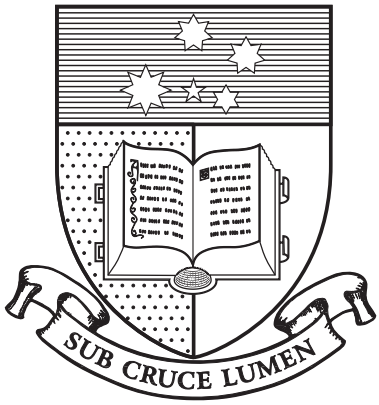
Coevolving a Computer Player for Resource Allocation Games - Using the game of TEMPO as a test space

by

© **Phillipa Avery, BCompSci(Hons)**



A thesis submitted to the
Graduate Centre
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy

Supervisor: Prof. Dr. Zbigniew Michalewicz
Associate Supervisor: Dr. Charles Lakos

School of Computer Science
University of Adelaide

September, 2008

Adelaide Australia

For Ben and Rachael.
Your support has gotten me to where I am today.
Thank you.

Contents

Abstract

decision-making in resource allocation can be a complex and daunting task. Often there exist circumstances where there is no clear optimal path to choose, and instead the decision maker must predict future need and allocate accordingly. The application of resource allocation can be seen in many organizations, from military, to high end commercial and political, and even individuals living their daily life. We define resource allocation as follows: the allocation of owner's assets to further the particular cause of the owner.

We propose two ways that computers can assist with the task of resource allocation. Firstly they can provide decision support mechanisms, with alternate strategies for the allocations that might not have been previously considered. Secondly, they can provide training mechanisms to challenge human decision makers in learning better resource allocation strategies. In this research we focus on the latter, and provide the following general hypothesis: *Coevolutionary algorithms are an effective mechanism for the creation of a computer player for strategic decision-making games*.

To address this hypothesis, we present a system that uses coevolution to learn new strategies for the resource allocation game of TEMPO. The game of TEMPO provides a perfect test bed for this research, as it abstracts real-world military resource allocation, and was developed for training Department of Defence personnel. The environment created allows players to practice their strategic decision-making skills, providing an opportunity to analyse and improve their technique. To be truly effective in this task, the computer player the human plays against must be continuously challenging, so the human can steadily improve. In our research the computer player is represented as a fuzzy logic rule base, which allows us investigation into the strategies being created. This provides insight into the ways the coevolution addresses strategic decision-making.

Importantly, TEMPO also gives us an abstraction of another component of strategic decision-making that is not directly available in other games – that of intelligence (INTEL) and counter intelligence (CI). When resource allocation is occurring in a competitive circumstance, it is often beneficial to gain insight into what your opponent is doing through intelligence. In turn, an opponent may seek to halt or skew the information being gained.

The use of INTEL and CI in TEMPO allows research into the effects this has on the resource allocation process and the coevolved computer player.

The development of a computer player for the game of TEMPO gives us endless possibilities of research. In this research, we have focused on the creation a computer player that can provide a fun and challenging environment for humans learning resource allocation strategies. We investigate the addition of memory to a coevolutionary algorithm for strategy creation. This includes mechanisms to select memory individuals for evaluation of coevolutionary individuals. We describe a successful strategy of selection, based on the way a human's short and long term memory works. We then investigate the use of INTEL and CI in the game of TEMPO, and the way it is used by the coevolved computer players. Through this work, we present a new version of the TEMPO game that more realistically represents INTEL and CI. Finally, we describe a process that uses coevolution to adapt to a human player real-time, to create a tailored game-play experience. This process was tested in a user study, and showed a distinct advantage through the adaptive mechanism. Overall, we have made some important discoveries, and described some limitations that leave future research open. Ultimately, we have shown that our hypothesis is an achievable goal, with an exciting future.

Coevolving a Computer Player for Resource Allocation Games - Using the game of TEMPO as a test space

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

Phillipa Melanie Avery
4th September 2008

Acknowledgements

First and foremost, I would like to sincerely thank my mentor and supervisor Zbigniew Michalewicz. His guidance and endless patience has made this work possible. When I first knocked on his door and asked him to be my supervisor, I could not have imagined the challenging and exciting path that lay ahead of me. It has been a true honour to work with him, and I could not have asked for a better supervisor.

I would also like to thank all the people who have made this research possible. Firstly, I would like to thank Martin Schmidt for helping to form the early stages of the research, and Charles Lakos for his input in the latter stages. Thanks also go to Garrison Greenwood, who came along for the ride and made it better, and I am privileged to call a colleague and friend. Additionally, I would like to thank SAPAC for their continued support and provision of resources.

Thanks go to all the friends and reviewers who have helped shape the work over the years. I would especially like to thank all my colleagues, who have always been there to provide me with a pat on the back, slap to the forehead, and occasional knock on the head. Thanks also go to my family, for the encouragement and support.

Special thanks go to my husband Ben, who has given me the support and strength to keep going. I would also like to thank Rachael, who has kept me sane for all these years. This thesis is dedicated to them.

Finally, I would like to thank Mike Brooks and Dave Munro, who gave a stranger at the door a chance. None of this would have been possible without your trust in me, and I am truly grateful.

List of Tables

List of Figures

List of publications

The work carried out for this thesis has been partially published in a number of conference proceedings and a journal article.
The research for Chapter 6 has been published in the following conference proceedings:

1. P. Avery, Z. Michalewicz, and M. Schmidt. *"A historical population in a coevolutionary system"*. In *IEEE Symposium on Computational Intelligence and Games*, Honolulu, Hawaii, USA, 2007.

2. P. Avery and Z. Michalewicz. *"Static experts and dynamic enemies in coevolutionary games"*. In *IEEE Proceedings for Congress on Evolutionary Computation*, Singapore, 2007.

As well as the following journal article:

1. P. Avery, Z. Michalewicz, and M. Schmidt. *"Short and long term memory in coevolution"*. International Journal of Information Technology and Intelligent Computing, 3(1), 2008.

The research for Chapter 7 was published in the following conference proceedings, with the accomplishment of winning best student paper for the conference:

1. P. Avery, G.W. Greenwood and Z. Michalewicz. *"Coevolving Strategic Intelligence"*. In *IEEE Proceedings for Congress on Evolutionary Computation*, Hong Kong, China, 2008.

The research for Chapter 8 has been used for the following paper, which was under review at the time of submission:

1. P. Avery and Z. Michalewicz. *"Adapting to human game play with Coevolution"*. Submitted to *IEEE Symposium on Computational Intelligence and Games*, in August 2008.

It will also be used to contribute to the following book chapter, which is currently in production:

1. P. Avery and Z. Michalewicz. *"Using coevolution to adapt to human gamers "*. In preparation for *Studies in Computation Intelligence*, with the volume *Recent Advances in Machine Learning*. Springer, 2009.

Chapter 1

Introduction

Every day, all over the world, people are involved in the task of strategic decision-making, particularly in business and defence environments. The success of a business relies on the ability of its decision-makers to reach the best decision on how to appropriate the resources of the organization. Often this decision-making is done in a competitive environment, where other organizations are in direct competition with the business's goals (such as gaining a bigger share of the market). The act of making these decisions over time can be considered a strategic game. In this game, each player must make the right choice, at the right time, to outmanoeuvre his or her opponent. The stakes at risk in these games are high, and the players must use every resource they can garner to their advantage.

One of the most beneficial resources an organization can have is their decision-makers, however people experienced in strategic decision-making can be hard to find. Training someone in the process of decision-making can be difficult, as most good decision-makers act on *intuition* [43] gained through experience.

This chapter introduces our research on providing a mechanism to assist with strategic decision-making training. We begin with the motivation for our research, defining some of the key concepts. We then explain how we addressed the needs identified, defining the objective goals. This includes the general hypothesis for the thesis, and the breakdown of objectives used to achieve this hypothesis. The next section describes the methodology used to address the objectives. We then highlight the original contributions the research has provided. We conclude with an outline of the thesis, describing the contents of each chapter.

1.1 Motivation

Playing is very general term, and includes a large variety of activities, however the ability to play games is universal. Games exist in many shapes and forms, and we learn different elements of strategic thinking through them. Children learn at a very early age to play games, and even to create their own games. Whether they are playing a make believe (imagined) game of cops and robbers, or are defining their own rules to the game of Chess, children have boundless ways of making something fun through game-play. Even at these early childhood stages of game playing, some clear characterizations in the play become apparent [25]. Firstly, the game must be fun. As soon as the player loses interest in the game, it no longer becomes a game to him or her. Secondly, the game must have a set of rules, even if there are no explicit rules. For example, in the make believe game of cops and robbers players create their own rules to characterize the game scenario. The cops must follow a certain behavioural etiquette to show they are cops, and similarly with the robbers. Lastly, if the outcome of a game is known before hand, the motivation to play the game is lost. The result of the game must be uncertain in the sense of not being able to predict the 'winner'. Whether it is the uncertain flow of where the imagination might lead us, or the thrill of competition, the consequence of a course of action continues to entice the player.

By playing, we learn general social concepts through interaction, and trial and error. We also learn the ability to form strategies to achieve our objectives. Developing and fine-tuning our strategic abilities through game playing helps to prepare for the real-world task of *strategic decision-making*. The definition of a strategic decision is hard to give, however Schwenk [91] provides the following characteristics. Strategic decisions are *ill-structured* and *nonroutine*, with an element of uncertainty in action and outcome. If there is a clearly visible path to take (for example from a set formula) for a decision, then it is no longer a strategic decision. The outcome of the decision is also an important aspect of strategic decision-making. If the outcome does not have a large impact on the goals of the decision-maker, it is no longer a *strategic* decision. Lastly, the decision-making process itself should be complex, for example through high information overload, temporal changes to the problem, and dynamic environments. Enterprises must be efficient at strategic decision-making to be successful, and the complexity and difficulty leads many enterprises to fail at their goals.

As identified in [92, p.486] *"Research on strategic decision-making is difficult to do because of the difficulty of observing the decision process in action"*. This difficulty also affects the ability to train people in the process of strategic decision-making, and to grade them effectively. The use of a controlled "laboratory" style allows observation of

the decision-makers in a strategic decision scenario. By providing the decision-makers a scenario to implement their strategies, it is possible to record the decision processes, and identify factors that they themselves might not even be consciously aware of [92]. The art of skilful decision-making is something many people seek. There is no 'magic bullet' solution and like most things, it requires practice and determination.

As stated by Donald Trump [100, pp.43-44] *"Money was never a big motivation for me, except as a way to keep score. The real excitement is playing the game."* To truly succeed in strategic decision-making, you need to bring out the love of the game (in the sense of a game as a generalized concept, e.g. success in business). To do this, the scenario needs to be challenging and attainable.

To create an opportunity for decision-makers to practice their skills, they need to be provided with an abstraction of a strategic decision-making scenario. This abstraction needs to be contained and follow well defined rules, so actions can be observed and analysed. Additionally, many strategic decisions must be made in a dynamic competitive scenario, where another enterprise is running in direct conflict with an organization's goals. The competitive nature of strategic decision-making in corporate and military alike, often involves investigation into the opposition. This investigation involves gathering intelligence about the competitors. As a result, an obvious corollary of this is for the organization to attempt to stop competitors gaining access to their strategic information, by creating counter-strategies. This is a key concept in strategic decision-making, and requires different strategy creation mechanisms. This concept of intelligence and counter intelligence should also be provided in any abstraction for training purposes.

To create an abstraction for this scenario, a challenging opposition must be used that can also make its own strategic decisions with its own goals. The need for an opposition creates a dilemma, as it requires another player who is capable of adapting to the decision-maker's strategies. A human player can be hard to find, as people with the right experience and skill levels to adapt at the necessary level of the player should be used. Additionally, the time taken to train each player at their individual level creates an unrealistic level of demand for the human trainers. The use of a computer player as opposition is therefore ideal, as a human player can make use of the computer player at any time, for any length of time. The problem lies in creating a computer player that can adapt to the level needed by the human player in training. Additionally, the computer player needs to create strategies that can cater for the dynamic decision-making scenario. These tasks are not simple ones, and are not something that is currently available. It is the creation of a computer player for this purpose that is the motivation for this research.

1.2 Objectives

As stated in section 1.1, our main objective for this research is to create a computer player that will be effective in helping humans to learn good strategic decisions. For the purposes of this thesis, we describe a computer player as a strategy for game-play. In the future, there may be a distinction between a computer player and its strategy, but currently they are the same thing.

To achieve our goal, we define the general hypothesis for this research as: *"Coevolutionary algorithms are an effective mechanism for the creation of a computer player for strategic decision-making games"*. To be effective, the computer player must develop strategic rules in a *reasonable* time (the game-play must be continuous, with no detrimental impact on the play caused by time delays), and should provide a challenging learning experience for human players.

To further this objective, we use a specific game that was developed to train Department of Defence personnel in *resource allocation*, which is a strategic decision-making task. The game is the TEMPO Military Planning Game, and it represents a simplified abstraction of real-world resource allocation in a dynamic environment. Therefore, we now narrow our main objective to be specifically for this game. The methodology described in this research can be applied to numerous scenarios, but the practical experimentation has been conducted using TEMPO.

Following from the motivation, the developed system must meet the following objectives:

1. The computer player must be able to allocate resources effectively in the game of TEMPO, for the purpose of being competitive against a human player.

2. The computer player and the TEMPO game should provide the necessary scenario for the human player to learn the task of resource allocation.

3. The challenge given by the computer player should be tailored to the individual human's ability level.

The challenge of the first objective is in evolving a computer player that can create rules for strategic game-play, given the competitive and dynamic nature of TEMPO. The second objective requires analysis of the computer players being generated, and the game of TEMPO itself. To create a challenging scenario for human players to learn with, the environment of the game needs to be designed to test their skills. Additionally, the computer player should be challenging enough to prompt human learning. Combined, the game and

the computer player form the scenario of a challenging learning task. The final objective is perhaps the hardest to achieve, with the difficulty level and strategy generation of the computer player adapting to the human player.

1.3 Methodology

The TEMPO game was created to allow decision-makers to practice their skills in a competitive environment. The game allows strategies to be implemented in a zero-sum allocation, which can be analysed for effectiveness. This auditing process allows the perfect opportunity for players to analyse and improve their decision-making, increasing their experience and 'intuition'. By creating a contained practice environment, the decision-makers are given a clearer understanding of how their strategic decision-making process is happening. As identified by Schwenk [92, p.488]:

> *"Executives cannot always recall the details of complex decision processes. What is worse, they tend to reconstruct events in a way that makes the processes seem more purposeful and logical than they actually were. They do not do this from a desire to deceive. Indeed, they may not even be aware of what factors affected their decisions."*

This is a common scenario in strategic decision-making in general, as normally information overload and dynamic factors affect the decision. By providing an environment where the complexity is managed, and the actions recorded, the TEMPO system allows decision-makers insight into their techniques. Good decision-makers are experienced in the *way* decisions are made, not necessarily the context of the particular decision. By providing potential decision-makers with a competitive game to practice their general decision-making skills, they are shown the challenge and fun involved in finding a good strategy. The game of TEMPO was designed for this purpose, and by creating a computer game with a computer player adversary, the auditing is made even easier. The TEMPO also game provides the ability to practice strategies with intelligence and counter intelligence.

As explained by De Jong [39], it can be difficult to define an evaluation function for a computer player, as the best evaluation is how well it plays the game. When evaluating against a static set of strategies, the computer player is only trained for those specific strategies. The use of coevolution to evaluate the game-play however can provide a much greater variety of evaluation strategies. Thus, a coevolutionary algorithm was considered the best mechanism to create a challenging opponent for the TEMPO game. The use of coevolution to successfully create computer players in many different types of games has led to much

research on the topic over the past decade. The application of the technique to an imperfect game such as TEMPO, where not all the information on the game is made available to the players, has not been as prolific.

The complexity of decision choice in imperfect games means that rules of game-play can be complex and hard to determine. The individuals for the coevolution in this research are represented as a fuzzy logic rule base. Fuzzy rules were used to allow the coevolutionary system to create general rules of strategic game-play, which were easily readable for analysis purposes.

This research continues the work started by Johnson et. al. [55] on a computer player for the TEMPO game, specifically for the objectives explained above. We analysed and addressed the deficiencies of the previous computer player. This included the addition of a memory to the coevolutionary system, to create more robust and challenging static computer players. Additionally, the game of TEMPO itself was analysed for deficiencies, particularly the intelligence and counter intelligence game components. As a final step, we created a new process to provide human players with a test environment for the game of TEMPO. This involved the creation of an adaptive process for the coevolutionary system. A user study was then conducted to test the overall effectiveness of the system.

1.4 Contributions

There are three major contributions from this research, which are described in depth in corresponding chapters in the thesis. The first major contribution is research into the addition of memory to the coevolutionary algorithm. We investigated different ways to use the memory, specifically selection from the memory for evaluation of the coevolution individuals (represented as strategies for the game). We demonstrate the benefit of using a mechanism for selection based on the way humans use short and long term memory. We also made additional experiments to further mimic the use of long term memory through ranking and clustering.

The contributions for this research have been published in [8–10]. The research met the first objective as required in section 1.2. Using the memory addition to the coevolution, we were able to develop computer players whose strategies were very challenging for human players. The generated players were however still static representations of a strategic rule base, for a particular strategy of game-play. While a computer player was able to best human players in initial game-play, the human players were able to adapt to the strategy over the long term.

The second major contribution was to analyse the role of intelligence and counter in-

telligence. This involved investigation into the deficiencies of the computer players and the game of TEMPO, in the intelligence and counter intelligence aspects of the game. We collaborated with Garrison Greenwood for the research, who has experience in both computational intelligence and as a Lieutenant Colonel in the U.S. Army. Through the collaboration, we were able to identify improvements that were necessary for the TEMPO game to be a more realistic and useful tool for intelligence and counter intelligence in resource allocation. Changes were made to create an intelligence mechanism for the game that allowed partial purchase of intelligence in the required categories, replacing the previous boolean decision mechanism. The counter intelligence was also changed in a similar manner, with experimentation on the way to apply the purchase of counter intelligence in the game. Deficiencies in the coevolved players were also investigated, through the changes to the TEMPO game, and experiments with parameters that were thought to affect the purchase of intelligence.

The research showed that the representation of the intelligence and counter intelligence was a major factor in the game-play. However, we also identified the difficulty in trying to coevolve individuals who recognized the importance of buying intelligence, instead of simply finding the best generalized solution. Findings from this research were published in [7], for which we were fortunate enough to be awarded best student paper at the Congress of Evolutionary Computation 2008. The research aimed to achieve the second objective of providing a challenging scenario for resource allocation. While this was addressed to some degree, there is still more progress to be made to the usefulness of the game of TEMPO.

The final contribution of research was to create a system that adapts to a human player, creating a tailored game-play experience. The complete system developed by this research contributed a novel way of adapting to human game players by *reverse engineering* the human game-play. The reverse engineering involves recording data from a human's game against a computer player, and performing an optimisation search to find rules of game-play that follow the same manner of game-play as the human player. The reverse engineered rules are then used in the TEMPO coevolutionary system to adapt to the human player's strategy. The coevolution is performed real-time, with an average of less than two minutes for the computer player to form a new strategy. The combination of reverse engineering the human strategy in this way, and combining it in a coevolutionary system to develop a computer player that adapts to a human player, is a unique contribution. The system worked very well overall, and provides a promising start for future research.

1.5 Thesis outline

The thesis is organized as follows. Chapter 2 gives a background to the general topics surrounding our objectives. We begin with a discussion on the field of game theory, as it offers many insights into factors concerning this research. The discussion then progresses into the use of computers for the implementation of strategic game playing, with the emergence of artificial intelligence (AI) and in turn computational intelligence. Some of the relevant computationally intelligent techniques used by this research are then described. We give a description of the general evolutionary algorithm and its components, followed by the coevolutionary algorithm. We then conclude the chapter with a brief description of fuzzy logic and fuzzy controllers.

Chapter 3 proves a detailed introduction to the game of TEMPO. The chapter begins with a history of the game and its origins. We then go into the specifics of how the game is played, and the structure of the game. This chapter is intended to give the reader an overview of the game purpose and mechanisms. As the game is changed at various stages throughout the research, it is only a superficial introduction to the game, with deeper understanding gained as the thesis progresses.

Chapter 4 provides a literature review of the work conducted for topics relevant to our research. The first of our research presented in this thesis is the addition of memory to the coevolutionary algorithm, for creating a more efficient and challenging computer player. Relevant work into the addition of memory for coevolutionary algorithms is discussed, with problems, benefits, and research needs addressed. This is followed by a general discussion on the research into the analysis and creation of strategies by artificially intelligent means. The difficulties involved in this are also discussed, as is the current research on these issues. In relation to this, we give a review of the different computationally intelligent mechanisms that have been used to create strategic computer game players, particularly relating to the evolution of the players. We then discuss research into games with similar characteristics to the TEMPO game, considering the current research in the area, and the similarities and differences to the research on TEMPO. We finish the chapter with a discussion on the research for creating a computer player that adapts to human game players.

Chapter 5 goes into detail on the technical aspects of the system used to create our computer players. We describe the previous research into the creation of a computer player for the game of TEMPO, and the changes implemented for our research. We give a detailed description of the specific coevolutionary algorithm used in the creation of the players, and the evolutionary mechanisms implemented. This includes a description of representation used for the individual computer players, followed by the coevolutionary algorithm

implementation.

Chapter 6 describes the first experimental stage of research for this thesis. This stage addressed flaws found from the previous research into creating a computer player for the TEMPO game. The addition of a memory population to the coevolutionary algorithm was implemented, and research conducted into the best way to use the memory. Extensive experiments were performed on the method of selection from memory, where selected individuals were used as part of the evaluation function. The growth of the memory population over the generations affected the selection pressure, and diminished the usefulness of the memory. Instead of restricting the size of the memory, and losing individuals, we experimented with ways of varying the selection pressure based on the concept of human short and long term memory. We found the technique gave us a distinct advantage, and proceeded to investigate further into different ways of mimicking the human long term memory. The chapter provides a description of the background research and analysis that led us to the addition of a memory. The different experiments conducted and their results are provided, with analysis and conclusions given.

Chapter 7 covers the next stage of the research that addressed the issues of intelligence and counter intelligence in the game. The work in this chapter describes the changes made to address the deficiencies in the TEMPO game, by creating a more realistic intelligence and counter intelligence mechanism. We also describe the experiments conducted to evolve computer players that make use of these changes.

Chapter 8 is the final chapter for the research in this thesis. The research involved the amalgamation and extension of the previous research, into the Human Adaptive Coevolutionary Process (HACP) for training human players. The system plays against a human, and adapts to the human strategy, creating a challenging and fun learning experience. As described above, it is this ability to provide a fun challenge to human players that ultimately allows them to develop their intuition for strategic decision-making. The chapter describes the creation of the HACP system, and the mechanisms used to adapt to the human player. We then provide the results from a user study where human players with nominal experience in strategic decision-making use the HACP system. The results were very promising, and an analysis of the benefits and the avenues for future research are discussed.

Chapter 9 concludes with a general summary of our findings and the future directions for research. The specific findings for each research stage are given in their relevant chapters, and this chapter will instead provide an overview of the findings from the thesis as a whole. We also describe some of the deficiencies and difficulties that exist in the research, and possible ways that they could be solved.

The research presented in this thesis has demonstrated the exciting prospect of coevo-

lution for strategic decision training purposes. We establish that the use of coevolution is indeed possible, and that it can create an effective computer player for resource allocation games. In the process, it also allows human players a fun and challenging way to gain experience in the area. We hope you enjoy the story of the research presented here, nearly as much as we have creating it.

Chapter 2

Background

In this chapter we describe some of the relevant areas of research, and provide some background information. We begin by describing the field of strategic decision-making, and discuss the difficulties involved in the process. We also give a description of the specific strategic decision-making task used by the TEMPO game, that of resource allocation. Next we give a summary of some general game theory, as the field of game theory is paramount to understanding the basic principles in strategic decision-making. We the describe the effects that games have had on computer science, and their continued importance. We then describe the different areas of computational intelligence utilized in this research. This begins with an overview of evolutionary algorithms, a general description of their purpose, how they are used and what is involved in their creation. We then extend our description of the evolutionary algorithm, with the coevolutionary algorithm. We compare coevolutionary algorithms to evolutionary ones, presenting the advantages and disadvantages. Finally we discuss the fuzzy logic rule base mechanism chosen to represent individuals in the system. This involves a brief explanation of the concept of fuzzy logic, and how it applies to fuzzy control systems.

2.1 Strategic Decision Making

The research into Strategic Decision Making can generally fall into two separate areas [92]. The first is analysing and creating the content of strategies. The second is observation and analysis of the process involved in the decision-making. The creation of general strategies can prove beneficial to the decision-making process, but generalized rules on strategy making can often prove too generic for situation-dependant real-world problems. Instead, by observing and analysing the way that strategies are created, guidance can be given to help decision makers in the process.

One of the oldest and most universal research streams on strategic decision-making is *rationality* and *bounded rationality* [43, 91]. Rational-choice (rational) strategic decision-making, is when strategy making results from logical decisions. As stated by Eisenhardt and Zbaracki [43, p.18] *"According to this model, actors enter decision situations with known objectives. These objectives determine the value of the possible consequences of an action. The actors gather appropriate information, and develop a set of alternative actions. They then select the optimal alternative."* The reality of problems however can contradict this ideal, with inconsistent goals existing among decision makers, and over time. Often the decision-making process can occur over long periods of time, with decision makers coming and going, and the information relating to the problem changing. Many alternative views of how strategic decisions eventuate have been suggested, with the rational thinking being bound by circumstance (bounded rationality).

For example, in many organizations the decision often reflects standard operating procedures and guidelines. This decision is made regardless of whether it is the most beneficial optimal alternative. The ability for different individuals with different agendas to all agree on a single decision also complicates matters. Often parties will rely on expertise from external consultants, or a devil's advocate to argue the inadequacies of the decision. Most commonly the success of the decision however, relates to the politics and power of the parties involved [43]. If a decision maker has a position of higher standing, obtained through past achievements or political prowess, he or she will often be given the final say. Obtaining this position of power may relate directly to the ability of an individual to quickly decide on a successful action. Studies [43] have shown that this quick decision-making is achieved through shallow research of relevant information, thus creating a bounded rationality strategy with incomplete information. As found by Eisenhardt and Zbaracki [43, p.22] *"Decision makers satisfice instead of optimize, rarely engage in comprehensive search, and discover their goals in the process of searching."* Often this ability to act on incomplete information is described as 'intuition' gained from experience. The ability to make a successful decision is something that comes with practice and experience.

When applying the science of strategic decision-making to a military organization, the similarities with corporate strategy are strong. Socrates himself described the similarities between a businessman and a military officer, showing that both make plans to allocate their resources according to their goals [21]. One key difference however, is that military strategy making can be broken into two different stages: *deterrent* and *combative* strategies [98]. The aim of the deterrent stage is to avoid going to war, usually by displays of dominance over an opposing country through force of arms. The combative stage has different objectives, with the communication/relationship management and resource allo-

cation aimed at ending the war favourably and quickly. There are also different strategies that emerge depending on the type of war being fought: offensive, defensive, pre-emptive, subversive etc.

In the past, one of the differences between military and corporate strategic analogies has been the clearly defined *terrain* of the war. As stated by Sudharshan [98, pp.28-19]:

> *"Military strategy is implemented in a terrain that is clearly defined and known to the participants and does not change over time. The valleys remain valleys and the high ground remains high ground. However, the marketing "battlefield" is constantly changing. In addition to external forces of change such as technology and customer values, businesses are continually changing the relationship-offering space as they attempt to erode advantages established by competitors."*

This emphasis of a static war scenario has shifted in recent years however, with modern warfare tending to focus on cell structured guerrilla warfare. The strategies involved in modern combative strategies must be adaptive, even in relation to physical terrain. While a map can show physical valleys, they will not show the hiding spots of the guerrillas that know the areas better than the soldiers do. They can not predict, when approaching a village, if it has allegiance to the guerrilla cause and to what degree. The battlefield itself has become an uncertain and dynamic environment, and the strategy making process needs to adapt in response.

One of the key factors in strategic decisions is how to allocate the resources of the enterprise to achieve their goals. The next section considers this area of the decision-making process in more depth.

2.2 Resource Allocation

The concept of resource allocation is intertwined with strategic decision-making. As Bracker [21, p. 221] summarizes *"Strategic management entails the analysis of internal and external environments of a firm, to maximize the utilization of* ***resources*** *in relation to* ***objectives****."* The resource allocation made by an enterprise, creates the artefacts (the action results) of their strategic decision-making. The mapping of resources to artefacts adds further complexity to the strategic decision-making process. While the high end goal is to allocate resources according to the objectives of the enterprise, often the sheer mass of options available for the allocation creates further complication through information overload. The individual pros and cons of solutions can be lost in the magnitude of choice. Add

to this constraints applied by the organization, and the differing objectives (both of which often change over the course of the decision-making process), and the problem difficulty becomes daunting.

While resources can include any asset of an organization, including its people and end products, often the resources are referred to in monetary and budget terms. Generally, the successful allocation of the budget can be thought of as an optimisation issue, with objectives relating to the company goals. Often there is not just one goal objective, and multiple objectives must be met. The internal problem environment might also change in the organization, as people and goals change, and different interested parties influence the decision-making process. External factors also play a role in the complexity with competitors, allies, and environmental issues out of the organization's control all contributing. Each factor has the capacity to change the significance of the resources and their worth, and create the need for an adaptive allocation process.

