration_rate  )
        {   // create new population of neurons
             popu[i]->decoded = 0;
    for(j=0;j<GENE_SIZE;++j)
    { if (j%2)
         popu[i]->gene[j]=normal_dist(0.0, 0.10);
      else
                  popu[i]->gene[j]=randint(0,(NUM_INPUTS+
 NUM_OUTPUTS-1));
             }
  }
   }
```

- Mechanism to reinforce some movements using the memory of moves of blueprints saved during the competitions.

```
 for(j=0;j<POP_SIZE;++j)
{
    for (i=0;i<best_nets[0]->nbhidden;++i)
    { if (best_nets[0]->hidden[i] == popu[j])
        {for (k=0;k<best_nets[0]->n1mov;k++)
           reinforce_learning(popu[j], best_nets[0]->
   o1_moved_player[k], k);
        }
     }
}
void reinforce_learning(neuron *n, int mov, int pos )
{
  int i,out;
  float beta=0.05;

  for(i=0;i<GENE_SIZE;i+=2)
  {
    if (n->gene[i] > NUM_INPUTS - 1)
    {  out = (int) n->gene[i] - NUM_INPUTS;
       if (out == mov)
          n->gene[i+1] += (1/(1+exp(pos*1.0)))*beta;
     }
   }
}
```

- Calculation of fitness function for Host player for coevolution and not coevolution learning process. If can be observed how is calculated the fitness function using CFS or CFSA, and if it is used the SCORE or number of wins in the evaluation function. The calculation is similar for Parasite player.

```
 //Black is Host player
if (COEVOLUTION)
{
//calculation of parameters used for fitness funtion
 xi=((_generations+1)%10)/10;
 al=1.0 - xi;
 para_inv=0.0;
 wtmp=0;

 for (i=0;i<NUM_TRIALS;i++)
 { if (tab_lost_host[i].win > wtmp)
     { wtmp = tab_lost_host[i].win;
       wi= i;
     }
```

```
  }

for (i=0;i<NUM_TRIALS;i++)
  { if (i<TOP_NETS_BREED)
       {sbp[i]=best_nets[i]->fitness;
       }
        best_nets[i]->fitness=0;
  }

for (i=0;i<(NUM_TRIALS*NUM_TRIALS);i++)
 {if (tab_score_sharing[i].score_parasite<tab_score_sharing[i].
    score_host)  //check if host wins parasite
    { para_inv = (1.0/(tab_lost_parasite[tab_score_sharing[i].
    parasite].lost+1));
      //If score in the fitness functions is considered, if it
    is not is considered number of wins
      if (FITNESS_SCORE)
        ifitness=xi*para_inv+al*((tab_score_sharing[i].
    score_host*1.0)/(NUM_TRIALS*81.0));
      else
        ifitness=xi*para_inv+al*(1.0/NUM_TRIALS);
      best_nets[tab_score_sharing[i].host]->ifitness[
    tab_score_sharing[i].parasite]=ifitness;
      best_nets[tab_score_sharing[i].host]->fitness += ifitness
    ;
     }
   else
   {  //This is to calculate fitness using CFSA. It is
    considering when parasite won games
     if (CFSA)
     {
       if (tab_score_sharing[i].score_parasite >
    tab_score_sharing[i].score_host)
   {
     para_inv = (-1.0/(tab_lost_parasite[tab_score_sharing[i].
    parasite].win+1));
         if (FITNESS_SCORE)
            ifitness=xi*para_inv + al*((tab_score_sharing[i].
    score_host*1.0)/(NUM_TRIALS*81.0));
         else
             ifitness=xi*para_inv;
    best_nets[tab_score_sharing[i].host]->ifitness[
    tab_score_sharing[i].parasite]=ifitness;
    best_nets[tab_score_sharing[i].host]->fitness +=ifitness;
         }
  }
    }
 }

```

```
 // Consider fitness from previous generation
54 if ((USE_PREV_FITNESS) && (_generations>0))
 {
56  for (i=0;i<TOP_NETS_BREED;i++)
    {
58   best_nets[i]->fitness=(best_nets[i]->fitness + sbp[i])/2.0;
    }
60 }
}
62 else   // When is not coevolution is considered the score
    obtained (by Host). If Parasite won is not considered the
    score.
{
64  for (i=0;i<NUM_TRIALS;i++)
  //1 in case parasite won, and 0 in case Host won and was
    drawn
66  if (tab_score_sharing_nc[i].who_win != 1)
    best_nets[tab_score_sharing_nc[i].host]->fitness=
    tab_score_sharing_nc[i].score_host;
68  else
    best_nets[tab_score_sharing_nc[i].host]->fitness=0;
70 }
```

- This section is calculating the population diversity using Edit Distance and the method proposed by the author (onsidering the sign + or - or the weigth in the neuron structures.

```
float w1,w2,dis_com,dc,dne, div, diversity=0,edtdis,g2;
2 int co,dct;
edtdis=0;
4 for (k=0;k<POP_SIZE-1;k++)
{ dct=0;
6  div=0;
  for(j=k+1;j<POP_SIZE;j++)
8  { if (popu[k] != popu[j])
    { co=0;
10    dis_com=0;
      for (i=0; i<GENE_SIZE;i+=2)
12    {  for (n=0; n<GENE_SIZE;n+=2)
          {  if (EDITDIST)   // using edit distance
14           { if (popu[k]->gene[i] == popu[j]->gene[n])
               { if (popu[k]->gene[i+1] < 0)
16               w1= popu[k]->gene[i+1] * (-1 );
                 else
18               w1= popu[k]->gene[i+1];
                 if (popu[j]->gene[n+1] < 0)
20               w2= popu[j]->gene[n+1] * (-1 );
```

```
                    else
                        w2= popu[j]->gene[n+1];
                    if (w1 > w2)
                        dc=(w1 - w2)/w1;
                    else
                        dc = (w2 - w1)/w2;
                    dis_com +=dc;
                    co++;
                    break;
                }
             }
            else
              {//considering sign + or - of neuron's weigth
                if (popu[k]->gene[i] == popu[j]->gene[n])
                  { w1= popu[k]->gene[i+1];
                    w2= popu[j]->gene[n+1];
                    if (w1 > w2)
                        dc = (w1 - w2);
                    else
                        dc = (w2 - w1);
                    dis_com +=dc;
                    co++;
                    break;
                    }
                 }
            }
         }
        g2=(GENE_SIZE/2)-co;
        if (co > 0)
            dne = ( 4*(g2/GENE_SIZE)  + (dis_com/co))/3;
        else
            dne = 1;
        div += dne;
        dct++;
        }
    }
  //edit disctance for all neurons compared against other
   neurons
      edtdis += div/dct;
}
 //diversity calculated for Black population
 diversity = edtdis/POP_SIZE;
```

Appendix C

This appendix describe the Technical Market Indicators (TMI) discussed in this thesis.

.2 Bollinger Bands

The Bollinger Bands indicator was created by Bollinger [2001] at the 80s. It addresses the issue of dynamic volatility by introducing adaptive bands that widen during period of high volatility and contract during periods of low volatility.

The main purpose of Bollinger Bands is to place the current price of a security into perspective, providing a relative definition of high and low volatility (therefore supply/demand) and trend.

There are three time series that compose the Bollinger Bands indicator, which consists of an upper ($UpBand_p$), middle ($MidBand_p$), and lower times series ($LowBand_p$).

$MidBand_p$ is usually a simple moving average, used as a measure of intermediate term trend. $UpBand_p$ and $LowBand_p$ are standard deviation of $MidBand_p$. Three times series are defined:

$$MidBand_p(t) = \frac{\Sigma_{j=t-p+1}^{t}}{price(j)}p \tag{1}$$

$$LowBand_p(t) = MidBand_p(t) - [D\sqrt{\frac{\Sigma_{j=t-p+1}^{t}(price(j) - MidBand_p(p))^2}{p}}] \tag{2}$$

$$UpBand_p(t) = MidBand_p(t) + [D\sqrt{\frac{\Sigma_{j=t-p+1}^{t}(price(j) - MidBand_p(p))^2}{p}}] \tag{3}$$

Where $price(j)$ represents the price of the security (commonly the closed price is used), p is the number of periods used for the simple moving average

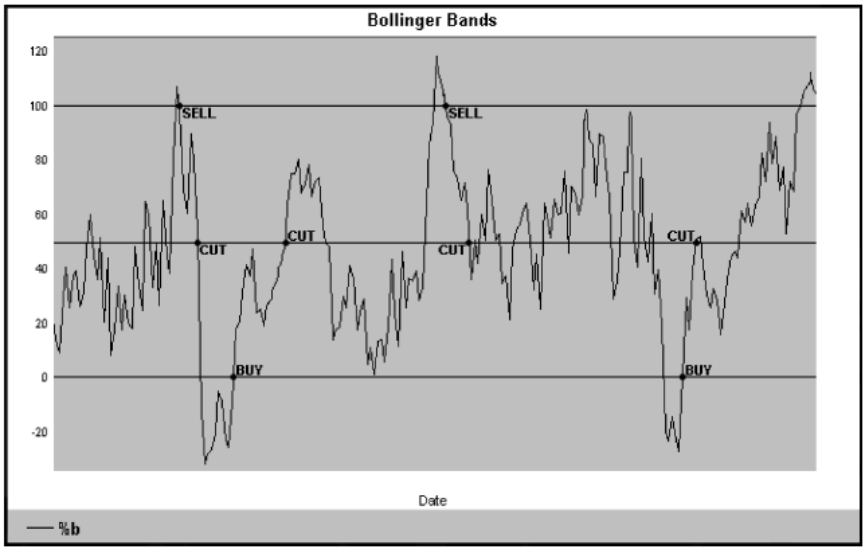


Figure 10: Bollinger Bands Chart

calculations, and the constant D is and adjustment value by which the standard deviation of the simple moving average is shifted above and below $MidBand_p$. The default variable values used by Bollinger are 20 for the period p and 2.0 for the adjustment factor D.