The use of computers should be beneficial to the decision-making process. The processing power of computers to determine optimal solutions for decision makers to choose from provides a simplified choice. However, the dynamic nature of the decision-making process makes the development of a computer decision support system difficult. For a computer program to be useful, it either needs to adapt to the changing goals and environment, or be rewritten to encompass the changes. Often this difficulty results in impractical solutions to the decision-making process.

While a decision support system may help provide solutions, the bottom line is that a human needs to make the final decision. If the human is not experienced enough and educated in the situation, he or she may make the wrong decision. Hence, we need to find a way to give the human decision makers practice at making decisions, in a changing and dynamic environment. By providing decision makers with a contained environment to practice their decision-making skills, the human players are able to learn through experience. The testing environment also gives future employers a mechanism to find and train employees that demonstrate the ability to make good decisions in a generalized manner.

Game theory has given much insight into the heuristics that humans use in decision-making. The next section discusses some of the concepts in game theory that are relevant to this research.

2.3 Game Theory

Game theory research has provided much of the foundations for computer players. Some of the great founders in the field of computer science, such as John von Neumann, were

also scholars of game theory, and much of the theory of computer science has been affected by it over the years.

The main purpose of game theory is to interpret and represent the scenarios involved in strategic decisions. By analysing and understanding the behavioural strategies, optimal strategies can become apparent. Game theory research has wide reaching applications, from understanding social and individual behaviour in social sciences, to analysing and predicting political strategy. It is a wide and varied field, and this background only describes some of the more relevant topics to this research.

This section begins with a discussion of how game theory has influenced computer players over the years. We then discuss some of the common techniques that have been utilized by the Artificial Intelligence (AI) community.

2.3.1 Game Theory and Early AI

The use of strategy in games has been analysed and taught for many centuries in many cultures. However, it was only recently that a research field has emerged that investigates the way people analyse decisions in a generalized manner, rather than for a specific game. John von Neumann and Oskar Morgenstern largely founded this field by publishing their 1944 book "The Theory of Games and Economic Behaviour" [102]. Hargreaves Heap and Varoufakis [51] summarize the definition von Neumann and Morgenstern gave to a game as follows:

> *"They [von Neumann and Morgenstern] defined a game as any interaction between agents that is governed by a set of rules specifying the possible moves for each participant and a set of outcomes for each possible combination of moves. One is hard put to find an example of social phenomenon that cannot be so described."*

This identification of games representing the interaction of social phenomenon is an important one, as it indicates the ingrained game playing that human behaviour comprises. Whether the specified rules are a set of statements to define a board game, or a set of legal, ethical and moral rules defined by social circumstance, all of these create a constrained environment for strategic decision-making. As a result, any form of study that seeks to investigate human behaviour in these environments, stands a high chance of crossing over with game theory. Thus, the research into AI, which has been largely based on finding ways to recreate intelligence at a human level, has remained closely linked to the field of game theory.

Game theory has produced some important observations on optimal decision-making, and how people make their decisions. Research for game theory often involves representing a social interaction in a constrained version of a game. This approach trivializes the details and reality of the circumstances, and instead abstracts the situation into a simulation test bed [18]. The game scenario allows two or more players in opposition to (and in some cases in cooperation with) each other, making strategic decisions on the scenario presented. This scenario provides a test bed to determine the actual heuristics that people use to play the game and make decisions in general.

When game theory research uses a constrained test environment for a scenario, it is often beneficial to perform the test using a computer. A computer allows for faster and more global testing of a hypothesis. In turn, the constrained abstractions provided by game theory are ideal for testing computer science algorithms for optimization and 'intelligence' heuristics. This mutual benefit ensures the ongoing overlap between the two fields. It is through the abstraction of scenarios that some interesting research topics can be modelled and tested.

2.3.2 The Prisoner's Dilemma

One of the most well known scenarios in game theory is that of the *Prisoner's Dilemma* (PD), which allows study of complex strategic decision-making. In the PD, police are questioning two suspects in relation to a crime. The police do not have enough evidence to convict both suspects for the maximum penalty of 10 years, but there is enough evidence for a minor conviction of 1 year. The police offer each suspect a deal. If the suspect confesses and agrees to implicate his or her partner in the crime, then he or she will receive a reduced sentence (from the 10 year maximum). If the partner chooses not to confess (and stay quiet), the suspect will go free and the partner will receive the maximum time of 10 years. If the partner however also confesses, both suspects will receive a sentence of 5 years jail time. The suspects are also aware that if they both stay quiet, the police only have enough to convict each of them of the minor crime. Each suspect must make his or her choice without talking to the partner [41, 69]. Therefore, we have a scenario where each suspect has the penalty matrix as defined in figure 2.1. where the years of jail time are given according to each suspect's choice.

To make the situation more general, we can define the decision as *cooperating* or *defecting*. If the suspect chooses to cooperate with his or her partner, they will choose the option that gives the most mutually beneficial result. If the suspect instead chooses to go with his or her most individually beneficial option, they defect from the mutual plan. In

		suspect B	
		Stay quiet	Confess
suspect A	Stay quiet	1,1	10,0
	Confess	0,10	5,5

Figure 2.1: Penalty matrix for the Prisoner's Dilemma

this situation, each suspect has the choice to cooperate with the partner and stay quiet, or to defect and confess. These can be thought of in terms of benefits, or *pay-offs*, for each suspect. The better the result, the higher the pay-off. We can assign a pay-off for each scenario from the PD. Each suspect has the highest pay-off when he or she decides not to cooperate and defect, while the partner continues to cooperate. Therefore, in this scenario we give the largest pay-off of 5 for the defector. The opposite scenario, where the suspect cooperates while the partner defects gives the worst pay-off, which we represent as 0. When both suspects cooperate, they receive a pay-off of 3, as they can only be convicted for the minor crime. When they both defect, they are given a pay-off of 1 as the penalty is still a lengthy jail term. These can now be represented in a pay-off matrix, which can be seen in figure 2.2.

		suspect B	
		Cooperate	Defect
suspect A	Cooperate	3,3	0,5
	Defect	5,0	1,1

Figure 2.2: Payoff matrix for the Prisoner's Dilemma

Study of the PD normally iterates the scenario, where the results of one iteration affect the next one. In a scenario of indefinite iteration (like that represented in cold and escalating war scenarios), the Tit for Tat (TFT) method [11] is still arguably the most optimal overall iterative strategy. TFT begins by cooperating for the first iteration, then changes strategy each time the opposition does. If the opposition defects, then next round the player also defects, if the opposition cooperates, next round the player also cooperates.

The IPD depicts a scenario representative of many real-world decision-making problems, such as cold wars (and active ones) and competitive business practices. In all these cases, the mutually beneficial case (both suspects cooperating) seems to give the best stabilized result. However, the temptation to defect and choose the individual best scenario (where the suspect goes free) often results in both parties choosing this option and getting a suboptimal result (5 years instead of 1 if they both cooperated). The PD also has a lot of situation dependencies, like how well the suspects know each other and how much they trust the other to stay quiet. Let us assume for the moment that both parties do not know

each other, and that each party is blind to the other players of the game with no knowledge of opposition strategy. Is there an optimal strategy for them to take?

2.3.3 Nash Equilibrium

In his PhD thesis of 1950, John F. Nash Jr. introduced the concept known as Nash equilibrium [74]. The Nash equilibrium defines a way to find an optimal strategy given games such as the PD. A Nash equilibrium is a steady state in a game, where a choice exists for both players, such that neither benefits from changing their strategy. Nash defined the equilibrium for zero sum games where the players make their choice *"... independently, without collaboration or communication with any of the others [players]"* [74, p. 5]. The PD is a good depiction of such a problem.

Analysing the PD options from figure 2.2, we can determine the Nash equilibrium. Let us start with both suspects choosing to cooperate. In this scenario, they both have a pay-off of 3. If however, a player decides to defect, he or she receives a higher pay-off of 5 (with the other player continuing to cooperate). This shows that the scenario is not in Nash equilibrium, as there exists a state where a player benefits from changing strategy.

For the scenario where one suspect cooperates and the partner defects, we have a pay-off of 0 for the cooperator, and 5 for the defector. This scenario is also not in Nash equilibrium, as the cooperator gets a higher pay-off of 1 if he or she changes strategy to defect.

The last scenario is when both suspects defect. In this scenario, they both have a pay-off of 1. If either suspect changes strategy and cooperates, the result is a lower pay-off of 0. Therefore, in this scenario we have found a Nash equilibrium. Note, the Nash equilibrium is not the most mutually beneficial scenario. If both suspects cooperate, they each achieve a higher pay-off. However, in a scenario where is not complete trust in the other party, the Nash equilibrium can be used to find the stable best option. It is also worth noting that there can be multiple Nash equilibriums in a scenario, and there can also be none.

The Nash equilibrium is just one source of information available to help in making a decision on what strategy to take in game playing. The next section will describe another scenario where different tools can be used for the decision-making process.

2.3.4 Sequential vs. Simultaneous Games

Dixit and Nalebuff [41] describe two kinds of strategic interaction: *sequential* and *simultaneous*. A sequential interaction occurs when players make their actions one after another. All previous actions from all players are visible, along with possible future actions for both

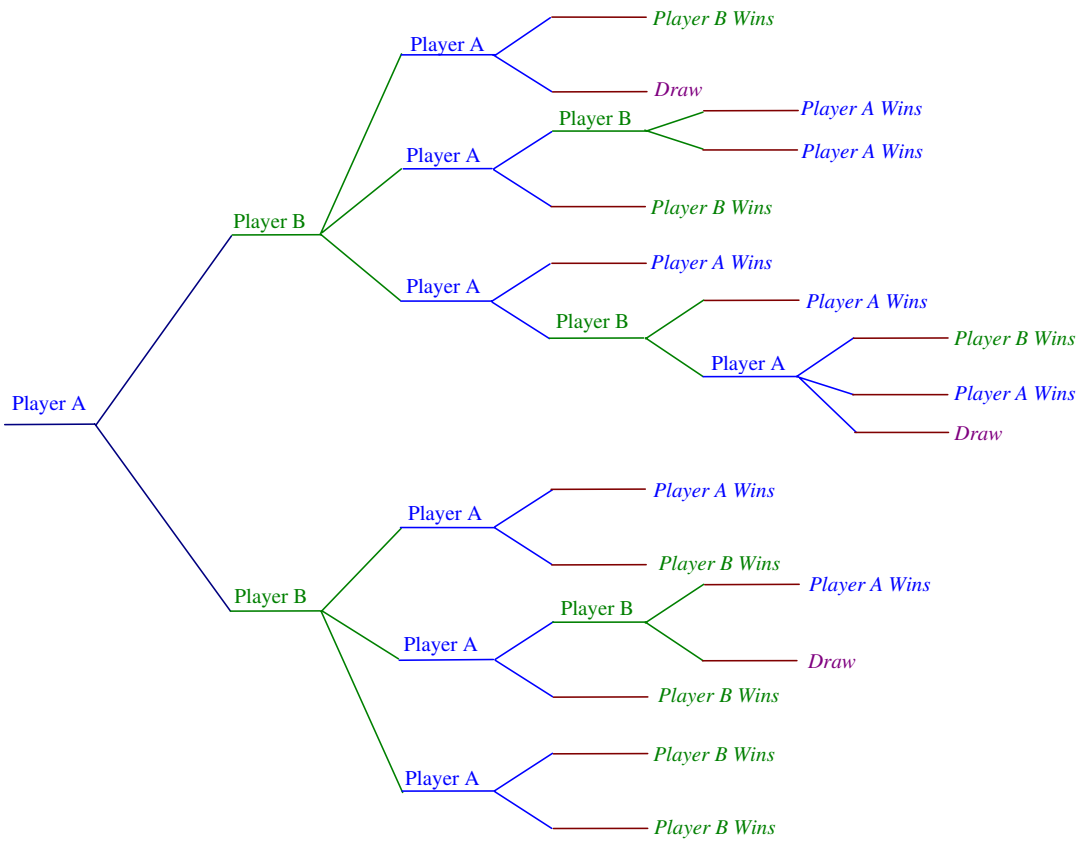


Figure 2.3: Example game tree

players. A simultaneous interaction occurs when all players act simultaneously, without knowledge of the opposition's concurrent actions.

The Prisoner's Dilemma depicts a situation where the player chooses and applies a best strategy for the entirety of the game, simultaneously with the other player. In other games such as Chess, the strategy can change depending on the current situation of the game. In games such as Chess, each player moves sequentially, and the choice depends on choices made in previous moves. These types of sequential games allow the player a perfect view of all possible moves by both players [77], and are thus called *perfect* games. Perfect games give each player *all* the information about the past player and opposition moves, and allows accurate prediction of possible future choices.

The sequential type of game decision-making can typically be represented through a decision tree, where all the possible choices from the beginning of the game to the end can be identified. As an example, figure 2.3 shows an abstraction of a simple sequential game-play.

The game tree starts with the first player (player A) making the opening move by choos-

ing from two options. The next player (player B) then has another set of options depending on what the previous player has chosen. For example, if player A chooses his or her second option (the lower branch), player B will then has three options available. Player B will likely choose the third option (the lowest branch), as it will lead to a definitive win. However, if player A had analysed all the moves at the first step in the game, he or she would be unlikely to choose the second option as it allows player B to force a win. By choosing the first option, player A can either force a win or draw. This tree is a very simple example of a decision tree, and in most games there is a much higher branching factor, and a greater depth (ply) of game-play, which can give a very large search space of decisions.

When a sequential game give a perfect view of choices, it is theoretically possible to force a win or draw from the opening move. Using tree traversal methods, a path can be found where win or draw leaf nodes can be reached regardless of the opposition choices. When such a path can be identified for a game, the game is referred to as *solvable*. A good example of this is the game of Ticktacktoe, where it is a simple task for the player moving first to ensure a win or draw. Thus, the game of Ticktacktoe is *solved*. The reality for many games such as Chess and Go however, is that the size of the game tree is too large for current computational power to solve, and so they are not currently solvable. We are currently not able to, within reasonable time, search every decision and its branches to the depth of the entire game. To emphasise this, even if we just wanted to traverse the tree for Chess to a ply of 8, with an average branch factor of about 35 for each move, this would give approximately $35^8 = 2,251,875,390,625$ nodes to traverse. The size of a game tree for the entire game of Chess makes it solvable in theory only, not in practice. Chess and Go are however still perfect games, as they still allow all player information available, they are just not solvable by today's standards. This makes them perfect, and *theoretically* solvable games.

It is the second category of simultaneous decision-making that applies directly to this research. The area of simultaneous decision-making is where the opponents are making decisions in parallel with each other. They act without all knowledge of their opponent's move, and have an *imperfect* view of the game. This situation requires the players to attempt to see through the opponent's decision. This prediction of opponent strategies can lead to *cyclic thinking*, where predicted actions lead to cyclic strategy making. A good example of cyclic thinking can be seen in the movie The Princess Bride (1973), where the Dread Pirate Roberts engages in a battle of wits with one of the villains of the movie Vizzini. The battle consists of each man drinking from one of two goblets of wine. Roberts poisons the goblet of his choice with the deadly 'iocane powder' out of view of Vizzini, and then Vizzini is given the choice of which goblet to drink from. The following quote is

taken from the movie,

> **Dread Pirate Roberts**: All right. Where is the poison? The battle of wits has begun. It ends when you decide and we both drink, and find out who is right... and who is dead.
> **Vizzini**: But it's so simple. All I have to do is divine from what I know of you: are you the sort of man who would put the poison into his own goblet or his enemy's? Now, a clever man would put the poison into his own goblet, because he would know that only a great fool would reach for what he was given. I am not a great fool, so I can clearly not choose the wine in front of you. But you must have known I was not a great fool, you would have counted on it, so I can clearly not choose the wine in front of me.
> **Dread Pirate Roberts**: You've made your decision then?
> **Vizzini**: Not remotely. Because iocane comes from Australia, as everyone knows, and Australia is entirely peopled with criminals, and criminals are used to having people not trust them, as you are not trusted by me, so I can clearly not choose the wine in front of you.
> **Dread Pirate Roberts**: Truly, you have a dizzying intellect.
> **Vizzini**: Wait till I get going! Now, where was I?
> **Dread Pirate Roberts**: Australia.
> **Vizzini**: Yes, Australia. And you must have suspected I would have known the powder's origin, so I can clearly not choose the wine in front of me.

and on it goes. In the end, the reasoning in this situation was inconsequential, as Roberts had poisoned both goblets (having built up an immunity to the poison). This scenario however, cleverly illustrates the difficulty and complexity in trying to predict what an opponent might be thinking in an imperfect game, where both players must make the next move simultaneously. The simultaneous game-play adds a random aspect to the game, as it is not possible to predict with 100% accuracy the opposition's tactics, and hence it is not possible to solve the game. Thus, simultaneous games are unsolvable, imperfect, and difficult to determine generic strategies for.

It is worth noting that when applying these concepts to real-world problems such as competing companies or political parties, the decision-making process often requires a mix of both sequential and strategic decision-making. To illustrate this, consider the situation where two politicians are trying to outmanoeuvre each other in an election campaign. The decision-making of each of the campaigns is done in parallel. Each party tries to anticipate

what the other's campaign might be, counter-acting with their own strategies. However, if one party goes ahead with a campaign announcement prior to the other party, the situation turns into a sequential one where someone has made the first move. Now the other party must make theirs in response.

Study of human strategy making has given great insight into how they think, and the optimal strategies to use in certain circumstances. The field of game theory was pioneered by greats such as John von Neumann, who were also founders in many computer science fundamentals. This close link between game theory and computers is well founded, as computers became the perfect technology to test the theories being developed. Following on from this, games were recognized as a great test bed for artificial intelligence. This relationship is discussed further in the next section.

2.4 Computers and Games

The use of games to challenge and test a person's abilities has been in place for many centuries. From the Chinese use of Chess and Go to enhance war strategies, to the different types of sport used to test various physical and mental abilities. Games are a part of human culture. This use of games to test the abilities of a person has also been adapted for the pursuit of AI. Through the use of games, computer scientists are able to test the ability of an artificially intelligent mechanism, be it the ability to think, learn or even communicate with other AI mechanisms.

The Macquarie Dictionary [3] defines a game as *"a contest for amusement according to set rules; a match."* Elements of this definition are repeated in other dictionaries, and the common factor is the inclusion of playing to a set of rules. The other important factor about games is the concept of the contest: one player competes against another with the goal of winning. The combination of contesting behaviour whilst obeying set rules makes game-play a wonderful area in which to test AI. The need to develop strategies to outplay the opponent while maintaining the rules allows computer scientists to create intelligent agents in a very testable way.

This section describes the relationship between AI and games, and how development in the field has progressed. We then describe some of the techniques used to create AI for games, including common heuristic and computational intelligent techniques.

2.4.1 The Evolving Field of AI

This relationship between AI and games was recognized early on in the development of intelligent systems when Turing published his paper "*Computing Intelligence and Machinery*" [101]. In this paper, Turing proposed a new mechanism (called the *Imitation Game*) of testing a computer for intelligence, which can be generally described as a guessing game. Turing adapted the following game that comprised three people: a man, a women and an interrogator. The three players are separated, and only communication with the interrogator is allowed. The interrogator then questions the other two parties, and by the end of the game he/she must guess which is the man, and which is the woman. During the game, the man tries to mislead the interrogator into thinking he is actually the woman. Conversely, the woman's job is to help the interrogator come to the correct conclusion. Turing adapted this game to test a computer's intelligence. The computer takes the role of the man, and a human the role of the woman. To pass the test, the computer must successfully play to the same capacity as a human player.

Whether or not the Turing test is a true depiction of a way to test intelligence, the precedent laid out by it is generally accepted. Since the original publication, the Turing test has been used as a milestone to test the success of AI at mimicry of human intelligence. This goal reached a pinnacle when in May of 1997, IBMs Deep Blue supercomputer [54] managed to beat the world Chess champion Garry Kasparov. A milestone had been reached; a computer had beaten a human in one of the most well known intellectual games in the world. However as it was happening, others were questioning the real intelligence of a computer that has been manufactured and tailored specifically to the single task of playing Chess, specifically a manufactured strategy made to beat Kasparov. The question being asked was: is this really what we would classify as an intelligent thought process?

It was with this question in mind that Fogel and Chellapilla set out to create a new form of intelligent computer; one which could learn to play the game of checkers without supervision or instruction [28, 29, 46]. This research opened up a new way for AI to make use of the area of games, as now the aim moved from creating a machine to beat a human, to creating a machine that can *learn* to beat a human. This means machines would no longer need to be manually trained to play specific games. Ideally, they could play any game, against any human, and play competitively. This new process has not only created promising computer players, it has also highlighted the possibility of a truly adaptive, predictive, and learning intelligence. Such intelligence could achieve goals previously only dreamt of in science fiction. The emphasis of this intelligence was no longer to mimic human intelligence, but to allow a computer player to develop its own.

Before we address the technology leading to the creation of self learning computer

players however, we should look at the technology that paved the way.

2.4.2 Commonly Used Methods

Many different types of heuristics have been developed to find the best strategy to use for a game. The heuristics work on the solution search space for the game, trying to find the *optimal* (best) solution (strategy). The categorization of these heuristics can fall into two general types. The first is heuristics that build a solution in steps. The second works on complete solutions, and focuses on points in the search space.

We begin by discussing the first type of heuristics: those that build solutions. The most well known category of these heuristics is greedy algorithms. The greedy algorithm works by analysing possible values for the next step in the solution, and assigning the best possible value. One example of these is the greedy Minimax algorithm, which works in a tree structured search space. The Minimax algorithm traverses the tree by making the move with the maximum advantage to the player, and the minimum advantage to the opposition. This requires a limited set ply depth that represents the next step of the algorithm. The best solution from the specified ply range is chosen. The process then iterates and the tree traversal continues from the branch chosen during the previous iteration. There are many other additions to this method of limiting the search space.

While the greedy algorithm depicts a solution being built consecutively, other heuristics use different mechanisms. For example, it can beneficial to divide a problem into sub-problems and solve these independently, combining for a complete solution. Others work by testing possible next steps, and back tracking to find a good solution (such as A*). All of these techniques are usually quick, but not very efficient at finding the optimal solution.

To find the optimal solution, it is usually more efficient to work with whole solutions in the search space. Thus, we now address the category of heuristics that use complete solutions. Possibly the most intuitive method of finding the correct strategy to use, is to perform an exhaustive search of the possible solutions for the game. In many games however, this is not a possibility. The processing time needed to evaluate all possible solutions is too much. As a result of this limitation, the use of heuristics to pinpoint a near optimal solution is employed.

Instead of searching the entire solution search space, we can define a starting point and then specify a surrounding neighbourhood that is searched for the best solution. There are a number of different ways to perform this search, such as using one of the local search categories of algorithms that involve iteratively searching the points in a neighbourhood. However, this is unlikely to find the optimum from the entire search space (the *global*

optimum), as it only finds the optimum in the defined neighbourhood (the *local optimum*). The solution to this problem is to find a way to break out of the local optimum.

Some of the early algorithms developed to break out of local optimum incorporated tactics such as the simulated annealing temperature variable, which allowed movement to other areas of the search space. Another method is to have a memory structure, such as that used by the tabu search algorithm [48], to force the search into new areas. These methods use different ways to break out of the local optimum. Simulated annealing uses randomisation to find a probabilistic solution whilst tabu search deterministically climbs through the search space.

These methods, and in fact all methods mentioned in this section, have one thing in common: they work on a single solution at a time. Generally, it can be said that many of the single solution heuristics used to create game strategies were inefficient in their ability find an optimum solution. Section 2.5 looks at the evolutionary algorithms that were implemented to work on a number of solutions at the same time [69]. The next section however concludes the computers and games section, by discussing one of the common techniques used to represent computer players, with an expert system.

2.4.3 Expert Systems

Traditionally, an expert system for a particular domain was created through consultation with one or more domain experts. The gathered information was then used to create a knowledge base. Often this knowledge base was represented in the form of rules that regulated what action or decision was to be taken, given a set of input values. These rule sets acted as a static representation of the knowledge of the experts at the time of the system development. The expert system did not have the ability to change without constantly requiring the experts to give updated information – thereby requiring a great deal of time that experts in the field do not have. The static design also meant the system often came across situations where it lacked a form of 'common sense'. For example, in one case a medical diagnostic expert system asked if a male patient was pregnant [67]. While this depicts a bad initial design, it highlights the difficulty in covering all possible avenues of decision-making. When using a static system that needs to be reprogrammed to include each newly 'discovered' rule, the whole process can become tedious. While the original expert systems were hampered due to the described problems, they still became common practice in everyday tools such as the decision support systems we can see in (some) medical practitioner's offices.

The concept of expert systems also crossed over into the game field. Many of the first

computer systems that outplayed human champions utilized a knowledge base of strategies and open/end-game databases. The strategies were a set of rules that gave a set procedure to follow if a number of conditions were met, and the open/end-game databases showed all the best moves to make for all the possible beginning or end of game situations. The knowledge bases were created through consulting experts and analysing books for various strategies of the game being played. These would be used to create the necessary rules to enforce the strategies. This is a direct application of an expert system, and when used in conjunction with decision tree open/end game traversal, many systems were able to make sure that the computer player always played the best move. It was through the use of these tactics (plus custom built hardware in some cases) that systems such as Deep Blue [54], Chinook [89] and Lago [85] were able to beat human title holders in their respective games.

The problem with these systems was the same as with the original expert systems: they contained static knowledge of the game, and they were not able to deduce logical conclusions as humans would. Instead, they needed to be manually taught, supervised and tuned until they performed well enough at the single task of beating the designated opposition. This manner of creating a system does not provide intelligence, more the ability to compute and process a greater number of moves and possibilities than a human. As a result, the past decade of creating computer players has introduced research focused on creating a player that develops its own strategies and knowledge about how it should play the game. This is the idea behind the development of a TEMPO player in this thesis. One of the mechanisms for performing this self-learning is through evolutionary algorithms, which are discussed in the next section.

2.5 Evolutionary Algorithms

Instead of trying to simulate intelligence, or create an "artificially" intelligent system, computational intelligence aims to understand the process of intelligent behaviour. Poole et. al. describes this goal as follows [82]:

> "The central scientific goal of computational intelligence is to understand the principles that make intelligent behaviour possible, in natural or artificial systems."

The field seeks to use this understanding to create intelligent systems. Over the years many computationally intelligent mechanisms have been created, most replicating the way humanity and nature provide intelligent behaviour. One of these methods was evolutionary

algorithms, which were inspired by Darwin's theory of natural selection. This section discusses the field of evolutionary algorithms in more detail.

2.5.1 The General Evolutionary Algorithm

Evolutionary Algorithms (EAs) describe a wide range of algorithms that can be used to represent the Darwinian selection model. The idea behind EAs is that instead of working on a single solution at a time, we now have a population of solutions (P) that we are examining at each generation (t) [69]. With a whole population of solutions available, we are able to replicate the process of *natural selection* in Darwinian evolution. With natural selection, every individual in a species is in a struggle to survive, and only the most adaptable and successful of the species add to the survival. We can reproduce this evolution by allowing our population to 'breed' a new set of solutions and then evaluate the new population for appropriate parents for the next population, and so on.

There are a number of different types of EAs, including the well-know Genetic Algorithms (GA) developed by Hillis [52]. However, all EAs generally follow the algorithm shown in figure 2.4, and have a number of common elements. These common elements include the *initialization*, *evaluation*, *selection* and *variation operators*.

procedure Evolutionary Algorithm
begin
  initialize $P(t)$
  evaluate $P(t)$
  while(**not** termination-condition) **do**
  begin
    select $P_{intermediate}(t)$ from $P(t)$
    alter $P_{intermediate}(t)$
    evaluate $P_{intermediate}(t)$
    select $P(t+1)$ from $P(t) \cup P_{intermediate}(t)$
    $t \leftarrow t+1$
  end
end

Figure 2.4: A generic evolutionary algorithm

The initialization of the population typically incorporates a uniformly random procedure. Using GAs as an example, each individual is represented as a binary vector. The

binary vector can then be translated into the *phenotype* (the observable traits) for the problem. For example, converting from a binary representation to a decimal one, as needed by the problem. Each individual bit (called a *gene*) in the vector (which represents the *chromosome* of the individual) is randomly assigned a value.

The evaluation function assigns a quality measure to each individual. For example, when trying to optimize a function, each individual is comprised of possible variables for the solution. The variable values are then fed into the function, with the function itself evaluating the individual. After evaluating each individual, the *fitness* for it is assigned. The idea of fitness is that the strongest (fittest) individuals should have a higher chance of surviving, al la *survival of the fittest*. As a result, the individual with the best (closest to optimal) solution should be assigned the highest fitness.

As shown in figure 2.4, there are two different selection steps for the EA [39]. The first selection is used to determine which individuals should be chosen to generate the offspring and become *parents*. Sometimes this includes all the population and there is only a single selection process. In other variations however, the parents can be a subset of the population $P(t)$ and offspring are chosen from these parents only. For example, only the 'best individuals' might be chosen to generate offspring, with the rest discarded. The second selection mechanism is when the next generation (the *survivors*) are chosen. This involves selecting individuals from the intermediate offspring population (the individuals created from altering the parents), and possibly individuals from the initial $P(t)$ population, choosing survivors based on fitness. The size of the survivor population can also vary.