A time series referred to as percentage bands (%b) is calculated to quantify the relative price of the security over time, defined as:

$$\%b(t) = 100\left(\frac{price(t) - LowBand_p(t)}{UpBand_p(t) - LowBand_p(t)}\right) \tag{4}$$

%b value close to 100 indicate prices that are at relatively high levels, possibly unsustainable, %b value close to zero indicates low prices level, possible unsustainable. A multiple price penetration of $UpBand_p$ indicates the security is over-bought while multiple price penetration of $LowBand_p$ indicates the security is over-sold.

These are the trading rules for this indicator:

- if $\%b(t-1) < 0$ and $\%b(t) > 0$ then BUY
- if $\%b(t-1) > 100$ and $\%b(t) < 0$ then SELL
- if $\%b(t-1) < 50$ and $\%b(t) > 50$ then CUT
- if $\%b(t-1) > 50$ and $\%b(t) < 50$ then CUT

The interval between the UpBand_p and $\mathsf{LowBand}_p$ time series indicates a volatility in the security. As more separated these two bands are, more volatile in the security. The following times series quantify the volatility of the security and can be used to understand the supply and demand of that particular security:

$$\mathsf{bandwidth}(t) = \frac{\mathsf{UpBand}_p(t) - \mathsf{LowBand}_p(t)}{\mathsf{MidBand}_p(t)} \tag{5}$$

.3 Moving Average Converge/Divergence (MACD)

MACD was developed by Appel [2005] as a stock maker timing device, utilizing market momentum and trend.

A short and long period exponential moving average (EMA) is calculated on the security price. The difference as these two values are referred as price momentum. A second time series is calculated applying another EMA on the price momentum using a smaller period that the ones originally used on the security price. These two time series formulate the MACD indicator.

$$\mathsf{PriceMomentum}(t) = \mathsf{EMA}_a(\mathsf{price}) - \mathsf{EMA}_b(\mathsf{price}) \tag{6}$$

$$\mathsf{MomentumTrigger}(t) = \mathsf{EMA}_c(\mathsf{PriceMomentum}) \tag{7}$$

$$\mathsf{EMA}_p(t) = \mathsf{price}(t)\left(\frac{2}{p+1} + \frac{\Sigma_{j=t-p+1}^{t}\mathsf{price}(j))}{p}\left(100 - \frac{2}{p+1}\right)\right) \tag{8}$$

where a, b, c indicate the periods to be used for the exponential moving average calculations, where $b > a > c$. Appel recommend using a 12 days period for the fast exponential moving average and a 26 days period for the slow exponential moving average to calculate the price momentum indicator. It was used 9 days period for the exponential moving average in the $\mathsf{MomentumTrigger}$ time series. Short period EMA is referred as fast EMA, and long period EMA as slow EMA.

The $\mathsf{PriceMomentum}$ time series oscillates around zero axis, highlighting positive and negative market momentum. Positive momentum indicates that the average price for the fast EMA exceed the slow EMA, indicating a rise in the underlying price or the security is over-bought. Negative momentum indicates that fast EMA has fallen below that slow EMA, which indicate that the security is over-sold leading a fall in the security price. When the price momentum shifts from a positive to a negative value or viceversa, a trend reversal is indicating to generate a CUT action. These are the rules describe above for this indicator:

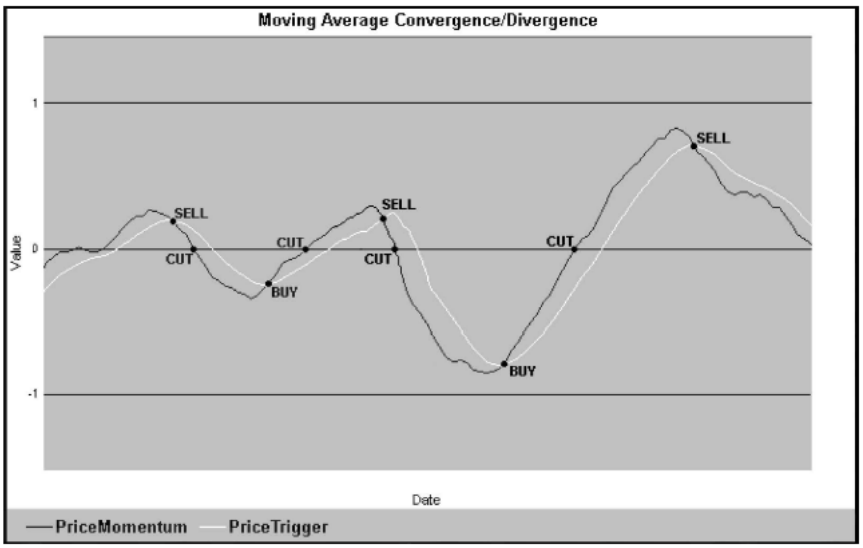


Figure 11: Moving Average Convergence/Divergence chart

- If $\mathsf{PriceMomentum}(t-1) < 0$ and $\mathsf{PriceMomentum}(t) > 0$ then CUT
- If $\mathsf{PriceMomentum}(t-1) > 0$ and $\mathsf{PriceMomentum}(t) < 0$ then CUT
- if $\mathsf{PriceMomentum}(t{-}1) < \mathsf{MomentumTrigger}(t{-}1)$ and $\mathsf{PriceMomentum}(t) > \mathsf{MomentumTrigger}(t)$ then BUY
- if $\mathsf{PriceMomentum}(t{-}1) > \mathsf{MomentumTrigger}(t{-}1)$ and $\mathsf{PriceMomentum}(t) < \mathsf{MomentumTrigger}(t)$ then SELL

The short-term trends may be up-trends or down-trends during periods of positive or negative price momentum.

.4 Relative Strength Index (RSI)

RSI indicator was developed by Wilder [1978]. RSI returns a value that continuously oscillates, tracking price strength and displaying the velocity and momentum of a security price, comparing the magnitude of a security's recent gains to the magnitude of its recent losses.

RSI is calculated as:

$$\mathsf{RSI}_p(t) = 100(1 - \frac{1}{1 + \mathsf{RS}_p(t)}) \tag{9}$$

$$RS_p(t) = \frac{TotalGain_p(t)}{TotalLoss_p(t)} \tag{10}$$

$$TotalGain_p(t) = \Sigma_{j=t-p+1}^{t}(price(j) - price(j-1) > 0) \tag{11}$$

$$TotalLoss_p(t) = |\Sigma_{j=t-p+1}^{t}(price(j) - price(j-1) < 0)| \tag{12}$$

If $average_{loss} = 0$, $RSI = 100$.

Where p is the number of periods used for calculating $average_{loss}$ and $average_{gain}$. Larger values for p result in smoother RSI curves, while small values for p results in large volatility in the curve.

The fixed levels need to be defined to aid the interpretation of RSI, namely an upper level (L_{upper}) and lower level (L_{lower}). It is recommended the upper level to be set at 70 and lower level to be set at 30. When RSI is above the upper level and then falls below the upper level, this is a warning of a potential trend reversal, meaning that there is an over-bought of the security. When RSI is below the lower and then rises above the lower level there is over-sold of the security.

A mid level (L_{mid}) for RSI is at 50. Values above 50 indicate that average gains are more than average losses, which can be used as a confirmation of a bullish trend. Bearish trends can be confirmed when RSI falls below 50, since the average losses are more than the average gains.

These are the rules applied to this indicator:

- If $RSI(t-1) < L_{lower}$ and $RSI(t) > L_{lower}$ then BUY
- If $RSI(t-1) > L_{upper}$ and $RSI(t) < L_{upper}$ then SELL
- If $RSI(t-1) < L_{mid}$ and $RSI(t) > L_{mid}$ then CUT
- If $RSI(t-1) > L_{mid}$ and $RSI(t) < L_{mid}$ then CUT

A divergence in the price and the indicator happens when a price has a new high and new low and the RSI fails over exceed its previous high and low respectively. A negative divergence during a period where the security is over-bought would entail a SELL action while a BUY action would be returned when a positive divergence takes place during a period where the security is over-sold.

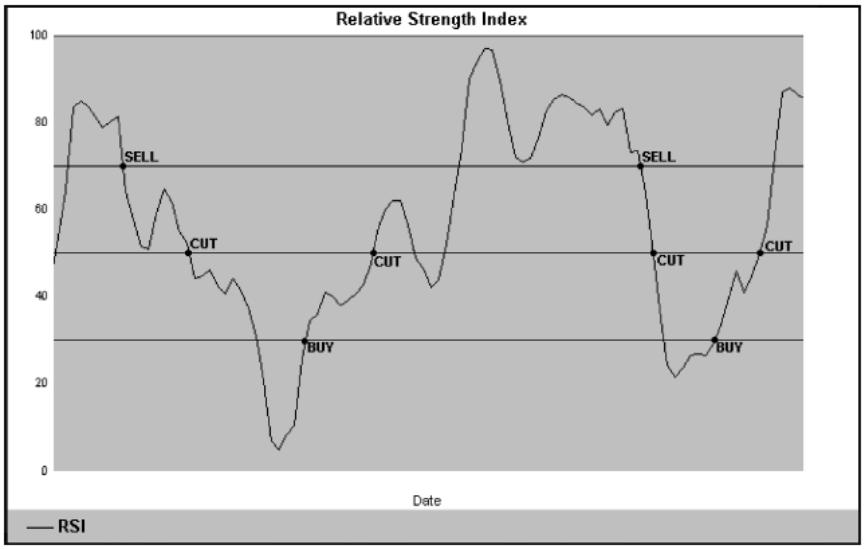


Figure 12: Relative Strength Index chart

References

Adamu, K. & Phelps, S. (2010). Co-evolution of technical trading rules for high frequency trading. *in Proceedings of the World Congress on Engineering*, **I**. 50

Allis, L.V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. thesis, University of Maastrich. 8, 15, 23, 24, 26, 48

Angeline, P.J. & Pollack, J.B. (1993). Competitive environments evolve better solutions for complex tasks. *Genetic Algorithms: Proceedings of the Fifth International Conference*. 55, 60, 86

Appel, G. (2005). *Technical analysis: Power tool for active investors*. Finance Times / Prentice Hall. 234

Axelrod, R. (1987). The evolution of strategies in the iterated prisoners dilemma. *Genetic Algorithms and Simulated Annealing*, 32–41. 56, 58

Axelrod, R. & Dion, D. (1988). The further evolution of cooperation. *Science 242*, 1385–1390. 56, 58

Azorin, J. & Sanchez-Crespo, J.L. (1986). *Mtodos y aplicaciones del muestreo*. Alianza Editorial. 74

Back, T., Hammel, U. & Schwefel, H. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, **1**, 3–17. 53, 54, 55

Bell, R. (1980). *Board and Table Games from many Civilizations*. Dover Publications Inc. 13

Berlekamp, E. (1991). Introductory overview of mathematical go endgames. *Proceedings of Symposia in Applied Mathematics*, 43. 32

Bollinger, J.A. (2001). *Bollinger on Bollinger Bands*. MacGraw-Hill. 232

Bouzy, B. & Cazenave, T. (2001). Computer go: an ai oriented survey. 8, 25, 32, 33, 34, 38, 47, 49

Brabazon, A. & O'neil, M. (2006). *Biologically Inspired Algorithms for Financial Modeling*. Springer. 42, 50, 54