After selecting parent individuals, the variation operators are applied to create offspring in different areas of the search space. There are a number of different ways to do this, but generally the operators consist of two classes: *recombination* operators and *mutation* operators. The recombination operator takes elements from parent individuals, and combines them into offspring. The most common form of recombination operator is the *crossover* operator. It takes two parents and creates one or more new offspring. To perform crossover, parent individuals are selected and the crossing point(s) chosen, with the segments created from each used to create the offspring. This is demonstrated in figure 2.5, where examples for crossover on binary string representations with single and two point crossover given. The more crossover points, the more the sequence is changed, which can be beneficial depending on the problem and the size of the chromosome. Recombination operators create new individuals, while preserving parts of the sequential chromosome phenotype. The recombination operator is only applied to a certain percentage of the population, at a rate dependent on the problem being solved.

The mutation operator is applied to a percentage of the genes in the population, applying

Single point crossover

$Individual_1 = \{01110010|0010010111\}$ $new\_Individual_1 = \{01110010|0100110101\}$

$Individual_2 = \{00101110|0100110101\}$ $new\_Individual_2 = \{00101110|0010010111\}$

Two point crossover

$Individual_1 = \{01110|01000100|10111\}$ $new\_Individual_1 = \{01110|11001001|10111\}$

$Individual_2 = \{00101|11001001|10101\}$ $new\_Individual_2 = \{00101|01000100|10101\}$

Figure 2.5: Example crossover implementation

changes according to the mutation ratio. Due to the large potential for change, it is typically smaller than that for recombination. It is usually applied after the chromosome sequence changes made from the recombination operator, however many variations on this exist. The next section describes a number of modifications that exist for the EA.

2.5.2 Variations of Evolutionary Algorithms

The EA in figure 2.4 is a very high level outline of a general evolutionary algorithm, and many variations on the implementation details exist. The design of an EA is likely to change for different problems. To design an EA for a problem, the following implementation categories should be addressed: representation of individuals, initialization of the population, the evaluation function, the variation operators, the selection mechanisms, and the termination condition [69]. These are discussed further in this section.

The representation of individuals changes dependent on the problem. Each problem has its own solution space, with some mechanism of representing a solution. These solutions need to be represented in a way that can be applied to the EA. Sometimes it is relatively easy to map a solution directly to a data structure such as a vector. Other times a more complex representation is needed, such as neural networks, or even individual computer programs. When deciding on the representation, it is important to consider how the representation will affect the variation operators. The variation operators need to be able to maintain enough of the previous generations, while providing continued diversity for population evolution. When designing a representation for an individual, the variation operators should also be designed accordingly.

The initialization of the populations has been discussed briefly in the previous section. As mentioned, the standard method is through uniformly random initialization. Once again however, the initialization is problem specific and other mechanisms exist. Sometimes a good mechanism is to force a certain distribution of starting individuals, to ensure coverage of the solution space. Another method is to initialize the population with a set of existing

individuals that represent up to date information. This means the evolutionary process does not have to start from scratch, and can instead focus on improving existing knowledge. Another approach is to supplement random individuals by adding non-random individuals. There are many ways to do this, such as using greedy algorithms to develop good starting solutions, or to use individuals that contain human knowledge. Whatever the initialization design, it is important to maintain diversity to allow the population to continue evolving.

The evaluation is also problem dependent, and is a major factor when attempting to find an optimal solution. The more complex the problem, the more complex the evaluation function. One of the complexities involved in such problems are *constraints*. Often real-world problems do not simply require a solution, but also one that is bound by a set of constraints. For example, let us look at the Travelling Salesman Problem (TSP). The TSP represents a graph traversal problem, where a salesman must visit every node (exactly once) in the graph and return to the starting node in the shortest possible time. In real-world situations representative of TSP scenarios, there is also likely to be other constraints. For example, the salesman may have a time window in which he must travel to a particular city to pick up more supplies. The evaluation of an individual must now evaluate an individual's merit, and its *validity*. If a solution does not satisfy a constraint then it is no longer valid.

There are also different types of constraints dependent on the flexibility of the problem. The salesman example constraint given might be considered a *hard constraint*, since if the salesman does not go to get more supplies, he can not continue his trip. Another constraint might be that the salesman should visit a particular city earlier rather than later, as an important client is there. This however might be a *soft constraint*, which is not considered as important as getting a shorter path.

There exist a number of ways to include constraints into the evaluation process. For example, the rejection of infeasible solutions (known as the *death penalty*) kills any created solutions that are outside the constraint bounds. This can however be problematic if feasible solutions are hard to find. Another method is to repair infeasible solutions. The process of repairing involves a local search for each infeasible individual to find a feasible repaired solution. This improved solution is then applied to the individual either as the evaluation result (the evaluation result for the repaired solution is given as the evaluation result for the original one), or by replacing the original solution.

Another way to include constraints in the evaluation function is to penalize invalid individuals. There are many different ways to incorporate penalties, and the method used is problem dependant. The design of penalties allows flexibility in the type of infeasible solutions included in the populations, which can be beneficial since a good infeasible solution may be just one step away from a good feasible one

Another problem that occurs when designing an evaluation function for real-world problems, is that often there are multiple objectives. For example, when designing a phone for today's market there are many considerations. People want their phones to be as small as possible, with as many extras as possible. They want the applications, the camera, the wi-fi, the touch screen and so on. All of these require more battery and processing power, which makes the size of the phone bigger. There is no clear preference for size over extras. To be competitive in the market the phone designers need to provide both. In this case, a trade-off between the objectives must be found.

The concept of making decisions given multiple conflicting objectives is given the name of *multi criteria decision-making* [69]. Sometimes it is simply a matter of finding a way to assign each objective unit with a given weight, and then translating the objectives into a single objective problem. Often however, the objectives are not directly comparable. To pick a solution for such problems (a *multiobjective optimization*), it is necessary to find the set of best solutions, given the objective trade-offs. This involves finding the sets of dominant and dominated solutions. Any solutions that are worse on all objectives than a given solution, are dominated by that solution. For example, figure 2.6 represents a solution space for a generic minimizing problem with two objective functions. We can see that solution z has a higher outcome for both objectives than solution y does, thus making it worse than y (with minimization). So z becomes a dominated solution, and y a dominant solution. However, z is not dominated by x, as z has a better outcome for objective 1. There can also exist a number of solutions that are all equally beneficial from some perspective (given all the objectives), but favour different objectives. This can be seen in figure 2.6, with solutions x and y.

Solutions that are not dominated by any other solutions are *nondominated*. If we find all the nondominated solutions for a search space, these form the *Pareto-optimal* set. In the case of figure 2.6 we can see that the Pareto-optimal set would consist of $[x, y]$ as neither of these solutions are dominated. When optimizing a multiobjective problem, it is beneficial to find a set of nondominated solutions that are as close to the Pareto-optimal set as possible. Once this set has been found, it becomes necessary to make a hard line choice of which strategy in the set to use. Due to the nature of multi-objective problems, the solutions found will not give 'the optimal' solution as they are biased towards different objectives. At this stage, it is necessary to analyse the importance of the objectives themselves and determine what degree of bias in the solution is deemed acceptable.

We now look at the choices available for the variation operators. There are literally hundreds of variations for recombination and mutation. Gwiazda has published a comprehensive review of crossover [49] and mutation [50] methods for GAs that alone add up to

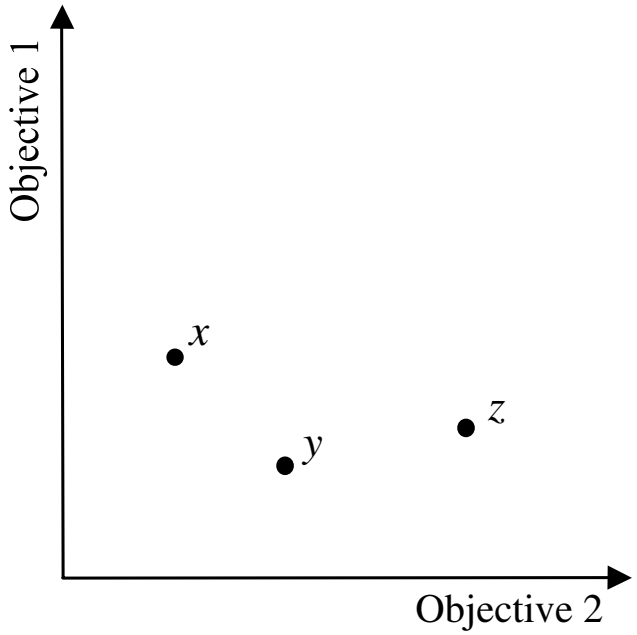


Figure 2.6: Example of solutions in multiobjective space

over 300 different techniques. Gwiazda has classified the operators based on the GA type, such as binary operators and real operators. The types of operators for EAs are even more extensive, and their classification can be difficult due to the diversity. Many types of recombination operators exist that take more than two parents, and have varying techniques of combining these parents. Additionally, the recombination operator can also combine other heuristics, such as a local search method to influence the evolutionary search [104]. Finding the right type of operator for a given problem can be can be a time consuming process of trial and error. In many cases (when possible), the ease of applying a tried and true method (such as k-point crossover) wins out over the benefits of other lesser known ones.

There exist a number of different selection methods, and again, the choice depends on the problem being solved. Three well-known methods are roulette wheel, tournament and rank selection. The roulette wheel selection assigns each individual a probability of selection proportional to its fitness (in relation to all the other individuals). A higher fitness value will give a larger probability of selection. The tournament selection method works by selecting individuals (usually randomly) and comparing them to other selected individuals. The individuals with the higher (winning) fitness are then selected as the parent/survivor. A typical tournament involves selection of two individuals for competition, but k selection

can be used where k individuals compete tennis tournament style (except losing individuals go back in the running) to find a winner.

Rank selection involves selecting based on the rank of an individual rather than the fitness. Each individual is ranked according to the fitness, and the selection is then proportional to its rank. Selection by rank can help to discourage *premature convergence* onto a sub-optimal solution. When selecting by roulette, individuals with higher fitness are likely to be selected many times. The resulting population can be saturated with the individual, hampering diversity. If the individual is a local optimal, the population has prematurely converged before finding the global optimal. Using ranking, the proportion for selection is controlled, and greater diversity can be maintained.

As mentioned, convergence occurs as the 'better' individuals are chosen to be copied to $P(t+1)$, and over time saturate the population. At the other end of the scale, sometimes there is not enough selection pressure, and good individuals can miss out on selection. This can be countered somewhat with the use of a *elitist model* [38], which ensures the best individual(s) from the population is included in the next generation.

The different selection steps, whether selecting parents or survivors, and the different selection techniques have a substantial impact on the EA performance. As shown, each of these methods has advantages and disadvantages. It is important to consider these when deciding on which mechanism is the best for the problem at hand. For further information, De Jong provides a comprehensive analysis of effects of selection, along with other elements of the EA in [39].

The termination condition also has many different forms [90]. The evolution could be terminated based on a time limit or generation count. Another way is to terminate once the population has converged on a solution and diversity has been lost. Termination could also depend on the gain made by the population, if the progress of the evolution slows, it could be terminated. Other times there can exist a cut-off threshold where the evolution stops when a sufficiently 'good' solution has been found. Often a combination of these methods (and others) is used.

As a final addition to this section, it is worthwhile mentioning the combination of approaches that can form hybrid systems. Often advantages can be found with different approaches [90], such as creation of individuals through local search methods, or representation of individuals with fuzzy rules or neural networks. Different methods can also be used for the decoding from genotype to phenotype. These different techniques can then be merged with EAs to create adaptive solutions, and thereby gain the benefits of both approaches. There are many different ways and forms of hybridizing EAs with other approaches, and sometimes the complexity of the systems can outweigh the advantages, but

overall there are also many benefits to be gained.

In addition to variations in the representation of the algorithm components in figure 2.4, there also exist extensions that add to the algorithm. The next section discusses one of these extensions, specifically the one that is used in the thesis; the Coevolutionary Algorithm.

2.6 Coevolutionary Algorithms

Evolutionary algorithms cover a wide range of different types of systems, all of which perform the same evolutionary function of pushing for survival of the fittest. One extension to the EA is the coevolutionary algorithm that is described here. In an EA, competition occurs among individuals in a population, with each trying to achieve the highest fitness. With a coevolutionary system, there exist two or more different species of individuals, usually represented in their own populations (although they can also exist in the same population). The individuals from one species then compete against other species, and their fitness is based on how well they do against the other species. As a result, each species continually influence each other's fitness landscapes. This process allows flexibility in the evaluation of individuals, as there is no defined optimal solution being sought. Instead, the individuals are encouraged to learn solutions for competitive (or cooperative) behaviour through a process of trial and error.

Hillis originally made this process popular when he used a parasite vs. host scenario to evolve sorting networks [52]. Hillis demonstrated the concept of two competing entities that, through the competitive process, evolve new and innovative ways to outmanoeuvre each other. This concept can be visualized as the struggle between rabbits and foxes. The rabbits are struggling to exist by not getting eaten by the foxes, so they develop stronger leg muscles to out-run the foxes, and smarter ways of evading them. The foxes in turn are also struggling to survive starvation and react to the rabbit's evolutionary changes by becoming faster and smarter themselves.

The coevolutionary process of generating these tactical traits is an interesting one, and many unusual and innovative traits have been developed. Some of the traits look logical and calculated, such as the mimicry of eyes on butterfly wings that fool predators into thinking they are spotted. The important thing to note however, is that there is no logical thought process involved. The amazing thing about coevolution is that these traits have been generated through random evolutionary variation. They have been tuned by breeding successful variant traits, and excluding unsuccessful ones. This is learning at a species level, not an individual one.

The mechanisms used for coevolutionary algorithms are essentially the same as that

used for traditional EAs. The only real difference is that instead of deciding the fitness of the individuals using a single species/population, the fitness is decided by how well they compete with other species. An outline of a coevolutionary algorithm is shown in figure 2.7, which extends the EA given in figure 2.4. The algorithm outlined shows the case of two species only, however there can exist multiple species.

procedure Coevolutionary Algorithm
begin
  $t \leftarrow 0$
  initialize $P_1(t)$
  initialize $P_2(t)$
  evaluate $P_1(t)$ against $P_2(t)$
  evaluate $P_2(t)$ against $P_1(t)$
  while(**not** termination-condition) **do**
  begin
    select $P_{1\_intermediate}(t)$ from $P_1(t)$
    select $P_{2\_intermediate}(t)$ from $P_2(t)$
    alter $P_{1\_intermediate}(t)$
    alter $P_{2\_intermediate}(t)$
    evaluate $P_{1\_intermediate}(t)$ against $P_2(t) \cup P_{2\_intermediate}(t)$
    evaluate $P_{2\_intermediate}(t)$ against $P_1(t) \cup P_{1\_intermediate}(t)$
    select $P_1(t+1)$ from $P_1(t) \cup P_{1\_intermediate}(t)$
    select $P_2(t+1)$ from $P_2(t) \cup P_{2\_intermediate}(t)$
    $t \leftarrow t+1$
  end
end

Figure 2.7: A generic coevolutionary algorithm

As with the EA, there are a number of variations that exist on the basic algorithm. There can be numerous populations/species, which can change the way individuals are evaluated. Coevolution can also be used to find solutions through cooperation instead of competition, where individuals are evaluated on how well they form a complete solution. For example, De Jong and Potter [40, 83] describe a technique that coevolves a complex structure by decomposing the structure into substructures, and assigning a species for each substructure. The species are then combined through cooperative coevolution to find a complete structure.

As the purpose of coevolution is to compete against the opposition species, the evaluation stage of the algorithm becomes an important factor. With EAs the correct design of the evaluation function is crucial to provide a useful outcome, and it is no different with coevolution. In coevolution on a general level, the fitness of an individual should be de-

termined by how well they do at the task at hand against the opposition population. One of the key factors here is the measurement against the opposition species. The manner of selecting opposition individuals can affect the performance of the evaluation. Rosin and Belew recognized this restraint in [87], and suggested techniques that select from the opposition population using random selection, best of generation selection and competitive fitness sharing selection. Bull [23] also evaluates the use of a roulette wheel style of selection for evaluation. Theoretically, any number of different methods of selection could be used depending on the desired outcome and problem at hand.

The quality of the evaluation function is also affected by the sample size (r) taken from the opposition species. To truly measure the effectiveness of an individual, it should theoretically be played against all opposition individuals to assign a fitness. However, often this is not a realistic task due to processing requirements, and restricted selection from the opposition must be performed. If the selection is too restricted however, the evolution could suffer from an insufficient view of the opposition solution space, and adaptation to the opposition species could be hampered. Ideally, a compromise between processing time and evaluation performance should be found.

Another factor to affect coevolutionary algorithm is that of *cycling* or *forgetting*. As mentioned before, by incorporating more than one species in the evolutionary process, the different species affect the fitness landscapes of their opponents. This works as follows: suppose one population develops a strategy that works against an opposition, that strategy should then spread throughout the population. The only way the other species can compete is to focus on beating that strategy. Thus, the focus of coevolution is to beat the other species' current winning solution, not find an overall optimal solution (if one even exists). In problems where an evaluation function for an optimal solution can be difficult to define, this can be a distinct advantage. However, this process can also lead to stagnation in the adaptive process as the species begin to cycle solutions.

Coevolutionary learning happens at a species level, and the individuals that are not performing competitively are phased out. As a result, individuals that were at one point competitive, but have been countered by an opposing species, are subsequently left behind. At some future stage in the coevolution these individuals may once again become good strategies, and must be 're-invented' by the coevolutionary process. This cycling behaviour in conjunction with the way the fitness landscape of one species chases the other species' form the Red Queen effect [33]. The Red Queen is a character from Lewis Carroll's *Through The Looking Glass*, who was constantly running and never getting anywhere, as the surrounding landscape was keeping up with her.

As mentioned, coevolution is ideal for problems where the optimal can be hard to de-

fine. This concept is a perfect application for the creation of self learned computer players for simultaneous games. Defining an evaluation function to find 'the' optimal player for this is near impossible, as there is no real measurement to use. By utilizing coevolution however, the players are evaluated on how well they learn play, and players with good strategies are allowed to emerge. They are encouraged to find winning strategies by simply playing against another population of individuals that are doing the same thing.

Section 2.4.3 showed us how expert systems could be used to represent strategies for game-play, and this section described the process of using coevolution to create players of the game. The next section gives the background on the representation of these players as fuzzy logic controllers.

2.7 Fuzzy Logic Systems

Fuzzy logic is a means of representing natural language human linguistic variables in a mathematically viable way. The use of fuzzy logic control systems as expert systems has a long standing success rate. When used for an AI player, it also provides human readable rules for strategic game-play. This section describes the basics of fuzzy logic, and the main types of control systems designed to use this logic.

2.7.1 Fuzzy Logic

Zadeh originally developed the concept of fuzzy logic in 1965 [105]. Fuzzy logic formally recognizes the way people naturally categorize information into logically recognizable sets. In classic set theory, there exist crisp boundaries for sets. For example, if a man is 180 cm and over, he is considered tall, while 160 cm – 179 cm could be considered medium. The problem is that this is not how people naturally classify things. People have a concept of a 'grey area', which does not have a decisive crisp value. While we agree that 180 cm is tall, we recognise that 175 cm is still tall to some degree. By using fuzzy sets we are also able to represent the categorization in a non-crisp manner. Instead of saying that anything under 180 cm is not tall, we can instead say that there is a range where a man can be tall *to a degree*.

The 'around 180 cm' is depicted by setting a degree of membership into the fuzzy set *tall*. This degree is defined by a membership function that maps the given input to the fuzzy set with a degree in the range $[0, 1]$. This can be seen in figure 2.8 where there are three fuzzy triangular membership functions represented: short, medium and tall. In the figure, if a man was 180 cm, using the membership function for the fuzzy set *tall* he would be

classified as tall to a degree of 1.0. If he were 170 cm, he would be classified as tall to a degree of approximately 0.5%, and medium to a degree of approximately 0.5%.

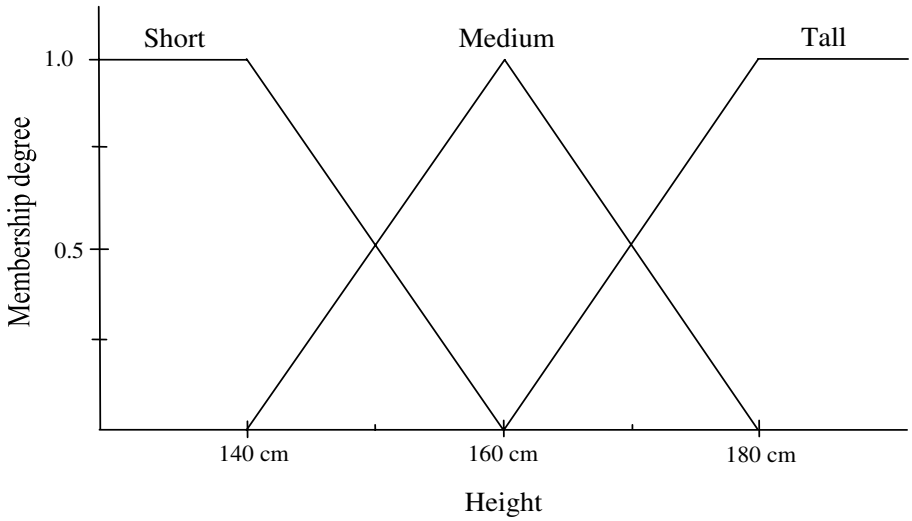


Figure 2.8: Example fuzzy set membership

2.7.2 Fuzzy Control Systems

To create rules using fuzzy logic, there needs to exist one or more conditional *IF* statements with an associated *THEN* statement. The *IF* statement tests the membership degree of an input variable. When triggered, the *THEN* statement maps the result to a resulting fuzzy set. For example:

IF man is tall, *THEN* distance_to_steering_wheel is far

IF man is medium, *THEN* distance_to_steering_wheel is medium

IF man is short, *THEN* distance_to_steering_wheel is close

These rules give two domains of discourse, the *input domain* (U) and the *output domain* (V). In the above example, the input domain is the variable *man*, and the output domain is the variable *distance_to_steering_wheel*. A *fuzzy controller* maps the input set $\vec{u} \subset U$ to the output set $\vec{v} \subset V$. Due to the large application field of fuzzy control systems, there exist many different mechanisms used to perform this mapping. Most of these methods form from some basic fuzzy logic systems, such as the Pure Fuzzy Logic System [103], which uses a Fuzzy Inference Engine to take the fuzzy input sets, quantify them and use fuzzy logic to create the output sets. If crisp (real valued) inputs and outputs are needed instead of fuzzy ones, then an extension to the Pure Fuzzy Logic System proposed by Mamdani

adds a fuzzifier and defuzzifier. The fuzzifier takes the crisp input values and maps them to a fuzzy set. The defuzzifier then takes the fuzzy output set and maps it to a crisp value.

The Takagi and Sugeno system can be also be used to input and output crisp values. This system takes the input variables (as a subset of U) in crisp form, and applies them to the fuzzy rule base. A weighted average of all the rules' output is then given as the final crisp value. This is the system our fuzzy logic control system is based on, and further information on our implementation can be found in section 5.2.1.

As shown, a fuzzy controller uses a set of fuzzy rules to determine behaviour. One method of creating expert systems is to use fuzzy logic to cluster and divide existing data into categories of information. These categories are then used to define fuzzy membership functions for use with fuzzy rules (which future data can be mapped against). A popular manner of obtaining these rules in recent years has been through a process of evolution. For example, by applying an evolutionary algorithm to a set of fuzzy rules and evaluating their performance against some training data. In this way, an expert system can be created without the need to extract knowledge from a human expert.

This concludes the background section of the thesis. We continue with a description of the problem domain in chapter 4, with a review of the current state of research related to ours. Before that however, we introduce you to the game of TEMPO. The next chapter gives a description of the game's background, its elements and the rules of play.

Chapter 3

The game of TEMPO

This chapter discusses the specifics of the TEMPO Military Planning Game, with an overview of the origin and purpose of the game for training US Department of Defence personnel. Also discussed are the possible applications of the concepts in TEMPO as a training and decision support system for other non military applications.

The second part of this chapter then gives a detailed description on how the game is played. This involves listing the rules of play and providing some of the known strategies that human players use.

3.1 Background of TEMPO

The use of strategic thinking is not limited to game playing, and many of the strategies used by players can be carried into real-world situations. This can be seen in a number business and defence organizations, where the organization is essentially competing against rivals for a strategic dominance in their field of expertise. This real-world situation can be directly compared to a zero-sum game of strategy, where there are two or more competitors, and only one can win.

In the area of defence, this game playing can occur when countries engage in espionage and weapon research and manufacturing. The ultimate purpose of this is to maintain sufficient utilities to win a war against rival countries if the need arises, but not spend excessively. The maintenance of only *sufficient* utilities is important, as it implies enough utilities to win a war, without neglecting other budgeting issues. To achieve this fine line, the personnel who perform the resource allocation need to know how to think strategically. The resource allocation is made difficult due to influences such as the political motivations of current (and future) governments, the changing field of the technologies used, and of course the opposing countries with their own changing environments [56].

The US Department of Defence (DoD) realized the difficulty involved in the task, and attempted to give their personnel an advantage through the creation of a management system known as the Planning, Programming, and Budgeting System (PPBS). The PPBS put into place a framework for the decision-making of defence budgeting, and incorporated a set way of planning for current and future objectives [2]. As part of the set-up of this new system, a major training program was initiated to enable the personnel to use the complex new system. The game of TEMPO was created by H. Hatry, F. Jackson and P. Leer of General Electric's TEMPO think tank as part of this initiative [4], and was used by the DoD in the training of their system personnel [56]. The game enabled the personnel to practice the strategies they would use in the creation of the resource allocation, and subsequent yearly DoD budget. Since its creation in the early 1960s, the TEMPO game has been used to teach resource allocation to over 20,000 students.

The original game of TEMPO was a paper-based game where opponents were pitted against each other, with decisions recorded for review by trainers. The efficiency of this was not optimal, as the time taken for game-play with other students limited the functionality. Steps were taken to automate the game with a computer player for the opposition. This allowed the students to play the game on their own time, and results were automatically recorded and available for the trainers. The computer player system provided a greater learning environment for the students, and was successfully used for training DoD personnel. The creation of the player also opened up a new area of research into AI for resource allocation games. The computer player developed is discussed further in section 5.1, along with its problems. The next section describes player objectives for the TEMPO game.

3.2 The Game Objectives

TEMPO is a zero sum game played between two opposing parties by allocating resources in a cold war style simulation. The goal of the game is to acquire more *offensive utilities* than the opposition before war breaks out. The decision-making process requires allocating the yearly budget on the following:

1. Operating existing forces.
2. Acquiring additional forces.
3. Intelligence and counter intelligence.
4. Research and development.

The forces of the game are comprised of weapons that are grouped into four unit types: Offensive A, Offensive B, Defensive A and Defensive B (OA, OB, DA and DB respec-

tively). Each of these units has its own weapons, such as OA1, OA2 and so on. Each weapon has its own attributes (discussed further in the next section), with its own power capability given as *utils*. It is the utils of the weapons that are currently operated for the year that give a player his or her score.

The purchase of intelligence is also provided to give insight into the opponent's tactics. Counter intelligence is used to prevent the opposition gaining this insight. Lastly, investment in research and development is available to provide for future weaponry. The use of research and development in the game allows budget allocation to provide for better weapons in future years. It was excluded in the computer player developed for the DoD however, and was not included in this research. We would like to include it in future research, but analysis on its implementation is required.

The resource allocation involved in the game is conceptually simple; determine what force category is needed and allocate accordingly. The reality is however very different, as the combinations of allocation plans can be high due to the amount of areas to allocate to. This complexity is then magnified by the changing environment that occurs yearly, such as the increase in the chance of war breaking out, and the addition of new weaponry.

The complexity is representative of a number of real-world situations in the corporate and defence world alike, where resource allocation can be a very complicated and difficult task to manage. To understand *how* to make an allocation, a person must have a good understanding of the strategies and mechanisms used in the process. Only through personal achievement and practice can they truly understand the value of various strategies. This is where TEMPO is a great mechanism to practice the techniques needed to develop a well thought out and balanced real-world allocation. The process could also be easily translated into a business training environment instead of a military one.

Now that we have introduced the game, we delve further into its mechanisms. The next section gives a detailed description of the game used in this research, and the rules of play.

3.3 How to Play

The goal of playing TEMPO is to obtain more offensive utilities than your opposition in a cold war scenario when the end of the game is reached, and war breaks out. To do this, a player must allocate a budget on a yearly basis to operate and acquire weapons, and purchase intelligence. Each year the environment changes, and more choices become available. This section gives details on exactly how the game works, and what information a player is given.

Figure 3.1 shows an example year of game-play excluding the intelligence component.