Brugmann, B. (1993). Monte carlo go. *Max-Planck-institute of physic*. 8, 38, 39

Burke, E., Gustafson, S. & Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, **8**, 47–62. 75, 76

Burrow, P. & Lucas, S.M. (2009). Evolution versus temporal difference learning for learning to play ms. pac-man. *IEEE Symposium on Computational Intelligence and Games*. 9

Cartlidge, J. & Bullock, S. (2004). Combating coevolutionary disengagement by reducing parasite virulence. *Evolutionary Computation*, **12**, 193–204. 75

Chaslo, G. (2010). *Monte-Carlo Tree Search*. Ph.D. thesis, University of Maastrich. 9, 40

Chaslot, G., Bakkes, S., Szita, I. & Spronck, P. (2008). Monte-carlo tree search: a new framework for game ia. *University of Maastricht*. 38, 39, 40, 48

Chellapilla, K. & Fogel, D. (1999). Evolving neural networks to play checkers without expert knowledge. *IEEE Transactions on Neural Networks*, **10**, 1382–1391. 9, 49, 60, 61

Chen, H. & Yao, X. (2010). Multiobjective neural network ensembles based on regularized negative correlation learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**, 1738–1751. 53

Chong, S.Y. (2007). *Generalization and Diversity in Co-evolutionary Learning*. Ph.D. thesis, University of Birmingham. 53, 54, 55, 58, 69, 70, 87

Chong, S.Y., Tan, M.K. & White, J.D. (2005). Observing the evolution of neural networks learning to play the game of othello. *IEEE Trans.Evol. Computation*, **9**, 240–251. 49

Chong, S.Y., Tino, P. & Yao, X. (2008). Measuring generalization performance in co-evolutionary learning. *IEEE Transactions on Evolutionary Computation*, **12**, 479–505. 69, 70, 73, 134

Chong, S.Y., Tino, P. & Yao, X. (2009). Relationship between generalization and diversity in co-evolutionary learning. *IEEE Transactions on Computation Intelligence and AI in games*, **1**, 214–232. 74, 75, 76, 82, 83, 134

Chong, S.Y., Tino, P., Ku, D.C. & Yao, X. (2012). Improving generalization performance in co-evolutionary learning. *IEEE Transactions on Evolutionary Computation*, **16**, 70–85. 73, 74

Clarke, R. (2010). *Cyber War*. HarperCollins. 211

Coello, C. (1998). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*. 53

Committee, A.R. (1991). Official aga rules of go, available at http://www.usgo.org/files/pdf/completerules.pdf. 15

Conway, J.H. (1976). *On Numbers And Games*. Academic Press. 32

Copeland, B.J. (1999). Alanturing.net, available at http://www.alanturing.net/index.htm. 41

Copeland, B.J. (2004). *The Essential Turing: The ideas that gave birth to the computer age*. Clarendon Press. 7

Corduck, M. (1979). *Machines who Think*. W. M. Freeman and Co. San Francisco. 28

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *LIFL, SequeL, INRIA Futurs, Universit Charles de Gaulle*. 40

Darwen, P. & Yao, X. (1995). Evolving robust strategies for iterated prisoners dilemma. *Progress in Evolutionary computation*, 276–292. 69

Darwen, P.J. (2001). Why co-evolution beats temporal difference learning at backgammon for a linear architecture, but not a non-linear architecture. 9, 37, 49, 58, 59

Darwen, P.J. & Yao, X. (1997). Speciation as automatic categorical modularization. *IEEE Transactions on Evolutionary Computation*, **1**, 101–108. 75

Dawkins, R. & Krebs, J.R. (1979). Arms races between and within species. *Proceedings of the Royal Society of London*. 52

de Jong, E.D. & Pollack, J.B. (2004). Ideal evaluation from coevolution. *Evolutionary Computation*, **12**, 159–192. 75, 82, 83

DEN BERGH, F.V. (2002). *An analysis of particle swarm optimiser*. Ph.D. thesis, University of Pretoria. 96

EASLEY, D. & KLEINBERG, J. (2010). Evolutionary game theory. *From the Book, Networks, Crowds, and Markets: Reasoning about a Highly Connected World*, 209–227. 12

ECKHARDT, R. (1987). Stam ulam, john von neumann, and the monte carlo method. *Los Alamos Science*, **Special Issue(15)**, 131–137. 38

EKART, A. & NEMETH, S. (2002). Maintaining the diversity of genetic programs. *Proceeding EuroGP '02 Proceedings of the 5th European Conference on Genetic Programming*, **2278**, 162–171. 75

ENZENBERGERL, M. (1996). The integration of a priori knowledge into a go playing neural network. 8, 34, 35

EPSTEIN, S.L. (1994). Toward an ideal trainer. *Machine Learning*, **15(3)**, 251–277. 70

FICICI, S.G. (2004). *Solution Concepts in Coevolutionary Algorithms*. Ph.D. thesis, Brandeis University. 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 69, 78, 79, 80, 86

FICICI, S.G. (2005). Monotonic solution concepts in coevolution. *GECCO '05 Proceedings of the 2005 conference on Genetic and evolutionary computation*, 499–506. 62, 79

FICICI, S.G. & POLLACK, J.B. (1998). Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. *Proceedings of the 6th International Conference on Artificial Life (ALIFE-98)*. 65

FICICI, S.G. & POLLACK, J.B. (2001). Pareto optimality in coevolutionary learning. *Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag*, **2159**, 316–325. 83

FOGEL, D. (2000). What is evolutionary computation? *IEEE Spectrum*. 53, 54

FRANKEN, C.J. (2004). Pso-based coevolutionary game learning. 9, 93, 94

GNEDENKO, B.V. & GNEDENKO, G.V. (1998). *Theory of Probability*. Taylor and Francis. 71

GRUAU, F., WHITLEY, D. & PYEATT, L. (1996). A comparison between cellular encoding and direct encoding for genetic neuroal networks. 45

Handa, H., Chapman, L. & Yao, X. (2006). Robust route optimization for gritting/salting trucks: a cercia experience. *Computational Intelligence Magazine, IEEE*, **1**, 6–9. 54

Heinz, E.A. (2003). Follow-up on self-play, deep search, and diminishing returns. 27

Hillis, D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, **42**, 228–234. 55

Hutchinson, C.R. (1995). *American Go Association 1995, Historical Book*. The American Go Association. 14

Jansen, D.H. (1980). When it is coevolution? *Evolution*, **34**, 611–612. 51

Juille, H. & Pollack, J.B. (2000). Coevolutionary learning and the design of complex systems. *Advances in Complex Systems*, **2**, 371–393. 61

Junghanns, A. & Schaeffer, J. (1997). Search versus knowledge in game-playing programs revisited. 27

Kauffman, S.A. (1993). *The Origins of Order, Self-Organization and Selection in Evolution*. Oxford University Press. 55

Kennedy, J. (1998). The behavior of particles. *In Proceedings of the 7th International Conference on Evolutionary Programming*, 581–589. 96

Kennedy, J. (1999). Small worlds and mega minds: Effects of neighborhood topology on particle swarm performance. *In Proceedings of the IEEE Congress on Evolutionary Computation*, **3**, 1931–1938. 95

Kennedy, J. & Eberhart, R.C. (1995). Particle swarm optimization. *In Proceedings of the IEEE International Conference on Neural Networks*, **IV**, 1942–1948. 93

Kennedy, J. & Eberhart, R.C. (2001). *Swarm Intelligence*. Morgan Kaufmann. 93

Kim, J. & Soo-hyun, J. (1994). *Learn to play GO. A Masters Guide to the Ultimate Game (Vol. I)*. Good Move Press. 22

Kim, J. & Soo-hyun, J. (1995). *Learn to play GO. The Way of the Moving Horse (Vol. II)*. Good Move Press. 22

Kim, J. & Soo-hyun, J. (1996). *Learn to play GO.The Drago Style (Vol. III)*. Good Move Press. 22

Knuth, D.E. & Moore, R.W. (1975). An analysis of alpha-beta prunning. *Artificial Intelligence*, 293–326. 24, 28, 29, 30

Kotnik, C. & Kalita, J. (2003). The significance of temporal-difference learning in self-play training td-rummy versus evo-rummy. 9, 37, 49

Kupfer, A. & Eckestein, S. (2006). Coevolution of database schemas and associated ontologies in biological context. *Technical University of Braunschweig, Institute of Information Systems*. 50

Lefkovitz, D. (1960). A strategic pattern recognition program for the game go. *WADD technical note*, **60**. 47

Lewis, A. (2009). Locost: A spatial social network algorithm for multi-objective optimization. *Evolutionary Computation*. 95

Lipson, H. & Pollack, J.B. (2000). Automatic design and manufacture of artificial lifeforms. *Nature*, 974–978. 50, 60

Liskowski, P. (2012). Co-evolution versus evolution with random sampling for acquiring othello position evaluation. 69