Player's Environment				Previous Year's Data					
		Budget		Player			Enemy		
Year	Pwar	Avail	Left	Type	Offensive	Defensive	Type	Offensive	Defensive
2	12.00%	11530	11530	A	1518	0	A	200	0
				B	200	325	B	0	465

Current Year Allocation:									
Weapon	MaxAcq	AcCost	Inventory	OptCost	Utils	Opted	Bought	ToOpt	ToBuy
OA1	15	75	0	150	120	30	0	0	0
OB1	25	50	0	30	20	30	0	0	0
DA1	25	40	0	20	15	25	25	0	0
OB1	25	100	0	60	50	20	20	0	0
OA2	35	75	0	35	60	0	0	0	0
DA3	25	100	0	50	200	0	0	0	0

Figure 3.1: Example screen of a year's game-play

The player's environment section shows the yearly information necessary to make decisions. This includes the current year of game-play (*year*), the percentage chance of war occurring at the end of the current year (*pwar*) and the given budget for the year (*budget*). The budget is represented as the amount given (available), and the amount left after spending. Each player in the game starts with the same environmental values, but has slightly different values for consecutive years, as each value is increased by a limited random amount. At the end of each year, the average of the *pwar* values for both players (represented in range [0,1]) is compared against a randomly generated number, and if the generated number is less than the *pwar* value, war breaks out and the game is over.

Using the amount given in the *budget*, the player can purchase weapons from the current year's available weapon list. Each year new weapons may become available, possibly with better attributes then previous weapons. The attributes for each weapon are:

1. *MaxAcq* – the maximum acquisition number for the weapon each year.
2. *AcqCost* – the cost to acquire (buy) a single unit of the weapon.
3. *Inventory* – the amount of weapons given to the player in inventory for the year (these are then available for operation).
4. *OptCost* – the cost to operate the weapon for the current year.
5. *Utils* – the power value for the weapon.

Each weapon can be in one of two stages during the game years. These stages are *acquisition* and *operation*. When beginning the game, a player has initial units in their inventory. To obtain additional weapons, they must be acquired. You can acquire up to the MaxAcq number of weapons during a year, and each one bought will cost the indicated amount in AcqCost. The weapons acquired in the current year of game-play, will be available to operate the next year. Operating a weapon activates the weapon for the year. The available

weapons to operate are any weapons in inventory, any weapons bought the previous year, and any previously operated weapons. If a weapon is not operated in the current year it is lost for future use. If a weapon is operated, it is then 'used' for that year, and the utils for the weapon are added to the player's total utils.

At the end of each year, the total weapon utils for each category/type are summed. For example, if a player purchases units of OA1 with total utils of 100, and OA2 with total utils of 300, then the total OA for the year is 400 utils. Offensive weapons of a particular type are countered by defensive weapons of the same type, and vice versa. For example, if Player A has 400 OA utils at the end of a year, and Player B has 100 utils of DA, then the result at the end of the year would be 300 utils of OA left for Player A. Extra defensive utils however, are wasted budget. For example, if Player B had 200 OA utils in the same scenario, and Player A had 300 DA utils, then Player B would have 0 OA utils left, and Player A would have wasted the cost for the extra 100 DA utils. This example is extended and shown in table 3.1 for clarity.

Table 3.1: Example TEMPO net offensive util scoring

Player A		**Player B**	
Type A			
$OA_{playerA}$(OA1 + OA2 + ... OAn)	400	$OA_{playerB}$(OA1 + OA2 + ... OAn)	200
$DA_{playerA}$(DA1 + DA2 + ... DAn)	300	$DA_{playerB}$(DA1 + DA2 + ... DAn)	100
Net Offensive A ($OA_{playerA}$ - $DA_{playerB}$)	300	Net Offensive A ($OA_{playerB}$ - $DA_{playerA}$)	0
Type B			
$OB_{playerA}$(OB1 + OB2 + ... OBnn)	200	$OB_{playerB}$(OB1 + OB2 + ... OBn)	600
$DB_{playerA}$(DB1 + DB2 + ... DBn)	300	$DB_{playerB}$(DB1 + DB2 + ... DAn)	100
Net Offensive B ($OB_{playerA}$ - $DB_{playerB}$)	100	Net Offensive B ($OB_{playerB}$ - $DB_{playerA}$)	300
Total Net Offensive Utils	**400**	**Total Net Offensive Utls**	**300**

If war had broken out in the year represented in table 3.1, Player A would have won the game by 100 utils (Player A total net offensive utils - Player B total net offensive utils). Correspondingly, Player B would have lost by 100 utils. Thus, a player can not win the game simply by maximizing offensive weapons, as the other player can cancel these with defensive ones. Additionally, a sliding scale 'diminishing returns' function is applied when currently operated utils in any one force type (e.g. OA1) produces more than 2000 utils. The adjustments applied are shown in table 3.2, with figure 3.3 depicting the diminishing returns distribution.

The amount of total net utils for OA, OB, DA and DB from the previous year are displayed to the player (on the top right of the screen in figure 3.1). If the player purchases intelligence, then they are also given the opposition results from the previous year (which are displayed to the right of the player's results), although these may be skewed somewhat

Table 3.2: Util adjustments to reflect diminishing returns

Gross utils (GU)	**Adjusted utils**
1 - 2000	Same as Gross utils
2001 - 3000	2000 + .9 $\times$ (GU-2000)
3001 - 4000	2900 + .8 $\times$ (GU-3000)
4001 - 5000	3700 + .7 $\times$ (GU-4000)
5001 - 6000	4400 + .6 $\times$ (GU-5000)
6001 - 7000	5000 + .5 $\times$ (GU-6000)
7001 - 8000	5500 + .4 $\times$ (GU-7000)
8001 - 9000	5900 + .3 $\times$ (GU-8000)
9001 - 10,000	6200 + .2 $\times$ (GU-9000)
10,001 - 11,000	6400 + .1 $\times$ (GU-10,000)
11,000 - ∞	6500

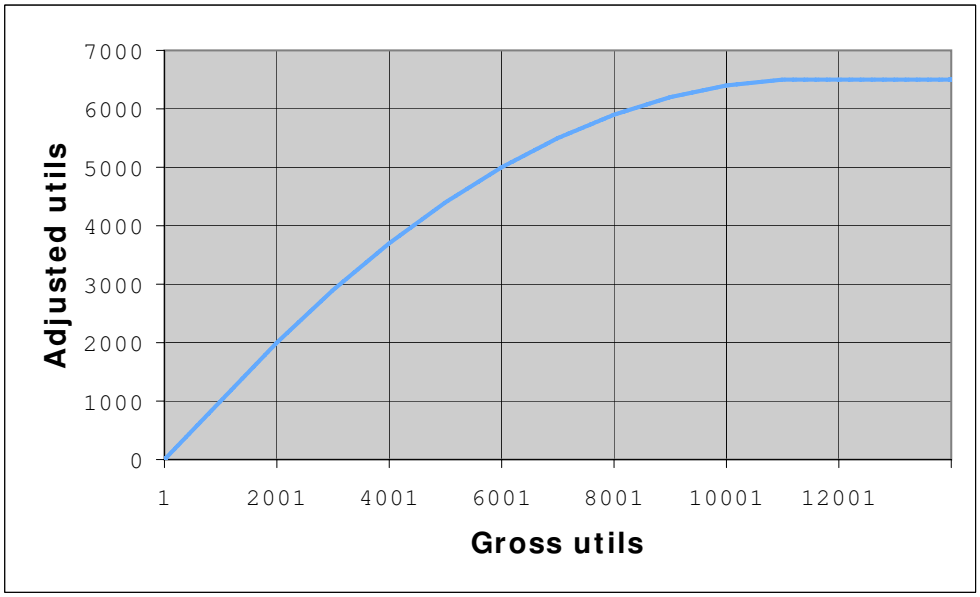


Figure 3.2: Diminishing return distribution for util adjustments

through the opposition purchasing counter intelligence.

Each year of game-play gives an increase in the *pwar* and *budget*, but it also gives new weapons with their own unique attributes. This allows the player to more choice as time goes on, but also increases the complexity of the choices.

The intelligence component of the game involves allocating part of the budget to purchase intelligence into the opposition's allocations. The intelligence is broken into two parts: the intelligence into the opposition's results (*INTEL*), and the counter intelligence used to stop the opposition from seeing your results (*CI*). When a player purchases INTEL, the opposition results for the previous year are given to the player. If CI is bought, the player is only told if opposition utils in a particular category *exist* or not. The original version of TEMPO included a boolean decision for both INTEL and CI. The INTEL was broken into offensive and defensive INTEL, with a set price that you either purchase it at or not. The CI was also a set price with the same boolean choice. For various reasons discussed in Chapter 7, this was then changed and INTEL and CI were broken down into the different types of Offensive and Defensive Intelligence A and B (OIA, OIB, DIA, DIB). The boolean mechanism of purchase was also changed and replaced with a maximum cost for each type. The player could then decide the degree of INTEL/CI to be purchased. The percentage of INTEL/CI purchased then effects the quality of the results obtained back.

There are various common tactics for TEMPO that human players learn through gameplay. After witnessing a number of games being played by humans, there emerge some common strategies that can be beneficial general tactics. These include such things as using the *pwar* variable to determine how you concentrate your allocation, for example if it is low, you might choose to focus your attention on building up your operational Offensive weapons. Another common tactic is to focus your allocations on the weaponry that gives you value for money, that is the weapons that have the highest amount of utils for the cost used to purchase/operate them. There are many other tactics in addition to the ones mentioned, however as yet there is no magical strategy that will win against any opponent.

The dynamic environment of TEMPO, combined with the increasing complexity caused by the number of weapons available, can make the allocation decision process difficult. This is amplified by the uncertainty of what the opponent is doing at the same time. It is only once you have committed your allocation for the year that you can find out what purchases the opposition has made (if you chose to purchase intelligence), and even then the information may be corrupted due to the opposition's purchase of CI. It can be a difficult game for human players to master, and the creation of a computer player is a challenging task.

The next chapter gives a review of the current research in the fields related to this thesis.

Chapter 4

Literature Review

In 1996 Doyle and Dean [42] published a review on the strategic directions for AI. One of the major directions suggested was that of modelling rationality, and two of their long term goals in this area are as follows:

- "Continued development of efficient representations and algorithms for rational decision and action that integrate, extend, and improve on current structured representations for probabilities, preferences, decisions, and (game-theoretic) games.
- Extending the application of theories of rationality to learning and adaptation, especially in situations where the learning process must both use and learn preference and utility information."

These are goals that researchers are still striving for, and characterize the importance of the research discussed in this thesis. This literature review describes some of the current work in this area, specifically relevant to techniques applicable to the TEMPO computer player. We start with a discussion on the addition of memory to the coevolutionary process to improve the creation of long lasting solutions. This is followed with a description of some of the methods being used to create artificially intelligent strategies in general, and how the decision-making process can be implemented with computers.

We first consider the general area of computers and strategy making, and then the specific area of strategy making for games. The focus in this section is on the unsupervised learning process of coevolving computer players for games. We follow on from this to describe research on games that have similar components to the game of TEMPO.

The final topic of discussion for this chapter is that of computer players adapting to human players. One of our main goals is to create a computer player that can help train a human player to become strong at strategy making for resource allocation. This involves

becoming stronger as the human does, and adapting the strategies to the human. The work in this area is presented as the final section to this chapter.

4.1 Memory in Coevolutionary Systems

As discussed in section 2.6, the Red Queen effect can cause the coevolution to 'forget' once good individuals, as the goal of each individual is to beat the opposition for the current generation only. Traits that were once beneficial could be phased out as new strategies emerge in the opposition, only to re-emerge as good strategies later in the coevolutionary process. This constitutes a typical rock-paper-scissors scenario where cyclic strategy creation can occur with a perpetual need to re-learn previously discarded strategies [45].

As discussed by Cliff and Miller [33], a later generation individual should be able to beat an early generation opponent, but this is not always the case. Ancestors of the opposition could have strategies that the later generations have forgotten about and have no way of beating. One solution to this problem is to incorporate some form of memory into the coevolutionary process, one that helps the populations to remember previous solutions [45, 87].

This section discusses some of the memory solutions that have been developed specifically for coevolution. There has also been work on the addition of memory to EAs for use in many domains such as game playing and multi-objective optimization. In such domains, it can be important to retain an external memory structure to store past information. Extensions to the EA exist for the initialization and supplementing of individuals [72], and using the memory to guide the search. There has been a lot of work on this subject, part of which is reviewed by Branke in [22]. Branke defines two forms of memory. The first is *implicit memory*, where the individuals themselves have some form of redundant information forming a long term memory. The second category is *explicit memory*, where there is a storage mechanism used to reintroduce previously learnt information at a later stage of evolution (e.g. replacing weaker individuals with previously good ones, or initializing an EA with individuals from a similar previous EA run). The remainder of this section however, focuses on the specific research of memory in coevolution, as the purpose and outcomes of its use differ from that of EAs.

Probably the most well known memory mechanism for coevolution is Rosin and Belew's hall of fame [87]. Rosin and Belew state two reasons for saving individuals, firstly to continue the genetic contributions of an individual to future generations (through selection and elitism). The second reason is for testing the progress of the coevolution. The hall of fame was introduced for this second purpose, and essentially acted as an extension to elitism by

preserving the best individuals from each generation for continued testing. Experiments were conducted into sampling methods from the hall of fame, including updating the fitness of the saved individuals and selecting based on the updated fitness scores. However, they found that performance issues made random sampling a more beneficial alternative and used this technique for all experiments. Rosin and Belew went on to investigate other mechanisms of selection for opponents in the coevolutionary process in [86], however they only looked at selecting the current best individuals, and did not investigate how to select individuals from the memory.

The use of memory in coevolution to retain previous winning (best of generation) solutions has been investigated using different memory mechanisms. Some involve retaining the best of generations for insertion or replacement in latter generations [22], while others use the memory to compete against and influence the fitness [87]. Puppala et al [84] even used memory as a feature for cooperative coevolution. This involved pairing individuals from the two populations and evaluating on how well they cooperate. The memory then stores any cooperating pairs that have a higher fitness than one of the pairs in memory, replacing the pair with the worst fitness. The individuals from memory are then used for evaluating individuals in the populations.

Whichever mechanism used, there remain the same questions regarding the memory's representation and selection: How should individuals be selected for insertion into the memory? What size should the memory be? How should an individual from the memory be selected for use in the evolution, and how should it be used once selected? All these questions form an integral part of the design of a memory, and research on this is still needed. We start with a discussion on the first issue of selecting individuals for inclusion to the memory.

Most systems choose to use best of generation individuals. However, this is not the only option as discussed by Bader-Natal and Pollack [13], who save the entire populations. Ficici and Pollack [45] also propose the use of a mixed strategy memory to find the Nash equilibrium solutions for a problem. This involves two forms of memory, one that stores the current optimal (mixed Nash equilibrium) strategy (N), and the other (M) stores all old optimal solutions up to a set memory size constraint. The experiments performed with this memory however used evolutionary tactics to evolve the population against the current best individual, and update the memory when new bests were found. The use of the memory in a coevolving system was not tested. The pruning of the M memory structure also means that some useful information could potentially be lost.

Work by de Jong [37] expands on the concept of using the memory to store the currently optimal solution. De Jong defines the coevolutionary process into two species of learner

and tester. The learner (L) is used to find an optimal solution, while the tester (T) is there to assess the performance of the learners. Once again two memory structures are used, one stores the current and past optimal solutions defined by Pareto dominance (the learner archive LS). The other memory stores all solutions that have been dominated by the learners in LS (the tester archive TS), where the current solution in TS must be a superset of all the previous solutions in TS. If a new L is found that can dominate all the solutions in TS, plus some new T not found in TS, then it is considered the new optimal solution. Both the new L optimal solution and the new T associated with it are then added to the LS and TS archives respectively. The individuals in the archives are also used to influence the coevolution through crossover (with a probability of 1.0). This use of memory focuses on finding the Pareto optimal solution for a problem with emphasis on monotonic growth of the solutions.

Having discussed some of the research into *which* individuals to select, the next step is to decide *when* the individuals should be selected and stored [22]. The selection of individuals directly affects the question relating to the size of the population, and the selection of an individual from the memory for use in the evolution. For example, storing the best individual from both populations at every generation can cause the memory to grow rapidly. This in turn causes the probability for selection of individuals to be influenced by the larger scale.

This has been discussed by Bader-Natal and Pollack [13] who describe a number of different ways to add to the memory, such as saving every generation, having a fixed number of generations in memory as a FIFO queue, and periodically adding generations to the memory. The main focus of their research however was to evaluate the use of their 'All of Generation' technique (as discussed before), and a comparison of the different mechanisms was not made.

The question of how to actually use the memory in the coevolution tends to fall into two areas: inserting individuals from memory into the coevolution, or evaluating individuals from the populations against the memory. Most of the current work falls under one of these two categories. Research into what individuals to select from the memory for either of these purposes has however, remained unaddressed. There has been some work on the case-based EA memory, where it was found that individuals that are closer in fitness to the current evolution are the most beneficial [59]. In the area of coevolution however, this remains a fertile ground for investigation. Chapter 6 of this thesis discusses our investigation into some different selection techniques.

The following section describes some of the different ways that researchers have attempted to create computers with the ability to think strategically.

4.2 The Creation of Artificial Intelligence Strategies

The use of AI for strategic decision-making now affects our everyday life; from the directions our cars tell us to go, to the strategies used against us in the computer games we play. To be useful, the strategies must be of a similar level (or higher) to human strategies. The problem here is that humans can intentionally mislead and change strategies at will, which is difficult for a computer to simulate, incorporate, and compete with. Barzel discusses this difficulty in-depth in [16]. One of the main arguments presented by Barzel as a distinction between natural and artificial intelligence, is that computers are not able to deceive, as they lack motivation to do so, given spontaneous circumstances. To deceive someone, is to make them believe what is false is true, or what is true is false [1], usually to the deceiver's advantage. This concept of a computer being able to fool a human has been a key factor in the development of AI.

Experiments into finding computational intelligent mechanisms to fool the opponent have been conducted, such as the bluffing technique by Hurwitz and Marwala [53]. This research uses a hypothesis that bluffing is purely a way of playing the odds, taking into account your opposition's known strategies. This means that if a player is known to 'play it safe' then another player may take advantage of this by continuing to play with a weak hand, and bluff the 'safe' player out of the game.

The game used in [53] is that of Lerpa, which has similar tactics to poker. The computer player used a backpropagation neural network trained against three random strategy players, and then later against other trained networks. By playing the same hand repeatedly, the authors were able to demonstrate that the players did indeed bluff according to their description of bluffing. However, the idea of bluffing often includes more than just playing the odds against an opponent you know well. In fact, a human player who knows how to play a game well will avoid using a single strategy that causes him or her to become a 'known strategy'. A good player will continue to update his or her strategy as the game-play iterates.

To truly read an opponent, even when they are changing their strategy, a player must be able to predict what the opponent is planning. This involves using the *past experience* gained from playing the game (and possibly the same opponent) to predict the opponent's strategy. When correctly predicting the opponent's moves, a player can make an optimized choice of game-play. When a computer performs this task, it too must find some way to learn from past experience (be it their own, or from other sources). There are two main techniques that allow this to happen: supervised and unsupervised learning. Supervised learning allows a human to determine the direction of the computer player's learning, whilst

unsupervised learning lets the computer find its own way to predict the correct action. This thesis focuses on unsupervised learning – supervised learning techniques are not discussed.

The typical training technique for learning is to reward a player when they make a correct choice once the result is known. One alternative of this is to use temporal difference learning (TDL) [99], where the training is done in incremental steps over time. With this method, each step is used to tune the current prediction, while simultaneously updating the previous predictions up to the current time step. TDL has demonstrated great success in unsupervised learning for computer game players, however they require a process that has incremental steps leading to a finish state.

The TEMPO game essentially deals with high level decision-making in the area of resource allocation. The way that high level decisions are made often requires layers of strategic planning, which is discussed by Pinson et al. in [78]. There, the authors present a distributed decision support system using agents to represent the partitioned sub-problems of the decision-making process for strategic planning. They specify that the decision-making process is very complex and involves several levels of decision, which are of an obviously hierarchical nature. By recognising this hierarchal structure, the authors deduced that agent-based systems would be a good way of representing the hierarchy with each of the agents playing the individual roles represented by these levels of decision. These levels are defined as strategic, decision-centre and specialist levels, which follow the idea that decision-making is done by four groups of individuals: top-level managers which define the strategic orientations and decomposition of goals into sub goals, middle-level and low-level managers who deal with the sub-goals, and lastly a group of specialists who define all the environmental constraints.

Our research has some high level similarities with the work by Pinson et. al., however they designed a model specifically for distributed systems, and spend a large amount of time discussing the problems associated with distribution. The hierarchical representation of the decision-making process is worth noting however, as the decision-making process in TEMPO can be described as hierarchical, and perhaps a decomposition into its individual sub-decisions could be beneficial. The research by Pinson et. al. supports the effectiveness of such a decomposition.

As a final note, Spangler published a literature review on strategic decision-making with AI in 1991 [94]. Although mostly outdated, he does make the distinction between AI for decision support systems that represent retrieved expert knowledge, and the need for more cognitive modelling of strategic decision-making. In particular, he presents the field of Competitive Intelligence as an investigative area for the research, as it provides "a highly complex, analytic component of strategic decision-making." The use of strategic

intelligence as a research topic is a major component of this thesis for these exact reasons. The cognitive modelling suggested by Spangler however, is created through observing expert analysts. Our research is more about finding mechanisms to create strategies for the analyst to learn against.

The area of strategic decision-making is very broad, so now we focus on its application to games. The following section gives an overview of the work into developing computer players to perform the decision-making for various games. In particular, the focus is on research where the computer players have been developed using the unsupervised technique of evolving their own strategies for game-play.

4.3 Evolving Computer Players

The 1990s saw a number of champion computer programs developed to beat the best human champion game players. These included the infamous Deep Blue [54] hardware/software combination that defeated the world Chess champion Kasparov in 1997, and the world checkers champion software Chinook by Jonathan Schaeffer [89] that took the world checkers title in 1994. The development of these programs led to debates on the brute force mechanisms used and the direction the AI field was taking. Some researchers then began investigating mechanisms that allowed a computer player to learn strategies of game-play for themselves, rather then being programmed how to play the game by experts and software designers. Leading the research into evolving players that would accomplish this goal, Chellapilla and Fogel [28,29,46] developed their Anaconda program to evolve neural networks to play the game of Checkers. The neural networks were evolved over 840 generations (6 months) [29], and the best evolved network was played online against human checkers players under the name of Blondie24. The computer player achieved an expert level of play against the online players.

The network learned strategies on its own through a process of coevolution, with the only information given being the piece positions, spatial characteristics and differential information. The position of the pieces on the board were represented through a vector of inputs from the set $\{1, K, 0, -K, -1\}$ with 1 representing a normal checker, K a king, 0 an empty square, and negative values as the opposition's pieces. The spatial characteristics of the board were achieved through a hidden layer in the network. The piece differential information (cost of network's pieces against the cost of the opposition's pieces) was given as a direct input to the output layer of the network.

One interesting observation of this work was the reactions of the people playing against Blondie24, as players commented on how surprised they were by the moves made. This

in itself shows one of the key elements of striving for actual artificial intelligence, and not attempting to mimic human intelligence. Something that thinks *differently* to humans would be very beneficial in both a practical and research sense.

Following on from the success of Chellapilla and Fogel, a number of researchers continued to coevolve neural networks to develop computer players for other games. These included Chong et. al. [31] who coevolved a computer player for the game of Othello. Their program comprised a search algorithm based on the basic Minimax search tree algorithm (with a ply-depth of 2) using a neural network based evaluation function that was coevolved. The same input procedure as Chellapilla and Fogel was used where the board positions are received as input and the pieces are represented as +1 for black, -1 for white and 0 for empty. The hidden spatial preprocessing layer is also included in this design, and the piece differential information is sent straight to the output node. The evolutionary process is also very similar to that of Chellapilla and Fogel.

The computer players evolved during the evolutionary process were compared against deterministic computer players. One used only a piece differential evaluation function with Minimax search (no neural network). The other player also used Minimax but included a simplified heuristic for positional play based on Rosenbloom's Othello player's evaluation function [85]. Eventually by generation 892, the evolutionary player was shown to outperform both the deterministic computer players. The computer player developed through this research did not really add much novel information to Chellapilla and Fogel's player except to implement the concept for Othello. However, the testing mechanisms used showed the benefits of using static testing techniques against the coevolutionary process.

Other games were also attempted in the same manner, such as the game of Kalah, which was addressed by Ooh and Lim [76]. Others used the coevolved neural network approach and applied it to the opening play of the game of Go. One example of this is Kendall, Yaakob and Hingston [58]. According to the authors, Go can be modularised into three stages: the opening game, the middle game, and the end game. They state that while the middle game has been analysed for AI techniques, the beginning game, which according to the authors is the most important, had not been addressed. To evaluate their neural network, the network plays against a static player for the first 30 moves, and then the static player takes over from the neural network and plays both players for the remainder of the game.

After the excitement caused by the neural network/coevolution approach, some other hybrid approaches were investigated. One such mechanism included using particle swarm optimization to competitively train neural networks for the game of Tiktacktoe [68]. Another was the work by Johnson et. al. [55], which coevolved a fuzzy logic rule base for the game of TEMPO.

It is also worth mentioning recent research on *imperfect information* games such as poker. Poker, like TEMPO, has opposition information that is hidden from the players. This makes it harder to predict movement. Much recent work has focussed on creating a computer player for poker, such as the opponent modelling work by Billings et al [19, 20], and the evolution of a computer player by Barone and While [14, 15]. Poker has some similarities with TEMPO, and the difficulties in evolving players for poker are similar to the ones for TEMPO in the areas of imperfect information, and deception. There is still much scope for addressing the problems involved in creating computer players for games with imperfect information.

The next section describes some of the research on games that have some fundamental similarities with TEMPO. This includes the need for strategies on resource allocation and intelligence into opposition movements.

4.4 Related Games

Real-time Strategy (RTS) games have a lot of similarities with the game of TEMPO. Both games have a large degree of opposition uncertainty, and resource allocation is at the heart of the decision making. RTS games involve players controlling units for resource gathering, construction, and warfare in a zero sum game. Game-play is performed real-time, with players acting simultaneously.

Davis [36] describes a strategy mechanism for RTS and Turn-based games (where game-play is turn based, with a single player moving each turn). In the system, the strategic AI for the computer player is broken down into three components: the Analysis Module, the Resource Allocation Module, and the High Level AI. The Analysis Module performs the role most similar to that of a TEMPO player – it defines the current strategic goals and then ranks them according to priority. The Resource Allocation Module then uses these goals to allocate the resources in the game environment. The High Level AI then defines a 'personality' for the player by giving it specific rules for a given scenario.

RTS and Turn-based games also gain opposition knowledge through intelligence, usually by allocating a scout resource. To successfully neutralize your enemy you must be able to obtain knowledge of what types of units the opponent has, how many there are and where they are placed. In TEMPO this 'scouting' is done through allocation of budget resources to INTEL categories. Counter intelligence however, as it is used in TEMPO, is not seen in the RTS or Turn-based genres. Resource allocation is also a large part of RTS games [24], and they can offer a much more complex representation of it than TEMPO does. Work in the RTS field has been conducted by a number of researchers. Schadd et. al. [88] have inves-

tigated ways to use data gained from game-play with the opposition to create an opponent model, which can then be used to predict a good strategy to use against them. Miles and Louis [62,63,71] have investigated a number of mechanisms to develop AI for RTS games, including evolving spatial decision making systems, and using case genetic algorithms to 'grow' better human opponents. None of these however has dealt with the high level creation of strategic resource allocation for changing environments. The use of intelligence and counter intelligence also remains unaddressed.

One technique to try and predict your opponent's strategies is to create an opponent model. Schadd et. al. [88] use opponent modelling to create an intelligent AI opposition for real-time strategy (RTS) games. Creation of a model involves collecting the opponent data during the game-play, and classifying it as an available model. Schadd et. al. use a hierarchical approach to divide the model into sub-models, each representing a different, more specialized strategy of play. The hierarchy used has a two level classification, with the top level consisting of a general game style (aggressive or defensive), and the bottom representing the choices of units available for this game style. This is very similar to the way TEMPO works, with its offensive and defensive weapon categories, and may be something that could improve the way the human player model could work.

The importance of using intelligence gained on the opposition to create a strategy needs further investigation however, and the game of TEMPO allows us a great test bed to focus specifically on this task.

First Player Shooter (FPS) games and Role Playing Games (RPGs) are other areas where it is advantageous for a player to obtain knowledge on the strategy of the opposition. As with RTS games, the AI for these genres usually rely on programmed scripts representing a static rule base of strategies. This means that the enemies are of a set level of expertise, and human players can quickly adapt to overcome them. A strong push to have AI that adapts to the computer player can be seen in many games, especially in the FPS genre with games such as Half Life 2 and Halo 3 creating very realistic opponent group combat mechanisms, which react to the battle circumstances. This can also be seen in games like BioShock, where each intelligent agent reacts independently to both the environmental changes and the human player [73].

Pre-programmed scripts comprise the majority of the AI in computer RPGs [79]. The scripts represent a static rule base of strategies. Spronck et. al. [95–97] made dynamic use of scripts, by modifying the choice of script during game-play. They call this adaptive learning technique *online learning* through *dynamic scripting*. The rules in the rule base are manually created and appropriate for the domain. Each AI agent has its own rule base, and each rule has its own importance weighting. Each time an agent is encountered by

the player, a new action script is generated by randomly selecting rules from the rule base, with bias given to rules with larger weights. After the encounter has concluded, reinforced learning is applied by adjusting the weights of the rules employed. Any rule that was used successfully in achieving the goal has its weight increased, and any rule weights that were detrimental to the goal are decreased. The rule base itself contained a set number of rules (depending on the character type; e.g. fighters had 20, wizards 50), and out of these approximately 20-25% of the rules were selected for each encounter. The rules themselves never actually changed, with the adaptation only affecting *which* rules were chosen.

There is a significant need for competent computer players that give human players a challenge. Having a player that can change its strategy depending on how the human is playing is something that is gaining momentum. The next section discusses the current research on computer players that adapt to human game-play.

4.5 Adapting to Humans

While there has been much work on evolving computer players to beat human players, not much has been done on evolving computer players that are designed to be *challenging* for humans. This means creating computer players that are specifically tailored to give the human player a challenge, not just trying to find the optimal way to beat them every time [36, 93].

One area of interest in strategic planning is that of predicting an opponent's strategy and modifying your strategy accordingly. This is a significant area of investigation, which ranges from opponent modelling [44] to human behaviour recognition fields [47]. Carberry performed a review on one of the more important techniques of plan recognition in [26]. The plan recognition technique uses inference to determine an opponent's plan and goal through observed actions. This is achieved through chaining actions to goals reachable through these actions. For example, on observation a person at the store purchases eggs, milk and flour. From this it might be reasonable to infer that they plan to bake something. They then add some Maple Syrup and ice cream to their purchase. It now might be reasonable to infer that they are specifically making pancakes. As humans we do this all the time, but this can be difficult for computers to achieve, especially as the search space of possibilities expands.

The individuals developed using the TEMPO coevolutionary system represent a static rule base that human players can adapt to beat over time. Our long term goal is to create a system where the individuals are evolving and adapting to beat a particular human player during real-time game-play. To find a way to adapt to a human player, we investigated

two aspects: ways to extract rules representing a human's strategy from the output of their game-play, and ways that the system can use these rules to adapt.

The most relevant work is by Louis et. al. with their Case-Injected Genetic Algorithm (CIGAR) research [60–63, 70]. The CIGAR system uses a data base (the case base) of problems mapped to solutions (cases) to prompt the GA to come up with better and more human-like solutions. The basic CIGAR system is a GA that starts with an empty case base and a randomly initialized GA. Once the GA is run, the best individuals (represented as cases) are saved in the case base. When another similar problem comes along, instead of starting with a randomly initialized GA, the case base from the previous problem is used to inject a percentage of the population with previously favourable individuals, and so on. The individuals to select from the case base are chosen by similarity to the current best individual in the GA, using a hamming distance metric. Interestingly, it was noted that individuals that were too advanced for the current population would actually hamper the evolution. The selected individuals then replace the worst individuals in the GA.

The application of CIGAR to games used a strike force real-time strategy (RTS) game, where the computer player has to allocate its resources to a set of aircraft platforms (the blue team). The platforms then attack the human players forces (the red team), which are represented by buildings that the aircraft can target, and defensive installations that can attack the computer players air force. The CIGAR system for this works in the same manner as described above. In this case it also includes cases learnt from humans playing the blue team in previous installations, by storing the human moves in the case base. Thus, the case base consists of human derived cases, and cases discovered by the GA from playing against human and computer players. The cases are then chosen using the same similarity metric, with the possibility of a human case being chosen – and the evolution learning from human game-play.

Our work differs from this in a number of ways. Firstly the TEMPO system uses a co-evolutionary mechanism, not a GA. This changes the way it can use the memory, and adapt to human players. The coevolutionary system can be run without any human interaction, and is constantly changing and adapting. There is no search for an overall optimum, just a way to beat the current opposition. There are also differences in the way individuals are represented. The TEMPO coevolutionary system uses individuals representing a strategy in fuzzy logic. To use human knowledge in the TEMPO system, the human strategy must also be represented as a fuzzy rule base, which can be difficult.

Additionally, the TEMPO game is not a RTS game – it is a turn based game where each player makes his or her decisions simultaneously. Each year of game-play, the environment of the game changes and becomes more complex, and the knowledge of what the opposition

is doing is minimal and can be misleading (if counter intelligence has been used). The players being developed in the TEMPO system are intended to encompass all this and produce generalised strategies for game-play.

Finally there is the way the entire adaptive system works. Our system creates a player that is tailored to an individual human. This means that the human rules have to be obtained from, and added to, the currently evolving system. This differs from the Louis et. al. approach, where the human knowledge was obtained from humans playing past games.

Other relevant work by Ponsen et al [79–81] extends the dynamic scripting method developed by Spronck et al [95–97] for RTS games. As discussed in section 4.4, the dynamic scripting method is used to change the strategy rules (the script) for an opponent during game-play. Rules that perform well in a particular dynamic situation are given a higher weighting, and are then more likely to be selected. The rules themselves are manually designed for the specific game being implemented (similar to most current AI for games).

Ponsen extends the dynamic scripting to RTS by changing the script during successive stages of the game as more resources become available. In addition, an offline evolutionary algorithm was applied that attempted to create scripts to counter well-known optimized tactics (for the game of WARGUS). Static players were used to measure against and the fitness was adjusted for losses and wins against the static player. Aha et. al. [5] addressed some of the disadvantages of using a static player. Aha et. al. used case-base reasoning to select scripts for game-play against random opponents chosen from 8 different opponent scripts. The evolutionary algorithm developed by Ponsen et al. was used to develop scripts for each of the 8 opponent scripts, and the case-based system (CAT) then chose which tactic to use (from possibly different scripts) at each stage of the game.

The research for dynamic scripting has a lot of similarities to the approach presented in this thesis, such as the mechanism used to change opponents during game-play described in Section 6.10. However, once again the research does not use coevolution, and the mechanisms are very different. The overall goal is slightly different as well, as the main goal for the TEMPO system is for training purposes.

This concludes our review on the current field of research. The next chapter describes our coevolutionary system, providing the coevolutionary algorithm used, and the implementation details.

Chapter 5

The TEMPO Coevolutionary System

This chapter presents the methodology used to create a computer player for the game of TEMPO. We begin with the previous computer player developed by Johnson et. al. [55]. We then discuss a number of deficiencies with the previous computer player, which were identified in [55]. Finally, we describe the changes made to the previous system, and give a detailed description of our system.

5.1 The Early TEMPO Computer Player

Johnson et. al. developed a coevolutionary system to create a computer player for the game of TEMPO in [55, 56]. The work by these authors is used as the baseline for the current research. In [55] Johnson et. al. describe the methods they used to create a player for the game. Initially, a Lisp genetic programming technique was used to experiment with creating a computer player for the game. Following the success of the early experiments, Johnson et. al. proceeded to create their TEMPO computer player by coevolving fuzzy logic rules.

The TEMPO game was used in [55] to test the validity of evolving a self-learning artificial player for the game. The system followed similar work by Chellapilla and Fogel [29] where a computer player developed its own method of play through coevolution. Instead of using the neural network approach by Chellapilla and Fogel however, the player was developed by coevolving a set of fuzzy rules. The system used a Mamdani fuzzy logic system with Gaussian membership functions. The coevolution involved two populations of individuals coevolving against each other.

The computer player consisted of two rule bases, one for weapon rules and one for intelligence rules. The rules were represented with Gaussian membership functions for each input value, with the evolution changing the shape of the membership function. The

chromosomes were composed of a number of concatenated rules up to a maximum of m rules, where m is made up of w weapon rules and q intelligence rules. Each rule is then built from the following fields. The first gene for a rule is U_j which depicts if the rule j will be used. This is followed by sets of four genes that repeat for each input. The sets consist of: B_{ij} for defining if the input i is used, C_{ij} to specify the centre of the Gaussian membership function for the input (in range [0,1]), and S_{ij} for the sigma of the Gaussian in range [0,∞]. The final gene for the rule is Y_j, which is the output in range [0,1] for rule j. Figure 5.1 depicts the chromosome structure, expanding rule 3. The rules are then used to decide if a particular intelligence category or weapon should be bought when the yearly budget allocation is performed, using the production operation rule for fuzzy-AND. The gene values are only ever translated into human readable fuzzy parameters (high, low, etc.) for the purposes of understanding what the rules are doing.

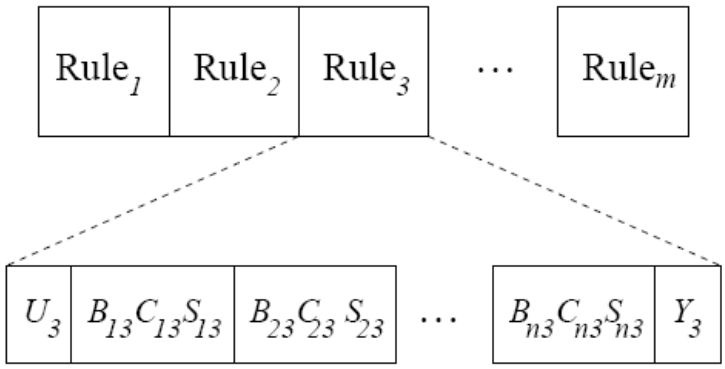


Figure 5.1: The early TEMPO coevolutionary representation

The evaluation function involved each individual in each population playing a set number of games against the opposition population. The outcome of each game was recorded, and at the end of all game-play the individual was evaluated using the average net won utilities. A penalty was also applied to minimize the number of used rules in the rule set – therefore applying the Ockham's razor principle. The variation operators consisted of a two point crossover and randomly created arithmetic mutation with a small probability of a major mutation and a large probability of small mutation.

The computer player developed in [55] was implemented as a limited version of the original game-play, and only goes up to a predetermined number of weapons. Specifically, two weapon categories (offensive and defensive), two weapon types (A and B) and three weapons for each type were used. In addition, the system does not include the research and development component of the game. This part of the game was deliberately left out due to the considerable added complexity that it creates, which was thought to be too much for

the development of the player.

The user interface of the game allowed a human player to play against a coevolved computer player (the computer player was static, chosen from the coevolution).

There was a number of observations made in [55] relating to the ability of the player to compete against humans. Firstly, the rules being developed by the coevolution process were difficult for a novice human to beat initially, but easy to quickly overcome. Secondly, there was the inability for the coevolutionary individuals to develop rules that bought intelligence or counter intelligence. These two topics are addressed in this research.

Further changes to the original system were also made to improve performance. The following sections describe the current coevolutionary system and the fuzzy logic rule base mechanisms used in our work.

5.2 Experimental Settings

Our work developed a coevolutionary algorithm, with fuzzy rule representation for TEMPO computer player individuals. The following sections present the details of the system and the settings that were used. We begin with the representation used for the individuals. We then describe the coevolutionary algorithm used.

5.2.1 The Representation

The original TEMPO fuzzy logic system was based on the Mamdani fuzzy logic system with Gaussian membership functions (see [55] for further details). From observation, we noted that the use of a Gaussian membership function with floating point precision resulted in the evolutionary process fluctuating over small variations that did not really improve the overall results. The fluctuation effectively slowed the system with no gain. To address this, we changed the Gaussian membership function to a triangular membership function with integer precision. It was anticipated that integer precision would stop the minor fluctuations observed with the floating point precision, as the integer representation involved less mutation variance. This meant the system would be able to spend time exploring a larger area of the search space.

The integer membership function was represented by a corresponding integer value for each fuzzy membership function, e.g. 0 for very low, up to 5 for very high. The previous Gaussian method had a floating value centre and sigma for each function, which could be mutated by multiplying a randomly allocated minor or large value. The new mutation mechanism for the integer precision involved randomly assigning a new membership

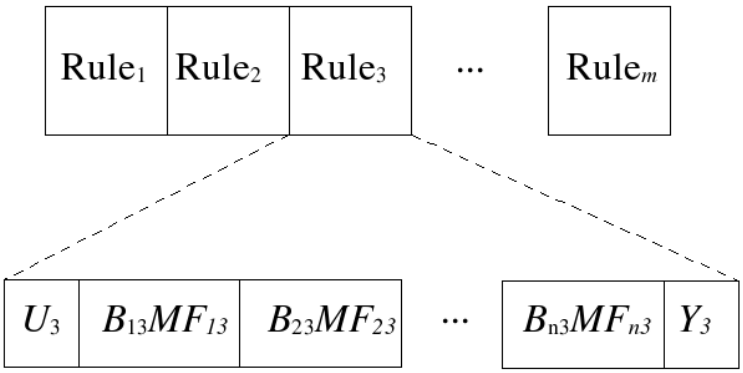


Figure 5.2: New structure of a chromosome

function for a large mutation, or moving to an adjacent membership function for a small mutation. The change was implemented at the start of the experimentation, and the results showed that the processing time was reduced to a third of the original speed. The minor evolutionary fluctuation that previously occurred in the Gaussian system was no longer an issue with the integer approach. The results against the static expert (a simple strategy used as a baseline measurement) were also marginally improved, thus the triangular membership function was an improvement over the Gaussian.

The new representation of the fuzzy logic system meant that the chromosome structure shown in figure 5.1 needed to be modified. The new structure can be seen in figure 5.2. As with the old system, there are $m = w + q$ rules (where w is the maximum number of weapon rules, and q is the maximum number of intelligence rules). Each rule is built from the following (figure 5.2 expands rule 3): U_3 is a Boolean defining if the rule is used, B_{i3} is a boolean defining if input i is used, MF_{i3} is the membership function used for the input i, and Y_3 is the output in range [0,1] for Rule$_3$.

As a final addition, we also added an extra gene onto the intelligence rule base chromosome. This gene represented an evolved cut-off weight used to buy intelligence and counter intelligence. If the fuzzy rules for the intelligence rule base produced a crisp value after passing through the fuzzy controller that was higher than the cut-off, then intelligence was purchased, otherwise it was not purchased. The same applied for the purchase of counter intelligence.

To implement the triangular membership functions, we used a representation of the Takagi and Sugeno fuzzy system. We created a single triangular membership function for an input dimension, corresponding to one of the linguistic variables used. Each input type has its own set of membership functions. The membership functions either represent the different types of input (e.g. category is either Offensive – 0, or Defensive – 1), or a range

measurement (very low – 0, low – 1, medium – 2, high – 3, very high – 4).

We also needed to represent the linguistic variables. Given the total minimum and maximum of a particular input, and the number of membership functions needed, the linguistic variable would create the required number of membership function objects for the input type. Each membership function has a minimum, centre and maximum value. A diagram showing a typical membership function is shown in figure 5.3. The membership function is represented as an isosceles triangle with a y value of 1.0. The bottom edge is of length $maximum - minimum$, and the centre value is used to divide the triangle into the left and right right-angle triangles. When passed an input value ($u \, \epsilon \, U$), the slope-intercept equation: $y = mu + b$ is used to calculate the membership degree. In our implementation $m = 1/(centre - minmax)$ where $minmax$ is either minimum or maximum depending on which half of the triangle u is in. We then use $b = -minmax/(centre - minmax)$.

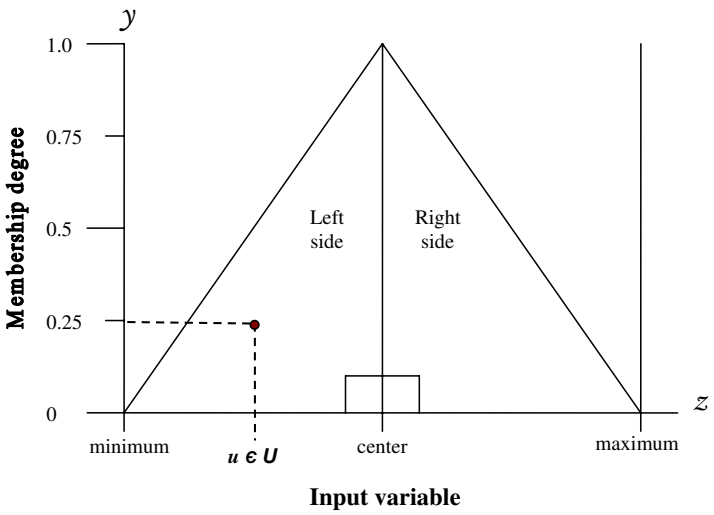


Figure 5.3: Membership function example

The fuzzy controller takes each input value for the year (e.g. budget, weapon category etc.), and matches the value against the fuzzy *IF* part of the rules, getting the membership degree for each input that triggered a fuzzy rule. The fuzzy *AND* product rule was used to sum all the membership values for all the used inputs in the rule. The weighted average of all the rules was then taken as

$$y(\vec{u}) = \frac{\sum_{l=1}^{m} w^l y^l}{\sum_{l=1}^{m} w^l} \, , \tag{5.1}$$

where $\vec{u}$ is the input vector for the rule base, m is the number of rules in the rule base, y^l is the crisp output value of rule l, and w^l is the product of the membership degrees for all

triggered input values. We define

$$w^l = \prod_{i=1}^{n} \mu\, var_i^l\, u_i \ , \tag{5.2}$$

where n is the number of inputs, and $\mu\, var_i^l\, u_i$ is the membership degree (μ) of input u_i in the corresponding linguistic variable var_i, for the rule l.

The above process is run for each of the different weapons available, with some common input values for all weapons (such as pwar, budget etc.), and some weapon specific values (such as category, type, utils etc.). The total $y(\vec{u})$ value obtained from all the triggered rules for each weapon with its specific $\vec{u}$ vector is then used to allocate a percentage of the budget (the $ratio$) for the specified weapon. The allocation is performed by normalising all the weapon $y(\vec{u})$ values with

$$\textit{ratio}_i = \frac{y(\vec{u})_i}{\sum_{j=1}^{m} y(\vec{u})_j} \ ,$$

where i is the weapon index currently being allocated a ratio, and m is the number of weapons available for purchase at the time.

The intelligence rule base goes through the same process, with one difference. As the purchase of intelligence is a boolean decision, the cutoff ratio given for each rule (as described previously) is used to determine if the intelligence category (offensive, defensive or counter intelligence) is purchased. If the $y(\vec{u})$ value for the given category is greater than the cutoff ratio, then the category is purchased, otherwise it is not. As a result, the purchase of intelligence occurs first. The remaining budget is then used to purchase weapons, with each weapon given the allocated amount (*ratio*$_i \times$*budget*). Any surplus left over is then added on to the next weapon's budget allocation.

The next section describes the coevolution algorithm used, and our implementation details.

5.2.2 The Coevolutionary Algorithm Implemented

Our coevolutionary system is based on two competing populations to evolve individuals represented by the fuzzy logic rule bases. The outline of our algorithm follows the one described in section 2.6, with some differences. Our algorithm is shown in figure 5.4.

To initialize the populations, each individual creates a gene array for the weapon and intelligence rule bases, and randomly assigns integer values from the range $[0..99]$. These values are then mapped from genotype to the appropriate phenotype, depending on the al-

procedure The Coevolutionary Algorithm
begin
  $t \leftarrow 0$
  initialize $P_1(t)$
  initialize $P_2(t)$
  while(not termination-condition) **do**
  begin
    evaluate $P_1(t)$ against $P_2(t)$
    evaluate $P_2(t)$ against $P_1(t)$
    select $P_{1\_elites}(t)$ from $P_1(t)$
    select $P_{2\_elites}(t)$ from $P_2(t)$
    select $P_{1\_intermediate}(t)$ from $P_1(t)$
    select $P_{2\_intermediate}(t)$ from $P_2(t)$
    alter $P_{1\_intermediate}(t)$
    alter $P_{2\_intermediate}(t)$
    select $P_1(t+1)$ from $P_{1\_elites}(t) \cup P_{1\_intermediate}(t)$
    select $P_2(t+1)$ from $P_{2\_elites}(t) \cup P_{2\_intermediate}(t)$
    $t \leftarrow t+1$
  end
end

Figure 5.4: The coevolutionary algorithm implemented

lele requirements, as part of the evaluation phase. For example, if the gene locus requires a phenotype to a linguistic variable, then the minimum phenotype value is the first membership function number, and the maximum is the total number of membership functions for the variable (e.g. 0 – 4).

After the initialization, the individuals are evaluated against the other population. The basic system applies the following evaluation technique (with modifications introduced later). We iterate over each population, with each individual played against r randomly chosen individuals from the opposition population. The random selection mechanism was carried over from the research by Johnson et. al., and it is possible that a different mechanism might be more effective. For the purposes of this research however it is sufficient, and further experiments on the matter are outside the scope of our work. For most of our experiments we used a sample size of $r = 20$. Through experimentation we found that this size gave us enough sample scope for the population size of 100 that was used in the experiments.

A single game-play involves a complete game, through to the final year when war breaks out and a total net utils is allocated for the game. In each game the two players distribute their budget as determined by the rule base (discussed in section 2.6). At the

end of the game, the total net utils for each player is determined and the fitness evaluation variables updated accordingly. These variables keep track of the number of games played by the individual, the total net utils for all games played, and the won and loss count for the games (a draw counts as a loss for this purpose). Each individual plays a minimum n games, but with a possible (but improbable) maximum $n + (O_p n)$ times, where O_p is the opposition population number. The maximum is due to the random selection for gameplay by the opposition's evaluation round. The evaluation function consists of a number of evaluation variables, defined as:

- *wonRatio* – the total number of wins divided by the total games played.
- *totalNetUtils* – the sum of all total net utils (both positive and negative) for all games played.
- *gamesPlayed* – the total number of games played whilst evaluating the individual.
- *weapRulePenalty* and *intelRulePenalty* – the constant parameters used to determine the weight of the rule penalty for the rule bases (weapon and intelligence respectively).
- *usedWeaponRuleNum* and *usedIntelRuleNum* – the number of rules that were marked as used for each rule base.
- *usedWeapInputNum* and *usedIntelInputNum* – the number of input variables used for each used rule, in each rule base.
- *weapInputNum* and *intelInputNum* – the total possible usable inputs for each rule base.
- *netUtilsPenalty* – assigned the highest net utils scored from all games played.
- *lossCounter* – counts all games lost or drawn from the games played.

The evaluation function *eval* is then calculated as follows:

$$
\begin{aligned}
\mathit{eval}(\mathit{individual}_i) \; = \; & \mathit{wonRatio} + (10e^{-6}\,((\mathit{totalNetUtils}/\mathit{gamesPlayed}) - \\
& \mathit{weapRulePenalty}\,(\mathit{usedWeapRuleNum} + \mathit{usedWeapInputNum}/\mathit{weapInputNum}) - \\
& \mathit{intelRulePenalty}\,(\mathit{usedIntelRuleNum} + \mathit{usedIntelInputNum}/\mathit{intelInputNum}) + \\
& 10e^{-6}\,(\mathit{netUtilsPenalty} \times \mathit{lossCounter}\;/\;\mathit{gamesPlayed})))
\end{aligned}
$$

Once all the individuals have been evaluated, the populations are sorted by the individual's fitness score (which maximises the evaluation function). After sorting, the elite individuals from each population are collected as $P_{1\_elites}(t)$ and $P_{2\_elites}(t)$ respectively. The number of elites saved is determined by the evolutionary parameter *elitismRatio*, which was set to 10% for most experiments. The intermediate populations $P_{1\_intermediate}(t)$

and $P_{2\_intermediate}(t)$ are then selected from $P_1(t)$ and $P_2(t)$ through tournament selection ($k = 2$). The $P_{intermediate}(t)$ populations are of size $Pn - (Pn \times \textit{elitismRatio})$, where Pn is the size of the corresponding $P(t)$ population. The tournament selection was carried over from the Johnson et. al. solution. We conducted experiments with rank selection, but found that tournament selection worked better.

The intermediate populations are then altered, with either mutation or crossover applied. The evolutionary parameter *xoverRatio* defines the chance of crossover occurring instead of mutation. If the mutation operator is used, the chosen parent has mutation applied at a rate defined in the evolutionary parameter *mutationRatio*. If mutation is applied to a gene, there is a 10% chance of a big mutation being applied where the gene is randomly reassigned a value. Otherwise a small mutation of plus or minus 1 is applied (with boundary checking put in place). If the crossover operator is chosen, a randomly selected two point crossover is applied. The variation operators were carried over from the Johnson et. al. research, but the size of the individuals make at least two point crossover necessary. Possible future work could investigate k-point crossover.

Once the alterations have been applied to the intermediate populations, the survivor populations ($P_1(t+1)$ and $P_2(t+1)$) are created. These are created as $P_1(t+1) = P_{1\_elites}(t) \cup P_{1\_intermediate}(t)$ and $P_2(t+1) = P_{2\_elites}(t) \cup P_{2\_intermediate}(t)$.

The parameters of the coevolution are modified in subsequent chapters, with changes described in the relevant section.

Now that we have described the basic algorithm used, the next chapter describes the first stage of experiments. To create a computer player that can self-learn its game-play, we give it the use of short and long term memory to enhance the learning experience.

Chapter 6

Short and Long Term Memory in Coevolution

6.1 Introduction

A major goal in evolving a computer player for TEMPO is to create a means of competitive strategy creation. Coevolution has proved to be a very effective way of creating computer players that perform self learning to find appropriate strategies for a problem. This has been shown repeatedly in experiments to create computer players for a variety of games as discussed in section 4.3.

The previous research performed by Johnson et. al. [55] on a coevolutionary system for the TEMPO game identified deficiencies that led to creation of weaker players. Not only were the strategies relatively easy to beat, they also acted blindly to the opposition and did not purchase intelligence. One possible reason for these deficiencies was that the coevolutionary process being used was not providing progressive growth of the individuals, and was cycling solutions. While we are not attempting to create an optimal solution through the creation of computer players, we do need a mechanism that will encourage some degree of consistent improvement. To provide this, the populations need some form of memory to prompt them to learn from the past and allow future development.

The purpose of this memory however, is not to enforce monotonic improvement. We want to encourage the populations to grow, but we also want to leave the coevolution free to find interesting and unique strategies. The goal of the system is to be challenging for humans, which is its ultimate purpose. To create this balance, we needed a memory system that would retain and use the memory, but only as a formative guide for the coevolution.

We begin with a description of the experimental settings used. This is followed by our

first extension, which includes of an 'expert' seed into the populations. We then discuss the structure of the memory that was chosen, and the initial experiments using this memory. This leads into our major contribution for this chapter, the introduction of a selection mechanism for the memory – that of short and long term memory retrieval. We describe the implementation techniques used, the experiments, and the results. We conclude with a discussion on the impact the memory made to the TEMPO coevolutionary system.

6.2 Experimental Settings

All experiments performed in this section have the same experimental settings, except when stated otherwise. Due to the stochastic nature of the experiments, we conducted each experiment over ten separate runs, all with the same environmental and evolutionary configuration. Due to the time and hardware restraints ten runs was consistently used for all experiments. However, we did experimentally run more than ten times, and there was not a distinctly noticeable difference. Less than ten runs did however have a greater impact.

The system was run each time for 50,000 generations with a population size of 100 for both populations (population A and population B). Each experiment also had the following parameters: the *elitismRatio* in each generation was 10%; crossover was applied with 30% probability of occurrence; if crossover was not chosen, mutation was applied (thus with a 70% probability of occurrence); if applied, each gene had a 50% chance of mutation, with a 10% chance of large mutation – otherwise a small mutation occurred. The number of evaluation games played changes throughout the experiments, and is detailed in the relevant sections. The rule penalty was set to a constant of 10. The intelligence rule base however assigned a proportional rule penalty according to the number of inputs (e.g. $intelRulePenalty = rulePenalty \times totalIntelInputNum/totalWeaponInputNum$).

The existing baseline measurement of the system used a static rule base 'expert' that represented the strategy of: buy weapons based on their utilities per operation cost. The higher the ratio, the more of the weapons would be bought. The term expert is used loosely here, as the strategy is a simple concept. However, the strategy has proved capable of winning against novice human players, and can be considered a good baseline player for measurement of system performance. It has also been noted through our user study (in chapter 8) that this strategy is one that human players naturally learn to use as a beginning strategy. At the end of each generation, the best individual from each population would play 100 games against the static expert, and the percentage of games won formed the *won ratio* used in results.

The results for each experiment are depicted in graphs showing the average results

across the ten runs. The graphs show the won ratio against the expert on the y axis, and the generations on the x axis. Additionally, the results are also shown with the Standard Deviation (SD) range. The SD results have been smoothed using a Bezier curves function to accentuate the changes.

6.3 Seeding the Populations

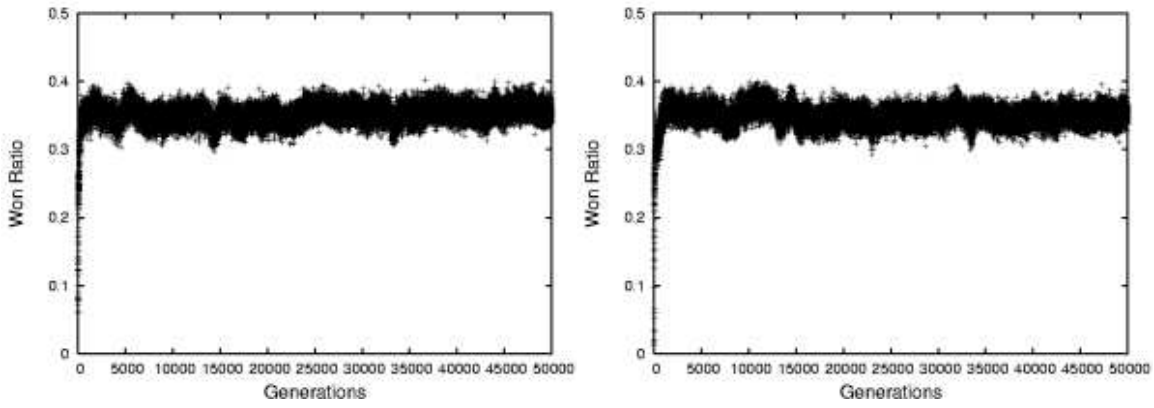
One suggestion made in [55] was to seed the population with hand crafted individuals to determine how human-like individuals would fare in the coevolutionary process. We investigated this area, and provided a simple experiment to determine the reaction of the coevolutionary process. When using the expert for performance measurement, it was noted that the best individuals from the generations performed relatively poorly on average against the expert. This was to be expected to some degree, in that the populations were focused on beating each other, and they had no incentive to find strategies to beat the expert. We decided to insert the static expert as an individual into each of the populations and record how it fared in the evolutionary process. To differentiate between the two functions of the same expert, we named the baseline measurement use of the expert as the *static expert*, and the seeded expert the *alien expert*.