Lubberts, A. & Miikkulainen, R. (2001). Co-evolving a go-player neuronal network. 49, 53, 58, 59, 110

Lucas, S.M. & Runarsson, T.P. (2006). Temporal difference learning versus co-evolution for acquiring othello position evaluation. 9, 37, 49, 53, 59

Luke, S. & Wiegand, P. (2003). When coevolutionary algorithms exhibit evolutionary dynamics. *In Workshop Proceedings of the 2003 Genetic and Evolutionary Computation Conference*. 64, 65, 138

Marois, G. (2008). Replacement migration : methodological and demographic issues in quebec. *Erudit*. 118

McCulloch, W. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115–133. 41

Messerschmidt, L. & Engelbreacht, A.P. (2002). Learning to play games using pso-based competitive learning approach. *In Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*. 93

Monroy, G.A., Stanley, K.O. & Miikkulainen, R. (2006). Coevolution of neural networks using a layered pareto archive. *GECCO 2006*. 82

MORIARTY, D.E. (1997). *Symbiotic Evolution of Neural Networks in Sequencial Decision Tasks*. Ph.D. thesis, Department of Computer Sciences of University of Texas. 45, 46, 49, 56, 109

MORIARTY, D.E. & MIIKKULAINEN, R. (1996a). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*. 3, 45

MORIARTY, D.E. & MIIKKULAINEN, R. (1996b). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, **22**, 11–33. 49

MORIARTY, D.E. & MIIKKULAINEN, R. (1998a). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*. 49

MORIARTY, D.E. & MIIKKULAINEN, R. (1998b). Forming neural networks through eficient and adaptive coevolution. *Evolutionary Computation*. 8, 45

MULLER, M. (1995). *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. Ph.D. thesis, Swiss Federal Institute of Technology Zurich. 8, 32, 116

MULLER, M. (1999). Descomposition search: A combinatorial games approach to game tree search, with applications to solving go endgames. *Proceedings IJCAI*, 578–583. 9, 32

MULLER, M. (2000). Computer go. 47

NASH, J. (1951). Non-cooperative games. *The Annals of Mathematics*, **54, issue 2**, 286–295. 11

NAU, D.S. (1980). Pathology of game trees: A summary of results. *AAAI-80 Proceedings*, 102–104. 28

NOBLE, J. & WATSON, R.A. (2001). Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. *in Proceeding Genetic Evolutionary Computation Conference 2001*, 493–500. 82

PAPACOSTANTIS, E. (2009). Competitive co-evolution of trend reversal indicators using particle swarm optimization. 50, 93, 96, 97, 99

PAPACOSTANTIS, E., ENGELBRECHT, P. & FRANKEN, N. (2005). Coevolving probabilistic game playing agents using particle swarm optimization algorithms. *In Proceedings of the IEEE Evolutionary Computation in Games Symposium*, 195–202. 9

Pazos, J. (1980). *Programas Abiertos como Principio del Metodo de Teoria en Inteligencia Artificial*. Ph.D. thesis, Universidad Policteсnica de Madrid. 111

Pazos, J. (1987). *Inteligencia Artificial, Programacion Heuristica*. Paraninfo. 8, 25, 28

Perez-Bergquist, A.S. (2001). Applying esp and region specialists to neuroevolution for go. 44, 56, 69, 110, 111, 113, 116, 123, 134

Pollack, J.B. & Blair, A.D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*. 58

Potter, M.A. & de Jong, K. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, **8 (1)**, 1–29. 56, 58

Poundstone, W. (1992). *Prisioners dilemma: John Von Neumann, Game Theory and the Puzzle of the Bomb*. Anchor Books. 10

Rawal, A., Rajagopalan, P. & Miikkulainen, R. (2010). Constructing competitive and cooperative agent behavior using co-evolution. *In IEEE Conference on Computational Intelligence and Games*. 50, 87, 91, 92, 103

Remus, H. (1962). Simulation of a learning machine for playing go. *Computer Games*, **II**, 428–432. 47

Richards, N., Moriarty, D.E. & Miikkulainenn, R. (1998). Evolving neural networks to play go. *Applied Intelligence*, **8**, 85–96. 111, 113, 123

Rider, J.L. (1971). *Heuristic Analysis of Large Trees as Generated in the Game of Go*. Ph.D. thesis, Stanford University. 47