The alien expert mechanism worked by placing the static expert as another individual in each of the populations. The individual would be subject to the same variation operators as all the other individuals. The baseline static expert measurement system remained the same.

6.3.1 Experimenting with the Alien Expert

As a baseline measurement, we ran the system without any seeding mechanisms. The evaluation function was calculated solely from game-play against the opposition, with the number of games played against the opposition given by $r = 20$. The results are shown in figure 6.1 . For these graphs, the scale is to 50% for readability. The results depict how well the best individual from each generation performed against the static expert player, which was the baseline measurement.

The results show that the players were not doing overly well against the expert, achieving an average of 36% won ratio. The results also showed that there were no trends to beating the expert, just occasional jumps in performance followed by decreases (this is not visible in the average results shown here, but is visible in individual runs). This was to be expected, as the coevolutionary process was not training the individuals against the expert,



(a) Population A (b) Population B

Figure 6.1: Baseline success ratio against the static expert

only against the other population.

To try and encourage the system to include the expert in the process, we used the alien expert approach discussed previously. Here, the expert was inserted as one of the individuals for each population A and B. The results from this approach were much the same as the original results. After the addition of the alien expert, there was a very brief increase in the won ratio followed by a sharp drop back to the previous average won ratio. Once again, this was mostly expected, as the nature of coevolutionary systems is to focus on beating the opposition in its current form, and promptly forgets any previous opponents once they have been beaten.

This led us to experiment on the inclusion of a memory into the coevolutionary system. We now consider this extension.

6.4 The Inclusion of a Simple Memory Structure

As discussed by Ficici in [45], the coevolutionary process can forget previously learned solutions in favour of ones that are more effective against the current opposition. This can be detrimental to the overall usefulness of the system. The issue of forgetting previously good solutions is well documented in coevolutionary research, and while analysing the results from experiments we found it was leading to the creation of strategies that were not highly competitive. By introducing memory we attempted to increase the competitiveness of our computer players to play against all strategies, not just the current opposition.

As discussed in section 4.1 there are a number of different ways we could implement and use memory in our system. Our first approach was to implement a separate memory for each population, and each generation the best of the opposition was stored in the other

population's memory. This was intended to remind one population of the other population's winning strategies, and vice versa. After further experimentation and analysis however, we came to the conclusion that to create a truly competitive individual, it should be able to beat the opposition and any previous strategies of its own. We decided to experiment with having a single combined (and decidedly larger) memory that both populations would augment and play against.

We used the memory in the coevolution as opposition for the current populations. Thus, at the end of each generation, each individual would play a defined number of games against the opposition population, followed by a number of games against individuals in the memory. Hence, the fitness was maintained in the same manner as before (the won ratio of the individual plus the total net utilities divided by the times played, with a penalty for number of rules used). The coevolutionary algorithm used was now modified as shown in figure 6.2.

procedure The Covolutionary Algorithm with Memory
begin
  $t \leftarrow 0$
  initialize $P_1(t)$
  initialize $P_2(t)$
  while(not termination-condition) **do**
  begin
    evaluate $P_1(t)$ against $P_2(t) \cup Memory$
    evaluate $P_2(t)$ against $P_1(t) \cup Memory$
    add $P_1(t)\_fittestIndividual$ from $P_1(t)$ to $Memory$
    add $P_2(t)\_fittestIndividual$ from $P_2(t)$ to $Memory$
    select $P_{1\_elites}(t)$ from $P_1(t)$
    select $P_{2\_elites}(t)$ from $P_2(t)$
    select $P_{1\_intermediate}(t)$ from $P_1(t)$
    select $P_{2\_intermediate}(t)$ from $P_2(t)$
    alter $P_{1\_intermediate}(t)$
    alter $P_{2\_intermediate}(t)$
    select $P_1(t+1)$ from $P_{1\_elites}(t) \cup P_{1\_intermediate}(t)$
    select $P_2(t+1)$ from $P_{2\_elites}(t) \cup P_{2\_intermediate}(t)$
    $t \leftarrow t+1$
  end
end

Figure 6.2: The coevolutionary algorithm with memory implemented

At the evaluation phase, individuals are also evaluated against the memory population. The fitness is calculated against totals of all games played. Each individual currently played

against random individuals from the opposition r times. In our memory strategy, the individual is then played an additional r_2 times against individuals from the memory. So the total games played to assign the fitness for the individual is now $r_1 + r_2$, where r_1 represents the games played against the opposition population, and r_2 the games against the memory.

The populations are sorted after evaluation (as before), but the fittest individual from each population is now added to the memory population. The rest of the algorithm performs as before. We decided to include every generation's best individual from each population into the memory, as we wanted to keep a full history of the past winning strategies used. Initially we implemented the memory as only containing unique entries, and later experimented with the inclusion of all individuals regardless of uniqueness.

After deciding on the structure and influence mechanism of the memory, one question remained: how would the individuals be selected from the memory. This question is a significant issue for this chapter. We hypothesize that the individuals selected from the memory can have a large influence on the effectiveness of the coevolution.

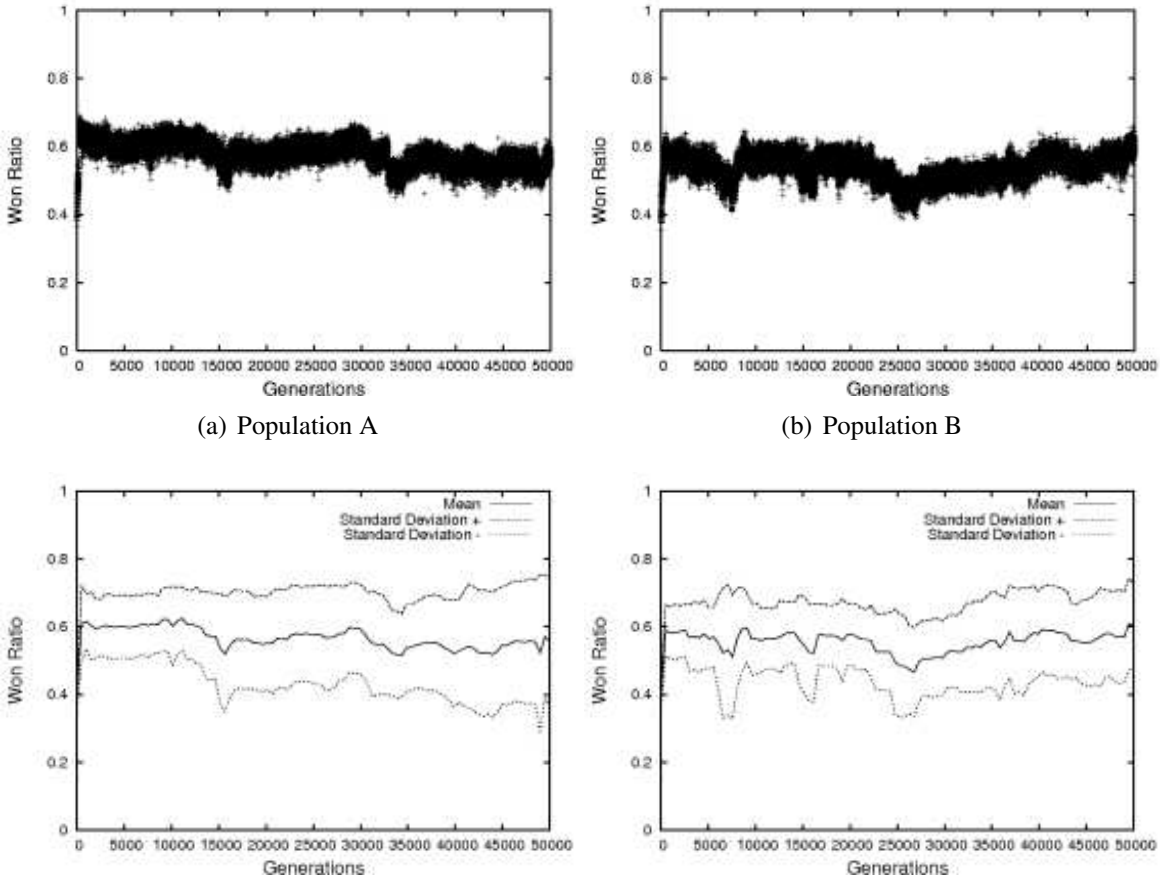
Our initial experiments with this system involved implementing the memory structure as described and randomly selecting individuals to play against. To evaluate the fitness, a number of games were played against the opposition ($r_1 = 20$), re-adjusting the evaluation after each game. To include this strategy of evolving against the memory, we then provided a mechanism for playing an additional number of games ($r_2 = 20$) against the memory, and adjusted the fitness of the current individuals in the same manner as that for the opposition games. The additional sample size of $r_2 = 20$ was chosen to match the opposition sample size, and minimize the impact of the memory on the performance.

The graph of the won ratio against the expert can be seen in figure 6.3. In these results (as with the rest of the results) the y axis represents the won ratio against the static expert to a scale of 100%.

The use of a memory gave us distinctly better results. We were now achieving an average of around 55% won ratio, and were achieving an over 90% top won ratio against the expert for single run results, with longer trends of winning.

These results were promising, but we noted that the populations were still not tending towards beating the static expert and were instead regularly staying around the average to below average won ratio. The populations were still 'forgetting' previous solutions and focusing more on beating the current opposition. We reasoned that the cause of this was the random nature of the selection from the memory.

By selecting at random, we were increasing the potential for larger amounts of the earlier, and presumably simpler, individuals from the memory to be chosen. What we actually wanted was the populations to improve against the harder (more recent) individuals, and



(a) Population A

(b) Population B

(c) SD for Population A

(d) SD for Population B

Figure 6.3: Success ratio against the expert using random selection from history

be occasionally prompted not to forget older strategies. Thus addressing the Red Queen effect, without losing the benefits of the coevolution. To achieve this, we decided to include a probability distribution for game-play selection from the memory.

The initial probability distribution was a simple linear time distribution, where the latest historical individuals had a higher probability of getting chosen than the older individuals. This is depicted in figure 6.4, where the distribution can be seen as a function of time $P(t)$, where N is the current generation number.

This distribution would force a higher playing ratio against the more recent individuals, but would still encourage play against the older strategies. To implement this probability distribution, the individuals in the memory were represented in a list, with the most recent additions added to the top of the list. When an individual from the memory is needed for game-play, a random number is generated in the range [0..N] representing the individual to play against (n). The probability for n viz. $P(n)$ is then calculated using the linear slope

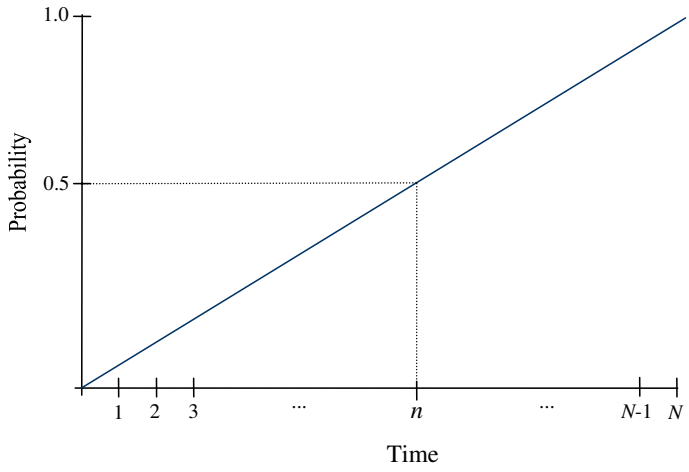


Figure 6.4: Probability distribution by function of time

function,

$$P(n) = 1/Nn \tag{6.1}$$

derived from the slope intersect function ($y = mx + b$, where $b = 0$).

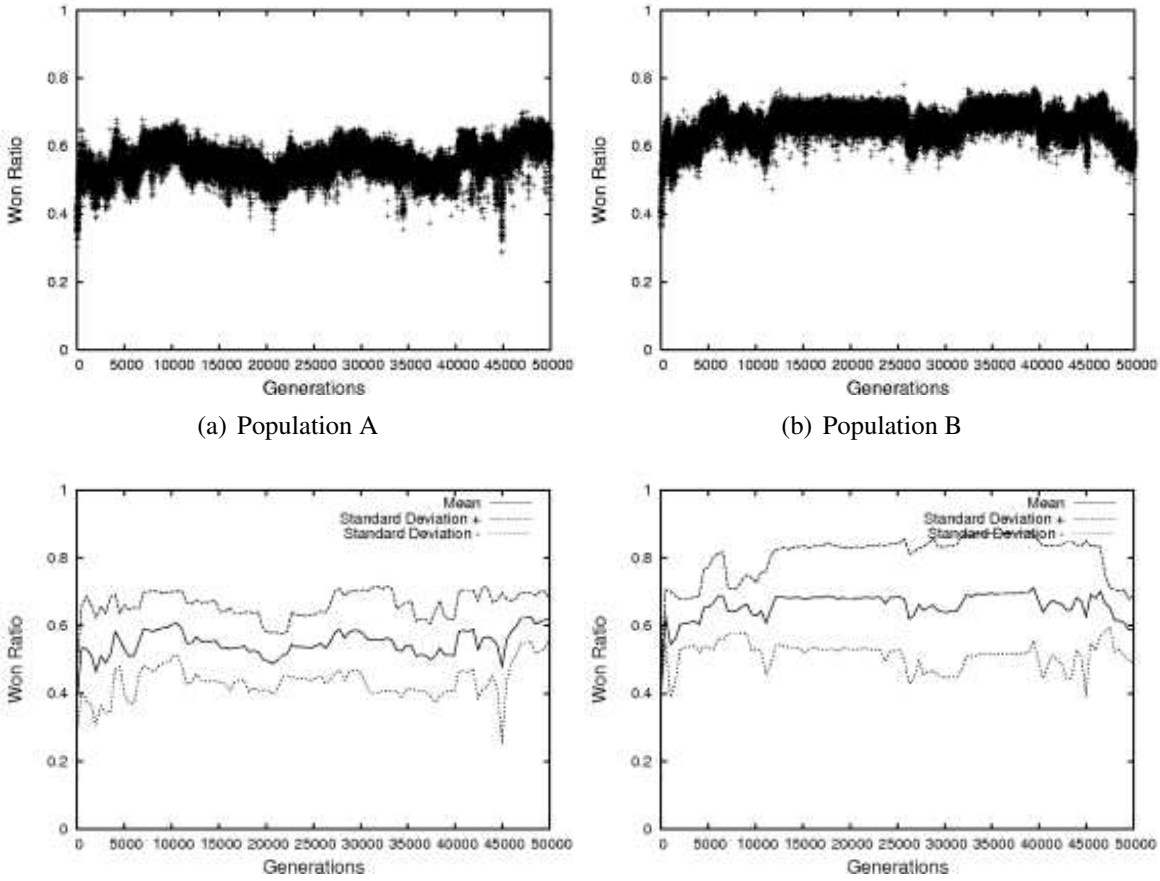
Once $P(n)$ has been found, another random number is generated in the range [0,1]. If the number generated is less that $P(n)$, then the individual is selected. Otherwise the process iterates.

The results of this experiment can be seen in figure 6.5, and showed an overall increase in the won ratio average against the expert. This was a good result, but we wanted to see if it could be improved.

The results from a typical single run of the linear probability distribution can be seen in figure 6.6. This shows the fluctuation that occurred in a single run, and the room for improvement against the expert.

The next question prompted by the previous results was whether the linear probability distribution was the best distribution. Perhaps a curved distribution could provide better results. To test this theory, we replaced the linear distribution with a Laplace (otherwise known as double exponential) probability distribution function. We were expecting that by introducing a curved distribution function with a sharp decline, we would force the populations to play more against the most recent players in the memory. The results of this experiment are shown in figure 6.7.

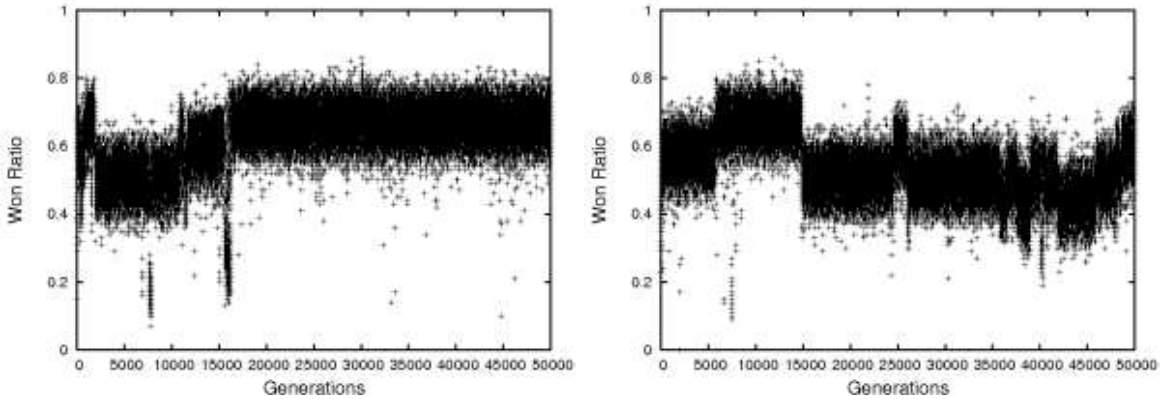
The individual results once again showed a slight improvement with a more pronounced trend towards winning against the expert. However, there was still a lot of fluctuation and the results were still not achieving the consistent won ratio we were looking for. What we



(a) Population A (b) Population B

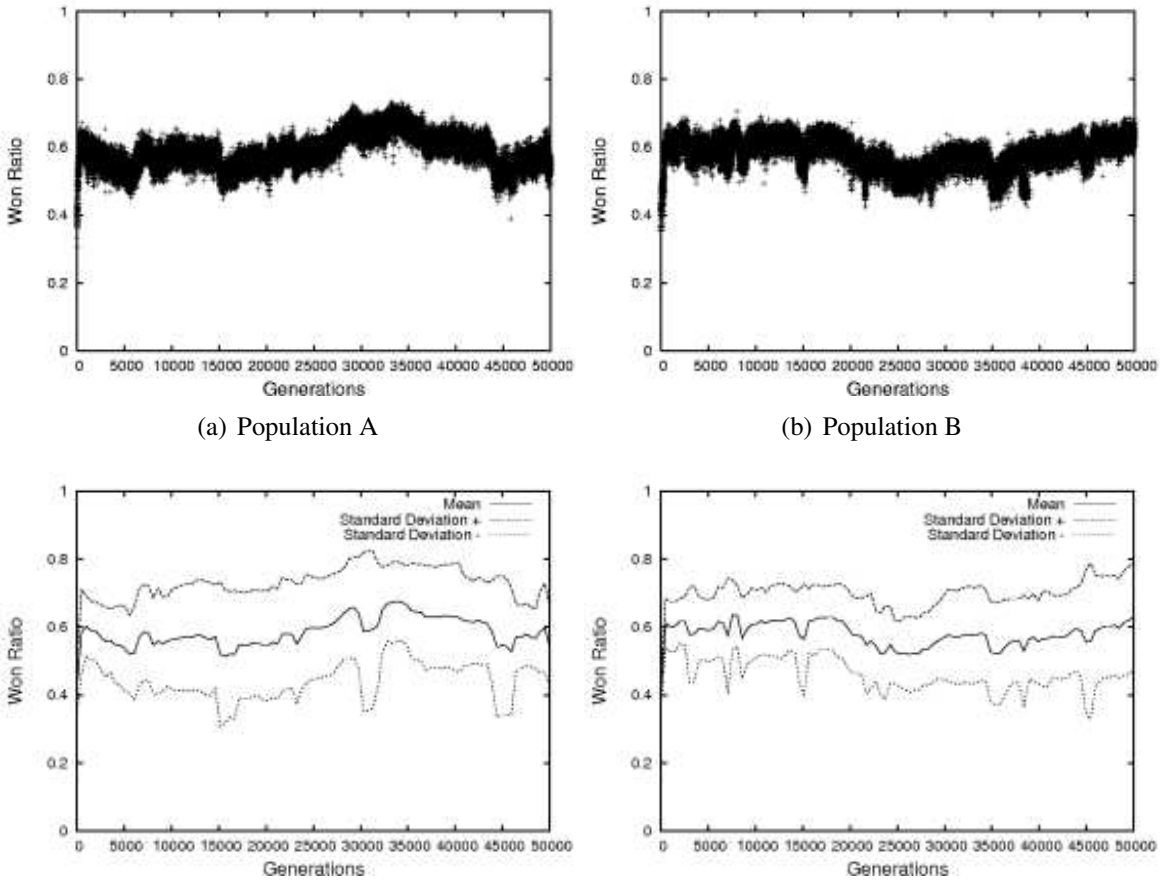
(c) SD for Population A (d) SD for Population B

Figure 6.5: Success ratio against the expert using single memory with linear distribution



(a) Population A (b) Population B

Figure 6.6: Success ratio against the expert for a single run using linear distribution



(a) Population A

(b) Population B

(c) SD for Population A

(d) SD for Population B

Figure 6.7: Success ratio against the expert using Laplace probability distribution

really wanted, was to see an overall increase over time in the won ratio against the expert.

To really achieve the degree of selection pressure on the more recent memory additions, we decided to experiment with the inclusion of a distribution mechanism that was inspired by the way humans use memory; that of short and long term memory. The following section gives a background to how humans use short and long term memory.

6.5 A Discussion on the Psychology of Human Memory

In our investigation of ways to select individuals from the memory for use by the coevolutionary system, we considered the way that humans utilize memory. In our experimentation it was decided to investigate the use of two types of memory selection: *long term memory* (LTM) and *short term memory* (STM). The approach was to simplistically mimic the way the human brain stores and uses memory. The concept of applying STM and LTM to heuristics has been widely used over the years, and the use with tabu search has been thoroughly

documented by Glover and Laguna [48]. In our research however, we seek to mimic the human mechanism of STM and LTM in a coevolutionary process.

Early theories of human memory consisted of one large memory system. However, in the 1960s researchers recognized that the memory consisted of two parts: a short term memory that acted as a temporary storage mechanism, and a long term memory that was more permanent storage [6]. This was re-examined in the 70s and the two memory systems broken down into further levels of processing [12]. The STM was re-termed the working memory, and broken down into articulatory and visuospatial components. The LTM was broken into *explicit* and *implicit* memory. Implicit memory involves learning that does not involve active recollection of information, but rather retrieval through indirect performance. Implicit memory includes information associated with: skills and habits, priming, conditioning and non-associative learning [12]. Explicit memory on the other hand is the active recollection of past incidents, and the semantic memory representing general knowledge of the world. Explicit memory includes storage and retrieval of facts and events.

The actual linking between short and long term memory was originally thought to happen serially; that the information being taken in by our sensors was processed in the STM and then if it stayed in the STM long enough, it would be transferred into LTM [12, 35]. Craik and Lockhart [34] challenged this theory, by introducing the concept of a levels-of-processing framework. This framework had a sequence of analytic stages that show how memory is set. The idea is that memory is not a separate faculty, but instead reflects the outcome of attempts to perceive and comprehend information. Thus, the ability to affect long term memory is directly related to the comprehension or ability to relate the knowledge with the meaning (semantic understanding). In this way, the retrieval of information is not a 'brute force' search mechanism, but instead the memories are encoded into the whole cognitive system, and there is an increased potential for the same pattern to be repeated on a subsequent occasion [35].

The understanding of how the STM and the LTM work, both independently and together, is still under extensive investigation. However, the knowledge that the memory is divided into two forms: a short term memory aimed at recalling recent and relevant information, and a long term memory to store necessary information in the long term, are fundamental theories for human memory.

6.6 Implementation of a Short and Long Term Memory Structure

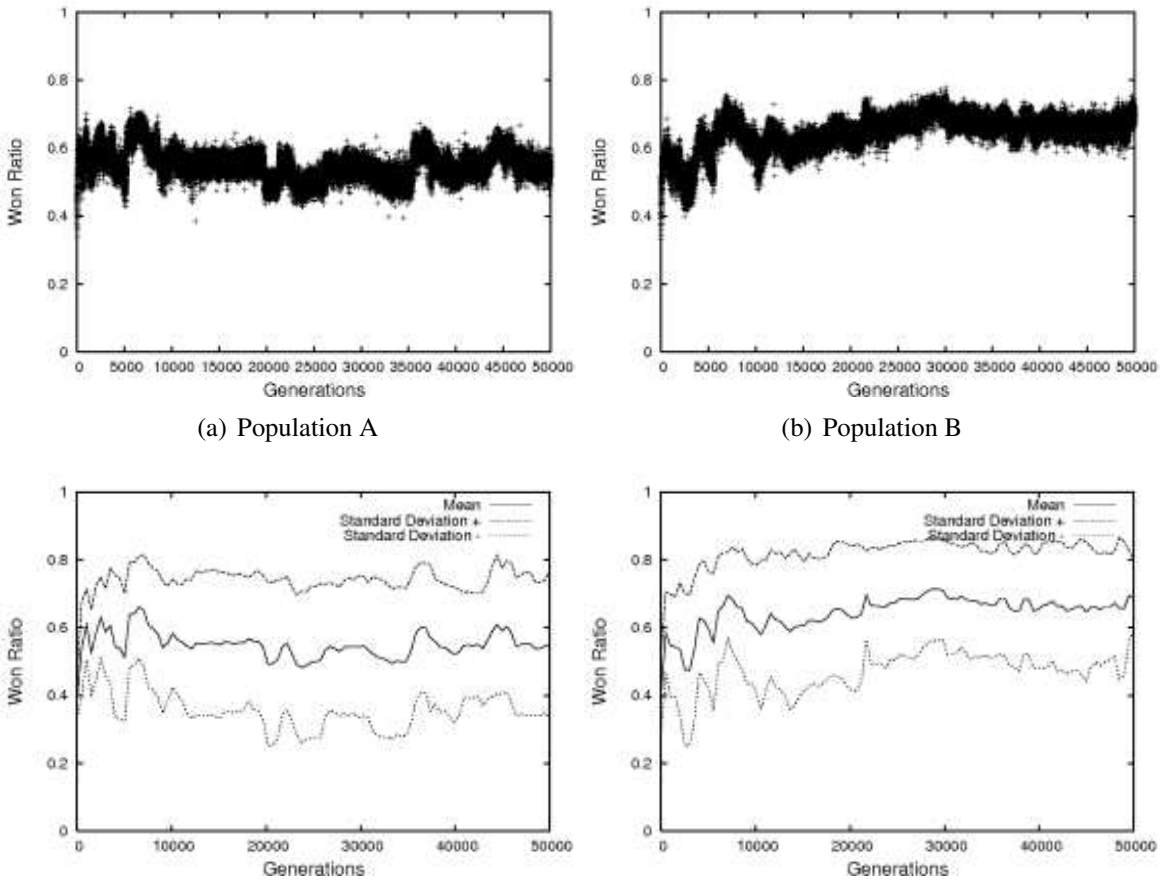
Our memory system is a continually growing population, with an increase of two individuals every generation (when uniqueness is not applied). This means that whatever selection mechanism we apply, it is still effectively a time based probability distribution. The more generations that pass, the larger the population becomes, and the less probability an individual has of being selected for game-play. We have investigated different mechanisms to bias this selection probability, but the selection probability of all these still diminish over time.

The inclusion of a memory to evolve against had produced some nice peaks, but even with the probability selection distribution it still seemed to average around a 60% won ratio. The time function probability selection we were utilizing allowed a biased selection mechanism, but it was still allowing a higher play rate against the older solutions than we wanted. Even though the time-based probability selection was biased towards the most recent individuals, as the memory grew, the chances of playing against the top end decreased. To address this we decided to introduce a specified top end window to the probability selection – the short term memory window. This window would be evolved against a set number of times prior to evolving against the whole memory – the long term memory.

As discussed before, humans have a current short term memory, and a larger long term memory. We wanted to replicate this ability, as it is the short term memory that humans use to determine the current situation, and the long term memory to bring past knowledge on how to act given the situation. This is precisely what we wanted the players to do. By forcing the populations to play against the most recent individuals in the STM, we are maintaining the concept that the most recent situation is the most relevant. By then allowing influence from the LTM, we allow learning from the past.

To implement the STM function, the top ten individuals in the memory were identified, and an additional amount of games played against them. The LTM was then applied by playing another set of games against the entire memory. So where previously the fitness was created by playing $r_1 + r_2$ games, now the games against the memory is split into r_{2S} for the games played against the STM, and r_{2L} for the games against the LTM. The memory individuals selected for the LTM game-play are selected with the linear time based probability discussed earlier, so the more recent individuals have a higher chance of being selected. The STM is also selected from with the same linear time based probability, but as the size is a static 10 individuals, as the entire population grows, the probability becomes more uniform.

Our experiments for the new memory mechanism included a window of the top ten individuals (the STM), and played against these individuals for $r_{2S} = 10$ games. This was followed by $r_{2L} = 10$ games against the LTM (the entire memory). The games played against the opposition remained the same ($r_1 = 20$). The results were promising, with an increase in won ratio for both populations as depicted in figure 6.8. More importantly, we were now seeing trends towards beating the expert. This can be seen in figure 6.9, which shows the results for a typical single run of the system. The single runs for the system showed that in most cases, one of the populations was outperforming the other population, and so the overall results were therefore improved as well.



(a) Population A

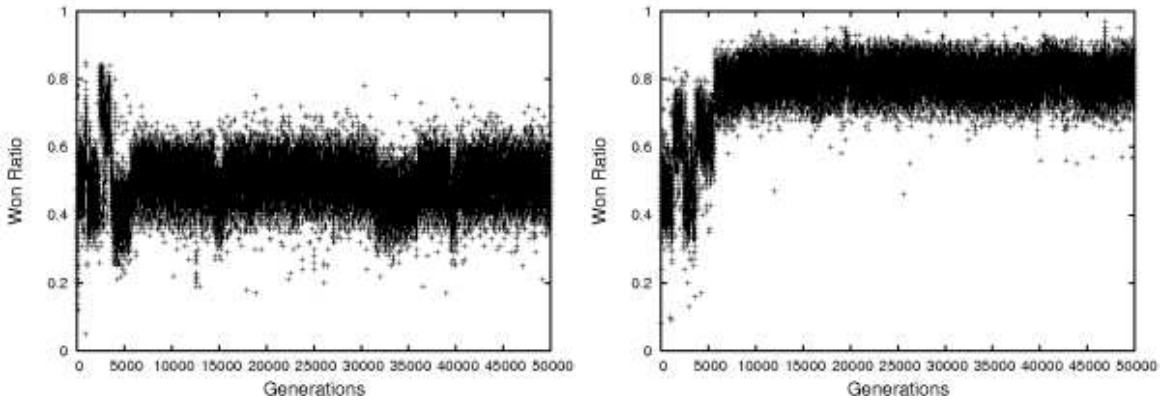
(b) Population B

(c) SD for Population A

(d) SD for Population B

Figure 6.8: Success ratio against the expert using short and long term memory

The short term memory mechanism works like the human memory mechanism for learning, which allows a person to overcome information overload and use the most useful and recent memory rather than their large long term knowledge base. The LTM is still there for prompting historical memories and reminding the person of past successes and mistakes, and allows the person to grow overall. We have attempted to replicate this in the



(a) Population A (b) Population B

Figure 6.9: Success ratio against the expert for a single run using short and long term memory

TEMPO system, allowing the system to have reminders of past strategies, while still giving the best information for the current situation. Additionally, we experimented with the size of the STM, to see if a window of 10 was the most beneficial. We tried smaller and larger windows, and found that a size of around 10 created the best results.

Following on from the success gained using the STM and LTM memory structure, we decided to continue experimenting with different ways to change the selection pressure. The following sections describe each of the variations using the STM and LTM.

6.7 Unique vs. Not Unique Memory Structure

One of the choices made early on was to make the individuals in the memory unique, and then increase the probability of getting chosen if the same individual was repeatedly added to the population. We decided to test this decision and performed a run with no uniqueness checking, instead best individuals for all the generations were added to the memory.

Although there was not much of a difference, this was actually the first time the populations achieved an outright 100% won ratio in the single runs. We were not expecting this, as we thought the unique population would have fared better due to the forced diversity of the populations. This higher achievement could possibly be caused by the short term memory filling up with the best strategy, and therefore forcing the populations to find a solution to that single strategy. Further experiments are needed to confirm this hypothesis however, and it is an area for future research.

6.8 Ranked "Gladiator" Selection

The next experiment was designed to further mimic the way the long term memory worked in the human brain. Instead of every single bit of information ever learnt being stored in long term memory, the information would be sifted and only relevant information stored. We attempted to mimic this by using a ranked LTM. When adding a new individual to the memory, the individual would play the top individual from the ranking. If it won it ranked above the top individual, otherwise it would continue down the ranking until it found the appropriate rank. This was only performed for the top 1,000 ranked individuals. If the individual was not in this top 1,000, it would not be retained in the LTM. We nicknamed the 1,000 top ranked individuals the "gladiators".

This mechanism provided a very loose ranking, as ranked individuals are only played against once to determine placement. It is therefore likely that the ranking would be incorrect in some cases. To address this, we included a mechanism where at each generation a random individual would be chosen from the gladiators and played against the surrounding 20 neighbours, adjusting in rank where required.

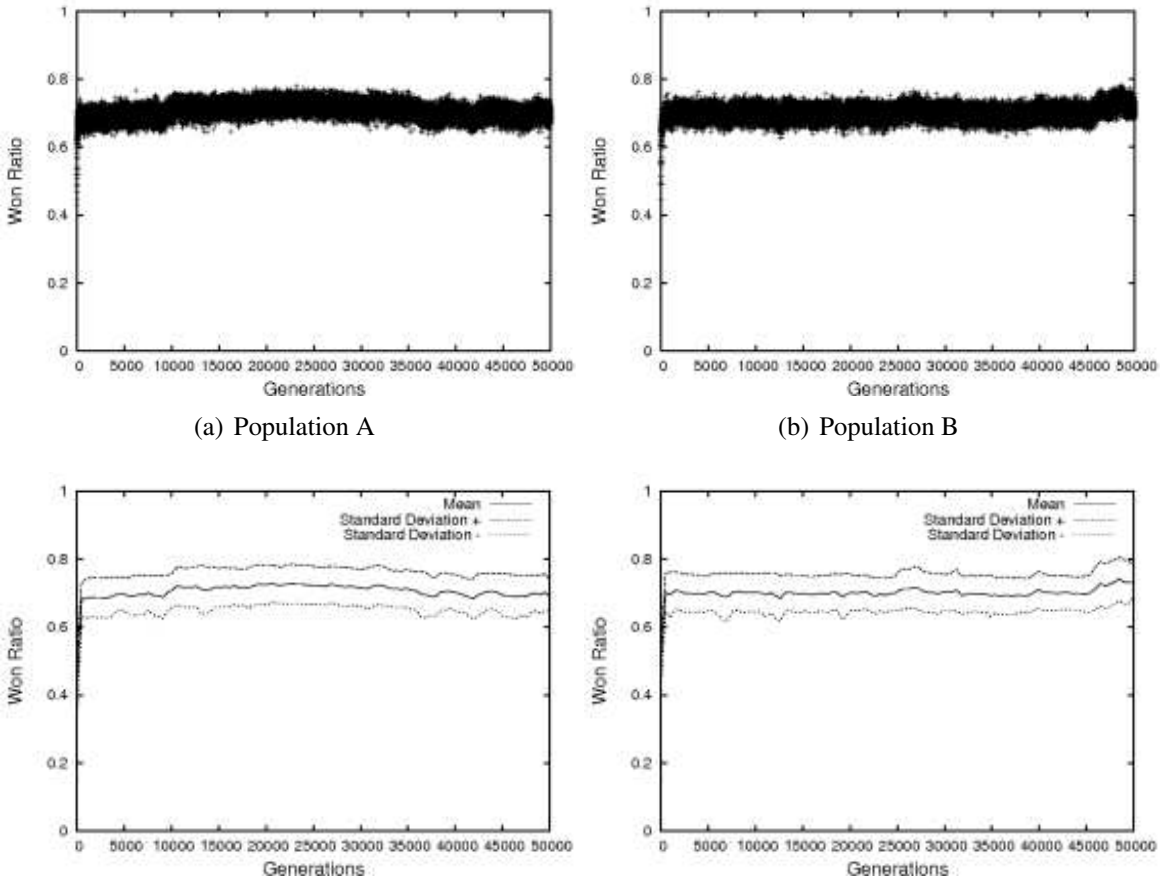
The fitness calculation process was then modified as follows. After playing $r_1 = 20$ games against the opposition, the individual would be required to play against the short term memory as before ($r_{2S} = 10$). They would then be required to play an additional number of games against the gladiators ($r_{2G} = 10$). The results can be seen in figure 6.10.

We were now seeing very promising results, with a much higher and obviously stable won ratio. The single results for the gladiator system showed that in most cases both of the populations were averaging an above 60% won ratio. However, we were still witnessing the stabilized advantage of one population over the other as noted in the previous experiment.

The gladiator mechanism proved successful, but future work could investigate different ways to rank the individuals in the gladiator system. Currently the mechanism is a very simple stochastic one, and possible future research could consider other alternatives.

6.9 Investigating the Inclusion of Migration

From the results of the short and long term memory and gladiator experiments, we noted a distinct pattern of one population dominating over the other. This domination was obvious (especially in the case of the short and long term memory) and happened almost every time the system was run. The interesting thing was the dominating population very quickly achieved and maintained a very high won ratio against the expert, while the opposition player stayed at an average won ratio. We theorized that the games against the memory



(a) Population A (b) Population B

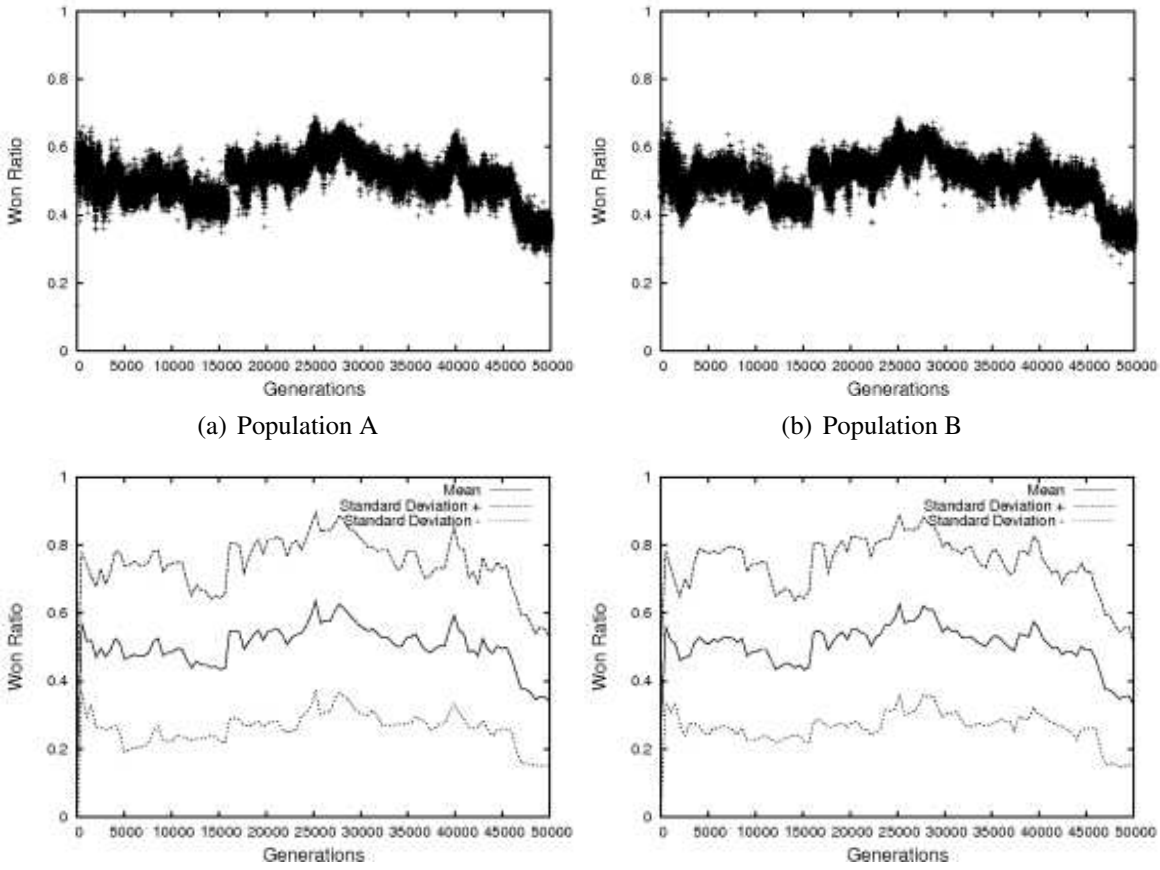
(c) SD for Population A (d) SD for Population B

Figure 6.10: Success ratio against the expert using gladiator system

were effectively rewarding one population over the other by adding extra play against the best individual from the 'better' population. This meant that the first population to find a strategy that was good against both its own solutions and the opposition's, would then be able to continuously dominate against the opposition. The question we had however was why the opposition was not able to catch up. Our ideal scenario involved both populations being competitive with the expert strategy, so we decided to investigate the direct influence of integrating individuals from the winning population into the opposition population, and vice versa. To do this we incorporated methods of population migration. While migration is a non-standard method of coevolving solutions, we felt it was worthwhile experimenting with in this scenario.

For these experiments we included a migration mechanism that involved a probability of performing crossover with an opposition individual. This was performed the same as the normal crossover (two point crossover), except that one parent was chosen randomly from the current population, and the other parent was chosen randomly from the opposition

population. The probability of crossover being performed was 30%, and if crossover was chosen there was then a 10% chance that migration crossover was applied. The experiment was then run using the gladiator selection. The results from these experiments can be seen in figure 6.11.



(a) Population A

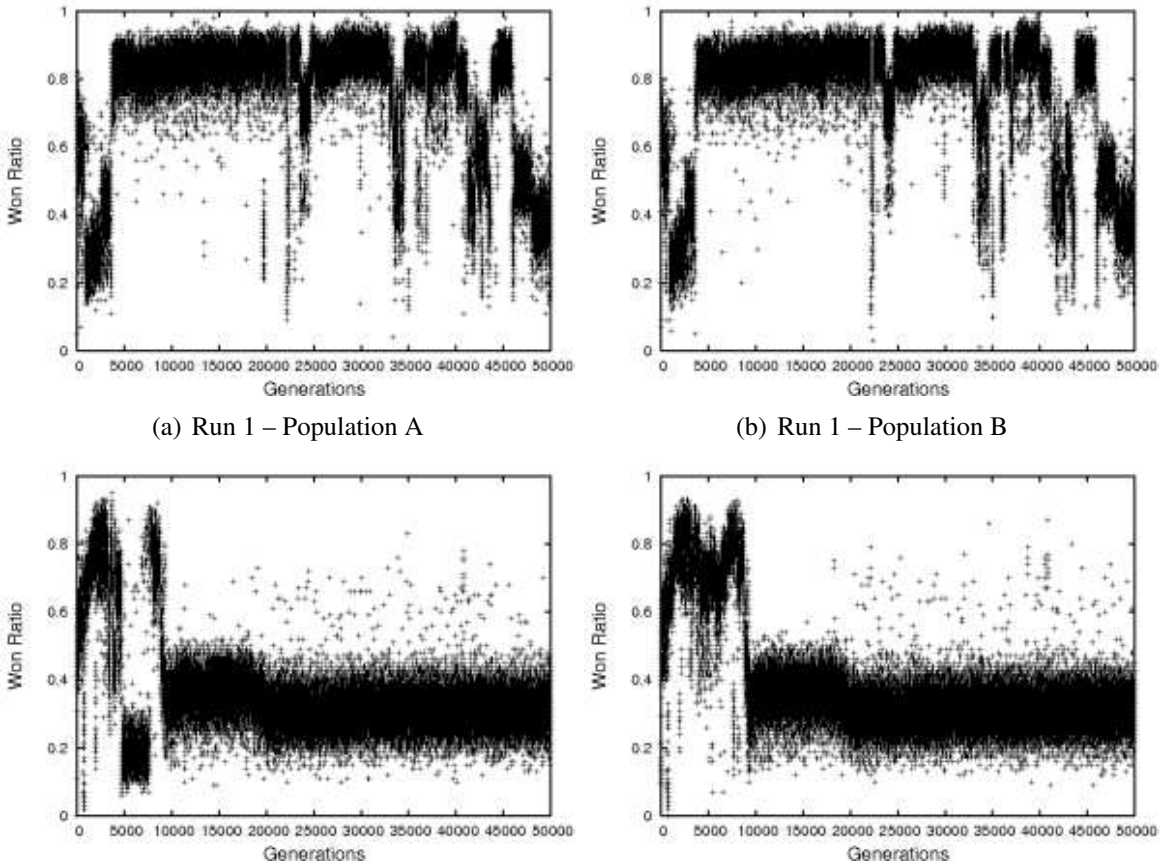
(b) Population B

(c) SD for Population A

(d) SD for Population B

Figure 6.11: Success ratio against the expert with migration incorporated into the gladiator system

These results show a very big change from the gladiator experiments without migration. The results are clearer when the single runs are viewed, and figure 6.12 shows some of the single run results for this experiment. Even with minimal crossover being applied, the results showed that the populations quickly started mirroring each other. The results also seemed to disprove our theory of the memory having the biggest influence in the creation of a dominating population, as the dominance disappeared with the small probability of migration applied. These results were very surprising, and further research into why this has occurred is warranted, but outside the scope of this research.



(a) Run 1 – Population A

(b) Run 1 – Population B

(c) Run 2 – Population A

(d) Run 2 – Population B

Figure 6.12: Success ratio against the expert for two single runs with migration incorporated into the gladiator system

6.10 Creating a Dynamic Enemy

The ability of the human memory to select relevant information implies that a person can make a more meaningful decision given the circumstances at the time. This ability is also something that is applicable to the TEMPO game, as the environment represented in the game is a dynamic one, where each game has a randomized generation of environmental variables. The game is made even more dynamic during the game-play, as each turn of play the environmental variables change and players are given new resources.

By creating a memory retrieval function that allows the memory to be selected based on the current circumstances in the game environment, we hypothesize that the players developed will represent strategies that incorporate the environmental change, not just the strategy of the opposition. To do this we introduce the concept of a dynamic rule base as a coevolutionary opponent. This involves changing the current enemy used for fitness evaluation, as the environment itself changes. The traditional way of coevolving against an

opposition requires taking two individuals, playing them against each other, and evaluating the fitness from the outcome of the game. With a dynamic enemy we are playing a single individual against a number of different individuals that change as the game itself changes. By changing the enemy according to the current environmental scenario, we are creating a dynamic enemy that changes its own rules accordingly. This is a more realistic scenario to the way humans play games, and allows better adaptation to the dynamics of real-world problems.

When a human recalls an event from long term memory, it is normally by relevance to the current situation. This implies that there is some mechanism the brain uses to trigger the appropriate memories. Mimicking this technique is difficult in our system, as the only thing that changes according to the 'current situation' is the environment of each game. To create a mechanism where the memory used is relevant to the current situation, there actually needed to be a current situation. As mentioned before, the evaluation function plays a set number of games against the opposition population and then more games against the memory structures, with adjustments made to the fitness at the *end* of each game. Using this technique there is no way to evaluate against the current situation *during* a single game.

To create a situation where the environmental changes could be used to trigger a relevant rule from the memory, the game playing mechanism itself was re-evaluated. Instead of playing all the years of the game against a single opposition and evaluating the outcome, we could instead use different players whose rules were pertinent to the current situation in the game and evaluate on the total outcome. This would create a dynamic enemy that changes as the environment changes. This is an entirely new way of coevolving the TEMPO players.

Additionally, as discussed in [57] it is common in coevolution to see groups or clusters of individuals form that are focused on solving a sub-problem of the end goal. This is particularly useful in our case as the different parts of the game can be broken up into sub areas of strategy that deal with changes in *pwar*, budget, offensive, defensive and intelligence. By including a clustered evaluation with the dynamic enemy, we hope to encourage this clustering of sub-solutions to create a hierarchy of sub-problems that together form a whole solution. The idea of clustering the populations of a coevolutionary system into the different similar solutions was also investigated in [75], where similar strategies in the populations were 'packaged' and evolution is on a package level as well as an individual level. This allowed the fitness to be calculated at both an individual and package level, and might be something worth investigating at a later date with our clusters.

The design of the triggering mechanism was based on the clustering of relevant information. This meant that if a trigger for a particular situation came up, the cluster relating to

that situation could be called upon and an individual selected with relevant rules. The problem with this mechanism was the complexity involved with the amount of clusters needed to represent each change in the environmental situation. To minimize this complexity, it was decided to focus on the percentage of war change (*pwar*) scenario for this experiment, and the memory was clustered on this single changing factor. The *pwar* is used to determine when war will break out. At the end of each year a random number is generated, and if the number is smaller than the *pwar*, the game is over and the scores are calculated. This 'race against time' scenario is one of the main dynamics of the TEMPO environment, and as a result is one of the larger influences on strategies.

To cluster the individuals five vectors are used, one for each of the membership functions for *pwar* (very low, low, medium, high and very high). Each time an individual is added to the memory, if it has any rules that use the *pwar* input, it is classified under the relevant vector(s). The coevolutionary game-play was then modified to make use of these vectors by changing the opposition as the *pwar* increased in the current game. For example, at the beginning of the game the *pwar* input would be low, so the cluster for the *pwar* membership function *low* would be used to choose an opposition. Once the *pwar* value changed to medium, then the *med* cluster would be selected, and so on. There were a number of issues involved in this, such as what would happen for multiple triggers (i.e. the *pwar* is very low to low), what if no rules are available for triggering, and what rule to select from the vector if triggered.

To solve the scenario of multiple vectors triggered, a very simple mechanism was implemented that involved selecting an individual from the first vector triggered, then giving a percentage chance that the next triggered vector would be used instead. The chance was dependent on the membership ratio of the *pwar* input, so the higher the membership in the second membership function, the greater the chance that function would be used.

If no individuals could be found from the clusters for the current *pwar*, then the individual would be selected based on a linear time bias from the entire long term memory (LTM). This mechanism still differed from the previous use of LTM, as the enemy was still being changed during the game-play.

The issue of what rule to select from the cluster was similar to the issues dealt with previously, and so two mechanisms were tested. Initially the individual was chosen randomly from the cluster, and then a time based linear probability distribution was applied. The time based cluster distribution provided a slight improvement to the results, and was used for further experimentation.

We were expecting average results from the clustered system for our experiments, as the use of the *pwar* input to cluster the memory could potentially hamper the memory

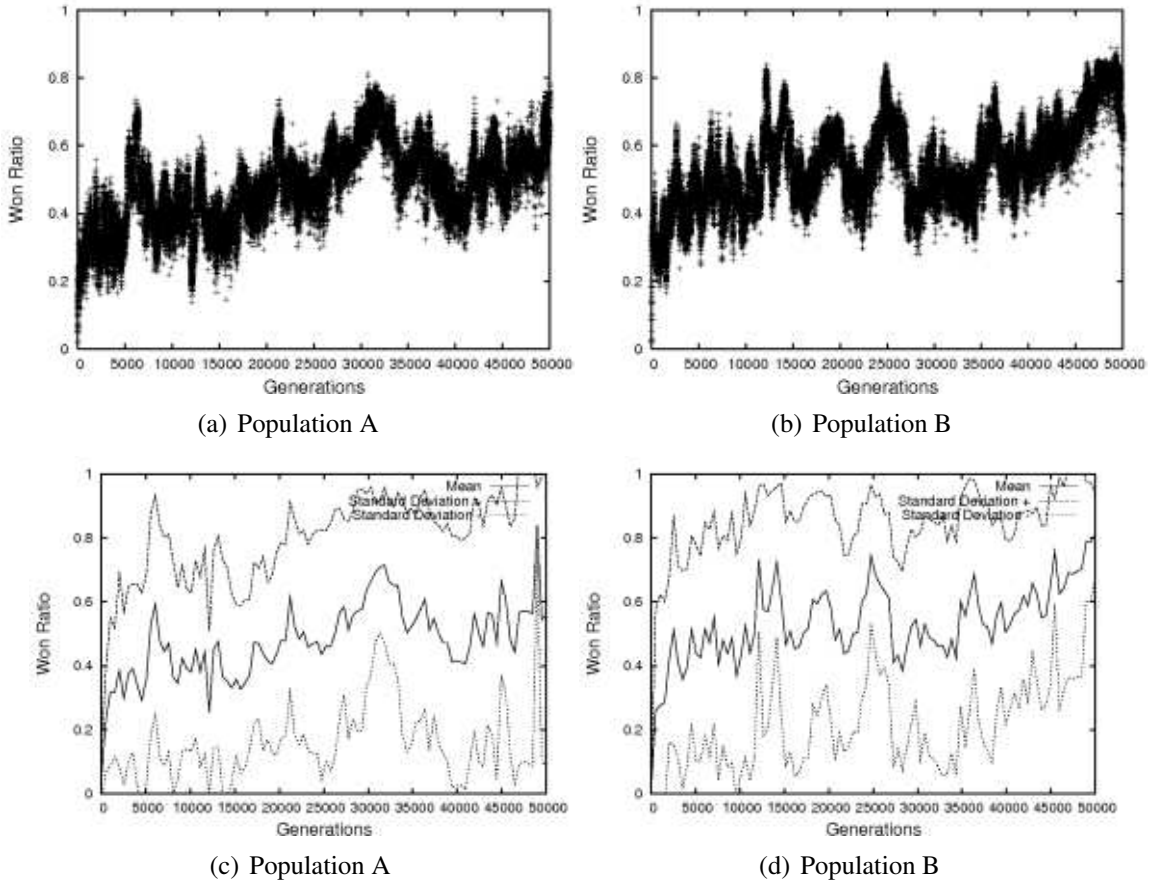
functionality. Focusing only on *pwar* means a lot of other possibly better solutions are ignored if they do not have the *pwar* input. This could then create a scenario where the populations are evolving against weaker individuals. There was also an issue involving the current alien expert, as by clustering by *pwar*, the alien expert was never being called upon to evolve against. To counter this we introduced a second "expert" individual; the *pwar* expert. As the expert was aimed at testing the *pwar* clusters, we derived a simple rule base that used the strategy of: buy offensive weaponry while the chance of war (the *pwar*) is still low, then start to build up defensive weapons when the chance of war gets higher. While the strategy is very simple, it is one that is often used by human players and (when combined with other strategies) can be quite effective. This expert was also made to be inserted into the population, and to be used as the baseline measuring mechanism as with the previous expert.

For this experiment we applied the clustering mechanism described to the long term memory. The individuals were now evaluated with $r_1 = 20$ games against the opposition, followed by $r_{2S} = 10$ games against the short term memory, and $r_{2C} = 10$ games against the clustered long term memory.

When we ran the clustered system using the same strategy as the previous experiments, the results were lacking when compared to the normal short and long term memory mechanism. We thought that this might be caused by the fact that the current expert had no reliance on *pwar*, and our clustered mechanism was tailored to adapt to *pwar* variation. To test this we created the new *pwar* expert to test against as described. We then ran the clustered mechanism again, this time replacing the static benchmarking expert and the alien expert with the *pwar* expert. The clustered mechanism performed better against this expert, and the results from this can be seen in figure 6.13.

For comparison purposes we then performed an experiment with the long term memory set to the simple linear time based probability selection using the *pwar* expert. The results from this can be seen in figure 6.14.

The clustered mechanism performed very differently to the normal long term memory selection mechanism, with very large variations and a slower climb towards regularly beating the static expert. This slow trend to improve was accentuated by the large variations, and was something we had not seen in prior experiments. The slow trend was something that was partly expected, as the clustering mechanism may overlook better opponents from memory to evolve against if they do not include a rule for *pwar*. Taking this into account, it was interesting to see the increase towards better solutions towards the end of the generations. This is rather promising, and a future step in this direction is to incorporate the other dynamic inputs into clusters, such as the budget and the weapon types and categories. The



(a) Population A (b) Population B

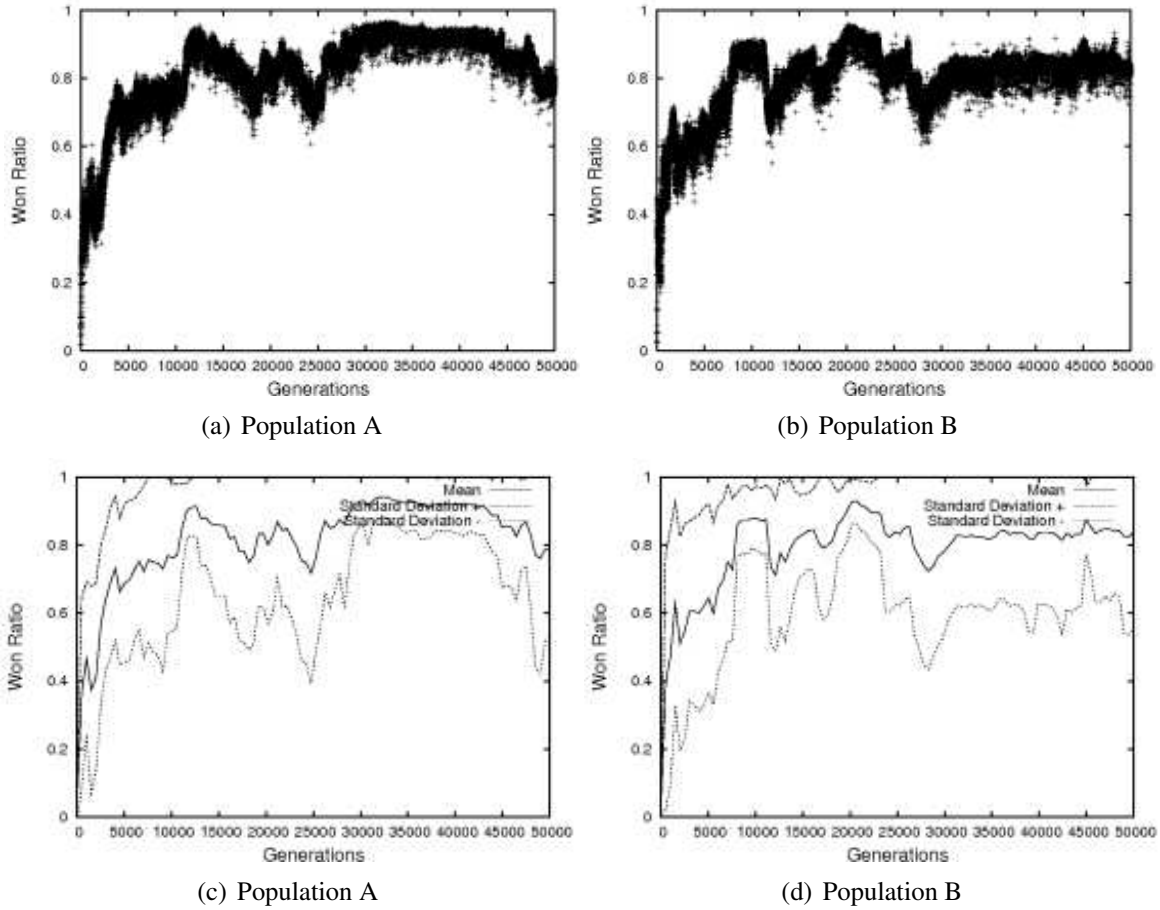
(c) Population A (d) Population B

Figure 6.13: Using the clustered memory mechanism against the *pwar* expert

variation itself was not expected to the degree seen, and reasons for this fluctuation are still to be investigated. It is possible that the clustered memory largely contains very weak or very strong rules due to only applying them by their *pwar* individuals against the expert, and goes through phases of applying these.