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, 386–408. 41

Rosin, C.D. (1997). *Coevolutionary Search Among Adversaries*. Ph.D. thesis, University of San Diego. 66, 85, 86

Rosin, C.D. & Belew, R.K. (1997). New methods for competitive coevolution. *Evolutionary Computation*, **5**, 1–29. 2, 56, 58, 63, 64, 66, 67, 69, 70, 75, 81, 82, 83, 84, 110, 125, 127, 131, 133, 134, 167

Runarsson, T.P. & Lucas, S.M. (2005). Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *IEEE Transactions on Evolutionary Computation*, **9**, 628–640. 9, 37, 49, 58

Salcedo-Sanz, S., Cruz-Roldan, F., Heneghan, C. & Yao, X. (2007). Evolutionary design of digital filters with application to subband coding and data transmission. *IEEE Transactions on Signal Processing*, **55**, 1193–1203. 50, 53

Sallam, H., Regazzoni, C.S., Talkhan, I. & Atiya, A. (2008). Measuring the genotype diversity of evolvable neural networks. *INFOS2008*. 76, 77

Sandholm, W.H. (2007). Evolutionary game theory. 12, 13

Schleicher, D. & Stoll, M. (2005). An introduction to conway's games and numbres. 32

Schraudolph, N.N., Dayan, P. & Sejnowski, T.J. (2000). Learning to evaluate go positions via temporal difference methods. 34

Sims, K. (1994). Evolving 3d morphology and behavior by competition. *Artificial Life*, **1**, 353–372. 59, 80

Sivanandam, S.N. & Deepa, S.N. (2008). *Introduction to Genetic Algorithms*. Springer. 93

Smith, J.M. (1972). Game theory and the evolution of fighting. *in On Evolution*, 8–28. 12

Smith, J.M. (1982). *Evolution and Theory of Games*. Cambridge University Press. 12

Smith, J.M. (1989). *Evolutionary Genetics*. Oxford University Press. 109

Smith, J.M. & Price, G.R. (1973). The logic of animal conflict. *Nature*, 15–18. 12

Stanley, K.O. & Miikkulainen, R. (2002a). Efficient evolution of neural network topologies. *Proceedings of the 2002 Congress on Evolutionary Computation*. 43

Stanley, K.O. & Miikkulainen, R. (2002b). Evolving neural network through aumenting topologies. 44, 45, 77

Suganthan, P.N. (1999). Particle swarm optimiser with neighborhood operator. *In Proceedings of the IEEE Congress on Evolutionary Computation*, 1958–1961. 95

Sutton, R.S. (1988). Learning to predict by the methods of temporal difference. *Machine Learning*, **3**, 99–44. 34

Tesauro, G. (1993). Td-gammon: A self-teaching backgammon program, achieves master level play. *AAAI Technical Reports*, **FS-93-02**, 19–23. 34, 61

Turocy, T.L. & Stengel, B.V. (2002). *Game Theory*. Encyclopedia of Information Systems. 9, 11

Uchibe, E. & Asada, M. (2006). Incremental coevolution with competitive and cooperative tasks in a multirobot environment. *Proceedings of the IEEE*, **94**, 1412–1424. 50

Ulam, S. (1991). *Adventures of a Mathematician*. University of California. 38

Valen, L.V. (1973). *A New Evolutionary Law, Evolutionary Theory*. 52, 64

van der Werf, E. (2004). *AI techniques for the game of Go*. Ph.D. thesis, University of Maastrich. 8, 9, 14, 23, 26, 27, 28, 30, 31, 47, 48

Wilder, J.W.J. (1978). *New Concepts in Technical Trading Systems*. Greenboro, N.C Trend Research. 235

Yang, L. & Anl, D. (2005). *Handbook of Chinese Mythology*. ABC-CLIO. 13

Yao, X. (1997). Automatic acquisition of strategies by co-evolutionary learning. *Proceedings of the International Conference on Computational Intelligence*. 58

Yao, X. (1999). *Evolutionary Computation: Theory and Applications*. World Scientific Publication. 53, 55

Yao, X., Liu, Y. & Darwen, P. (1996). How to make best use of evolutionary learning. *Complex Systems: From Local Interactions to Global Phenomena*, 229–242. 69

Yong, C.H. & Miikkulainen, R. (2007). Coevolution of role-based cooperation in multi-agent systems. 56

Zela, W. & Zato, J. (2011). Evolving and co-evolving computer go players using neuro-evolution. *In the conference COPCOM 2011*. 2, 49, 58, 59, 68, 109, 117, 118, 130

Zobrist, A.L. (1970). *Feature extraction and representation for pattern recognition and the game of go*. Ph.D. thesis, University of Wisconsin. 47