Interestingly, when comparing the linear time probability results in figure 6.14 to the same memory mechanism with the original expert instead of the *pwar* one as shown in figure 6.8 (in section 6.6), it can be seen that initially the coevolutionary process takes more time to improve against the baseline expert. In the long run however it performs much better against this expert than against the old one. The individual runs against the *pwar* expert showed large and consistent sequences of 90-100% won ratios against the expert. The *pwar* expert appears to be very easy to beat consistently by the later solutions, but works well against earlier solutions.

The clustered system presented in this research was the first step in creating a long term memory that uses the changes in the environment to trigger appropriate coevolution. The observations from the experiment displayed results that were very different from previous



(a) Population A (b) Population B

(c) Population A (d) Population B

Figure 6.14: Success ratio against the *pwar* expert using linear time probability long term memory

experiments, with a highly varied result against the *pwar* expert. This was very interesting, as it showed that on average there were variations that occurred drastically for all the runs. This variation is likely caused by restriction to the *pwar* input, as many individuals from memory (that may well be higher fitness individuals) are not chosen for evaluation against, as they do not have any *pwar* fuzzy rules. This means that the fitness of the population can rise as a result, and vice versa for the opposite scenario. This is something that needs testing in future research, as more of the input classifications are added to the system. Overall however, the clustered system improved in its rating against the *pwar* expert and provided a very promising result.

The end goal of the clustered system is to have clusters for each of the inputs to the fuzzy rules (e.g. budget, weapon category, type etc.), and then evaluate against a number individuals from the different clusters that match with the current game scenario. By doing this, we can also lead up to another goal, which is to incorporate a hierarchy to the game-

play. It is possible that it is better to have rules relating to weapon category (defence, offence) before looking at type and subtype, so we could include a mechanism to evolve more times against rules including the category followed by the type and so on. As a first step in this goal, the clustered system performed very well, and the mechanism works with interesting results.

6.11 Reassessment of the Alien Expert

The alien expert was used to test the effectiveness of the memory to retain and prompt future generations, however it was never really tested in this capacity. This section gives further details of the alien expert and its use, and introduces a strategy of evolving against the expert.

Early research for this thesis showed that the memory mechanism did manage to improve on the fitness against the static expert, but no further research was conducted into whether the alien expert provided any assistance to this improvement by being in the memory. The research presented here investigates this matter further.

As discussed previously, the alien expert experiments involved inserting a simplistic human-based expert individual into the coevolutionary process and seeing how it fared against the other individuals. As was expected, a solution to beat the individual was soon evolved, and the individual itself was disseminated through the future generations. The alien expert was then used in the subsequent experiments to see how well the memory worked in overcoming past strategies. The use of the alien expert in this capacity was never tested however, and the effect the expert had on the evolutionary process needed further investigation. To effectively test the alien expert, we have performed further tests and experiments with the results presented here.

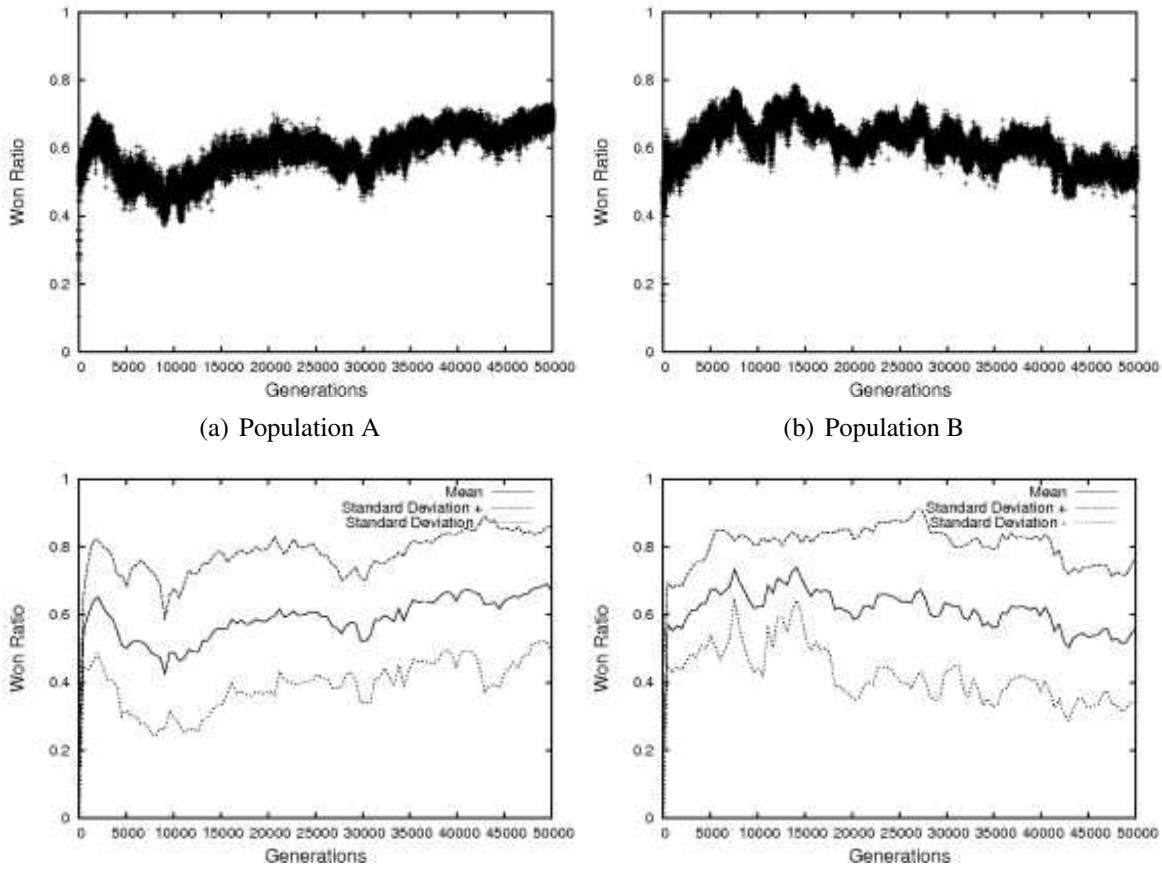
To test the usefulness of inserting the alien expert into the population at the beginning of the evolution, a number of scenarios were put in place. All experiments involved running the system using the short and long term memory mechanism with linear time distribution. The first experiment was run with no alien expert included. We were expecting that the results would show only a slight increase in the fitness against the static expert, as there was no reward for beating it. In theory, as described in section 6.3, the alien expert should prompt the populations to create solutions that perform well against the expert. This should begin in the populations, and continue from the memory. Therefore, we were expecting the alien expert experiment to outperform the one without an alien expert.

In addition to this, investigation was also conducted into the possibility of evaluating *against* the expert each generation. Originally, the best individuals were played against the

static expert at the end of each generation for the benchmarking progress, but the fitness of the individual was not updated to reflect the game-play. We hypothesized that by evolving against the expert we should see a sharp incline in the fitness against the static expert, as we were effectively training the individuals against the expert.

A tracking mechanism was put in place to tell when the alien expert was called upon in the fitness evaluation process. This meant tagging the expert upon insertion into the population, and any times it was inserted into the memory – from the first generation and any subsequent generations when reused unchanged through elitism. Each time a tagged individual was called upon from the memory to participate in the game-play for evaluation it was recorded.

The first experiment was run with no alien expert inserted. The fitness was calculated by playing $r_1 = 20$ games against the opposition, followed by $r_{2S} = 10$ games against the short term memory and $r_{2L} = 10$ games against the long term memory. The graph of the won ratio against the expert can be seen in figure 6.15.



(a) Population A

(b) Population B

(c) Population A

(d) Population B

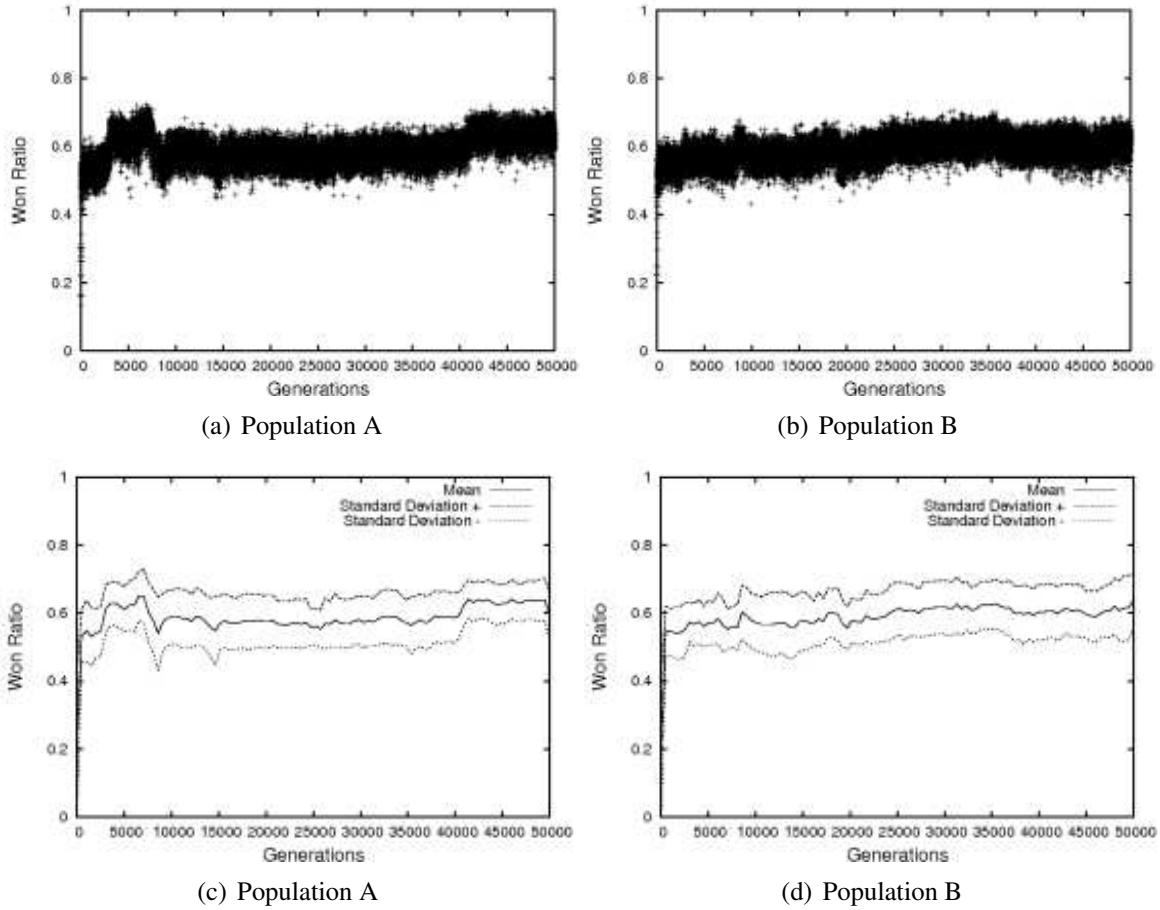
Figure 6.15: Success ratio against the expert with no alien expert

When compared with the results from figure 6.8 (the experiment with the same parameters as this one, but with the seeded alien expert), the results showed that the alien expert stabilized the evolution with less fluctuations in the results. However, the fluctuations of the experiment without any expert produced some better results in some of the peaks of the evolutionary process. Also of interest, was the speed at which the coevolution was able to regularly beat the expert. We were expecting the experiment without an alien expert to show a slow increase against the expert, and a lot of variation with worse performance overall than the alien expert experiment. What we found instead, was the memory by itself quickly improved to beat the expert, even without any mechanism to know of the expert's existence.

This result led us to investigate whether the insertion of the alien expert into the populations may actually hinder the evolutionary process. As noted by Rosin and Belew [86], having an initial opposition population that is hard to beat can stunt the evolution. The individuals will not fare well against the opposition, and there will be less variation to guide the search. By inserting the alien expert into the population we are essentially adding an individual that initially beats almost all the randomly generated individuals in the populations that it plays against (from the opposition population or from the memory). Without the expert, the populations are free to search and improve without having their fitness decreased by being evaluated against the expert. This means that they are able to explore a more diverse range of solutions, and create possibly better solutions against the expert that are not specifically tailored to beat the expert.

As further testing, we directly evaluated the populations against the static expert. Instead of inserting the alien expert into the populations, we included an evaluation against the expert each time the fitness was evaluated. We performed an experiment with this method, with the same memory mechanism as previously, but with each individual being played an additional game against the expert for fitness evaluation. As each individual plays a minimum of 40 games for their fitness evaluation ($r_1 = 20$, $r_{2S} = 10$, $r_{2L} = 10$), a single game against the expert would not really affect the fitness. To counter this, we increased the weighting of the game against the expert by a factor of 15% of the total games played. The results of this experiment can be seen in figure 6.16.

The results showed even less variation than the results with an alien expert, and there was also a lower won ratio when playing against the expert for benchmarking measurement. This follows the previous hypothesis, that by evolving directly against the expert we are hindering the evolutionary process. Our initial thought when developing the evaluation against the expert mechanism was that we were effectively training the evolution against the expert, and so the populations should increasingly become better against the expert.



(a) Population A (b) Population B

(c) Population A (d) Population B

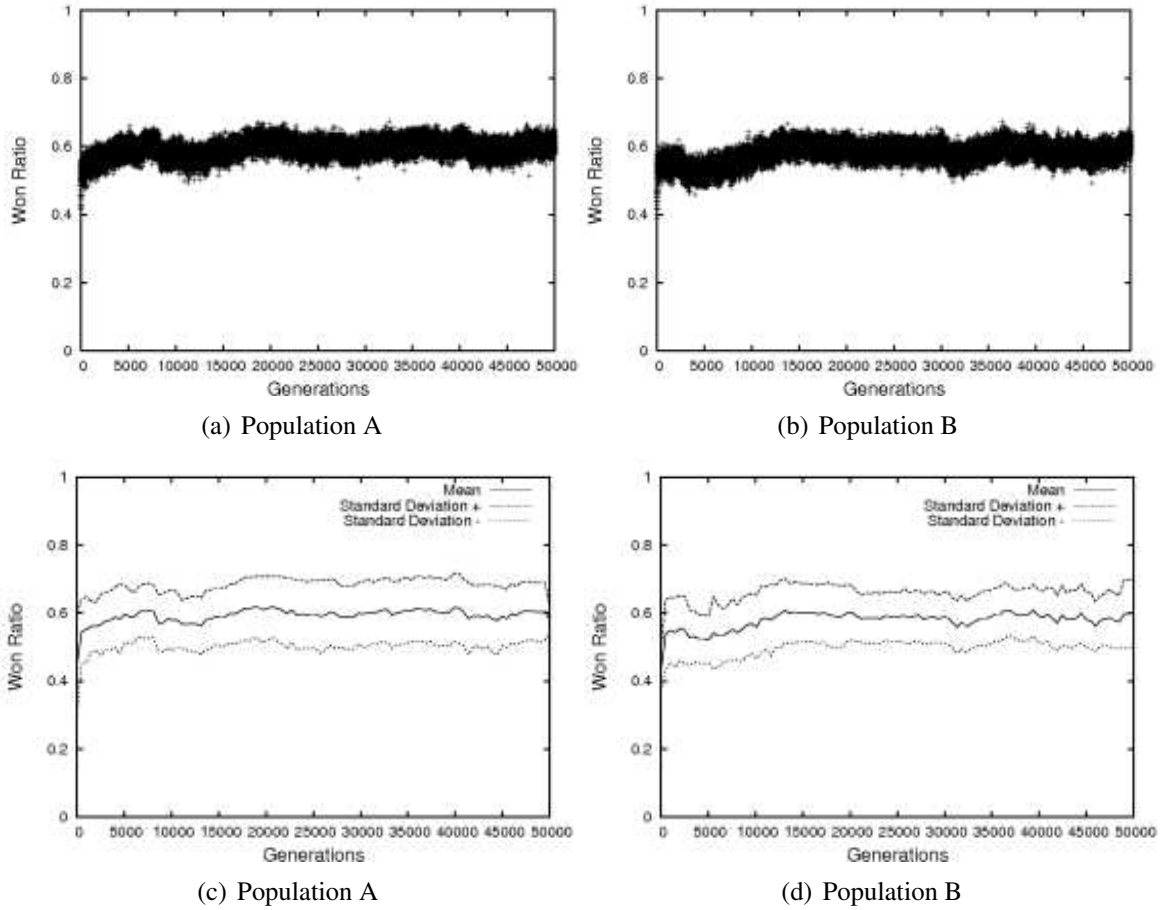
Figure 6.16: Success ratio against the expert with individuals evaluated against the expert

Instead we found that by leaving the evolution to develop its own strategies to evaluate and play against, there were far better results. By evaluating against the expert we are restricting the variation of the populations and the memory.

As a final experiment, we included both the alien expert seeding, and the fitness evaluation against the expert mechanism. The results from this can be seen in figure 6.17. This shows again that the alien expert stabilizes the populations, with less variation seen than in than in figure 6.16. When compared to figure 6.8 with the alien expert included, it can also be seen that the evaluation against the expert decreases the average won ratio.

From the experiments above, we see that the best performing mechanism was when no expert was included in the evolutionary process. Although the seeding of the alien expert does stabilize the evolution results, the performance of the experiments with no expert reached higher won ratios and performed better overall.

As an interesting observation, the stabilization occurring with the alien expert showed the continued affect of the expert in the memory. This however, restricted the coevolution-



(a) Population A (b) Population B

(c) Population A (d) Population B

Figure 6.17: Success ratio against the expert with alien expert seeding and evaluation against the expert

ary mechanism and the populations were not able to traverse the search space for solutions that were stronger strategies. We hypothesize that this is due to the alien expert repeatedly beating the early individuals and hence lowering their fitness and diversity overall. Then when the populations evolved to overcome the expert, they had developed a set of solutions and a memory affected by, and tailored to, the expert.

When the alien expert is left out of the evolutionary process, the populations are free to develop a wide and varied set of solutions that can be more efficient at beating the static expert in the long run. This is a very interesting observation, as it goes against what the expected result was. It shows that the use of human knowledge inserted early into the coevolutionary process does in fact hamper the result and not help to obtain better solutions.

6.12 Experiment Parameter Comparisons

A summary of the different experiments and the difference in parameters can be found table 1. The information depicted is as follows. The type of memory mechanism used is single for the initial case, and then split into short and long, with the long replaced with gladiator and clustered in latter experiments. The *Probability Selection Distribution* depicts the type of distribution used to select an individual for game-play from the memory. The *Game Number* shows the number of games each individual from each population must play to calculate the fitness: r_1 is the games played against the opposition, r_2 is games against the memory, r_{2S} is games against the short term memory, r_{2L} is games against the long term memory and r_{2G} is games against the gladiator long term memory, and r_{2C} is games against the clustered long term memory. The *Memory Lengths* show what size each of the memory mechanisms are, where L is the single memory length, L_s is the short term memory length, L_l is the long term memory length and L_g is the gladiator memory length. The *Unique* field is a boolean field representing whether the memory applied uniqueness of individuals or not. Finally, the *Alien Expert* field depicts if the populations were seeded with the alien expert for the experiment.

Table 6.1: Summary of the experiments

Exp.(Section)	Memory	Prob. Select Dist.	Game No.	Memory Length	Unique	Alien Expt.
Baseline(6.3.1)	$\emptyset$	N.A.	$r_1 = 20$	N.A.	N.A.	no
Memory(6.4)	Single	Uniform	$r_1 = 20$, $r_2 = 20$	$L = \infty$	yes	yes
Linear(6.4)	Single	Linear	$r_1 = 20$, $r_2 = 20$	$L = \infty$	yes	yes
Laplace(6.4)	Single	Laplace	$r_1 = 20$, $r_2 = 20$	$L = \infty$	yes	yes
Short & Long Term(6.6)	Short, Long	Linear	$r_1 = 20$, $r_{2S} = 10$, $r_{2L} = 10$	$L_s = 10$, $L_l = \infty$	yes	yes
Unique(6.7)	Short, Long	Linear	$r_1 = 20$, $r_{2S} = 10$, $r_{2L} = 10$	$L_s = 10$, $L_l = \infty$	no	yes
Gladiator(6.8)	Short, Gladiator	Ranked	$r_1 = 20$, $r_{2S} = 10$, $r_{2G} = 10$	$L_s = 10$, $L_g = 1000$	yes	yes
Migration(6.9)	Short, Gladiator	Ranked	$r_1 = 20$, $r_{2S} = 10$, $r_{2G} = 10$	$L_s = 10$, $L_g = 1000$	yes	yes
Clustered(6.10)	Short, Clustered	Linear	$r_1 = 20$, $r_{2S} = 10$, $r_{2C} = 10$	$L_s = 10$, $L_l = \infty$	yes	no
Expert Tests(6.11)	Short, Long	Linear	$r_1 = 20$, $r_{2S} = 10$, $r_{2L} = 10$	$L_s = 10$, $L_l = \infty$	yes	no

6.13 Conclusions and Further Work

The experimentation of memory in a coevolutionary system has shown that the short and long term memory approach was beneficial, particularly when applying a structured long term memory. We demonstrated that the sorting and selection mechanism for the memory affects the usefulness of the memory, and that by including a specialized memory selection mechanism represented by the short term memory, the system could make better use of its memory.

The short term memory mechanism works like the human memory mechanism for learning; it allows a person to overcome information overload and use the most useful and recent memory rather than their large long term knowledge base. The long term memory is still there for prompting historical memories and reminding the person of past successes and mistakes, and allows the person to grow overall. We have attempted to replicate this in the TEMPO system, allowing the system to have reminders of past strategies, while still giving the best information for the current situation.

The inclusion of more than one selection mechanism from a memory based on long and short term memory has many different areas that could still be investigated. The idea of an even more biased memory selection, where the current situation could affect what strategies from memory to play against, was investigated in the clustered experiment. However, more work is required in this area. As mentioned in section 6.5, the long term memory in a human is divided into explicit and implicit areas. It might be beneficial to try and find a way to mimic this. In particular, this involves looking at the linking of relevant information for retrieval from the long term memory. While we attempted this to some success with the clustered mechanism, there remains a lot of room for improvement and investigation. It might even be possible to include some mechanism of adaptation to the clusters using some generic semantics to represent 'social' learning for the whole population.

The use of a 'gladiator' system for the long term memory worked well. The mechanism created some very good results, and is worth future investigation. One possible area for research is the ranking mechanism used. Currently a very simple stochastic mechanism is used to rank the individuals, and it is likely that a better mechanism could be incorporated.

Another area that remains to be investigated, is to have an adaptive number of games played against the memory dependent on the environment and the current stage of evolution. At the moment there are user defined parameters representing each set of games to be played to get the fitness, including games against the opposition and the memory. It might be that a better alternative would be to let the system itself decide what number of games to play. To this end, investigation has begun into the use of a fitness threshold, where the

number of games played adjusts itself as the fitness variation increases or diminishes.

The use of memory with the TEMPO system has given rise to some very challenging computer players. The individuals provide a much greater challenge to human players than the individuals created without memory. However, the computer player produced is still a static rule base that human players can adapt to given enough time. The major goal of our research is to create a system where the individuals are evolving and adapting to beat a particular human player during real-time game-play. The use of memory was the first step in this research, and was necessary for the creation of the adaptive system discussed in chapter 8.

Another deficiency that remained after the memory experiments was the area of intelligence and counter intelligence. Unfortunately, the addition of memory did nothing to improve the usage of intelligence by the computer players. Further investigation into why intelligence was not being bought was needed. The next chapter describes the research undertaken to this end, the changes made to the TEMPO game to incorporate intelligence, and the interesting discoveries made in the process.

Chapter 7

Intelligence and Counter Intelligence

Strategic decision-making done in parallel with the opposition makes it difficult to predict the opposition's strategy. An important aspect in deciding a move is evaluating your opponent's past moves and using the evaluation to predict future movement. In the game of TEMPO this is done through the purchase of intelligence, which gives you a relative view of your opponent's choices. The research presented here seeks to evaluate the way this intelligence is used in the current game, and present an alternative method of representation. This alternate mechanism is then used in the coevolutionary system to obtain a computer player that self-learns the importance of using opposition data in strategic decision-making. The research presented here is partly from a paper published with Garrison Greenwood [7].

7.1 Introduction

The United States military is currently implementing the most comprehensive transformation of its forces since World War II. The goal is to improve the joint force warfighting capabilities to meet current and future full spectrum requirements [30]. This new methodology, called *Operational Sense and Respond Logistics* (OS&RL), is based on the tenets of adaptive systems. This has the underlying assumption that the military establishment, and the threat environment under which the military operates, is inherently an adaptive process [66].

The sense and respond (S&R) capability deals with an enterprise that is complex, adaptive and nonlinear. Sensing of operational needs in real-time, and providing response to those needs within the time frame that meets commander's intent, is critical to success of the enterprise. Adaptation (and the speed of adaptation) in complex environments is the most critical factor in meeting the objectives of the enterprise. Operations, logistics, and intelligence aspects of military command must be integrated to coherently execute com-

mander's intent.

Adaptation is essential to OS&RL. Under this new philosophy logistics can only be effective if it adapts to an evolving commander's intent and the current combat situation. OS&RL supports the integration of operations, intelligence, and logistics through providing the following capabilities:

- Support the concept of force capabilities management through continuous integration and coordination of logistics capabilities with operations, intelligence, surveillance, and reconnaissance functions.
- Synchronize logistics operations with commander's intent, operations functions, and intelligence, surveillance and reconnaissance (ISR) by maintaining and exploiting total situation awareness based on: evolving commander's intent; the strategic, operational, and tactical situation; the operational environment; and force capabilities.
- Anticipate force capability and logistics needs to proactively sustain the force and alter initial conditions.
- Provide commanders with operations and sustainment options based on predictive adaptive logistics.
- Implement commander's intent, expressed in effects, missions, and tasks, in every aspect of logistics, across the full range of military operations, and for the full set of force capabilities.

The game of TEMPO has been used for many years as a tool for developing skills at resource allocation in a war scenario. It is an iterative game, where at each turn you and your opponent allocate individual budgets to cover a range of weapons and intelligence categories unique for each player. At the end of the turn, you are given information about the number of utilities you have left after the opposition's corresponding utilities force has countered yours. If you have bought intelligence (INTEL), then you also receive some information about your opponent's current utilities; if you bought counter-intelligence (CI), you restrict the information your opponent receives.

Unfortunately, TEMPO currently does not handle INTEL or CI information in a very realistic manner. TEMPO treats INTEL information as a boolean decision where you either buy it or don't buy it. This decision-making strategy does not match reality. Military combat operations and logistic support operations always incorporate INTEL information into the planning process, and use CI to help safeguard those operations. However, INTEL and CI information can only affect plans if it is timely and digestible.

TEMPO also provides relatively useless information about the INTEL received from the allocation of budget to it. For example, a TEMPO player could receive INTEL information saying the opponent has "some equipment of weapon type B". Such ambiguous information is useless for logistic planning purposes and inadequate to properly train logisticians.

OS&RL is the wave of the future and logistic planners need training in its fundamentals. TEMPO in its present form cannot accommodate OS&RL scenarios, primarily because of the way INTEL and CI are implemented. Moreover, TEMPO has no current capability to incorporate commander's intent. It does however provide a framework for changing battlefield scenarios, since the game iterates until war is declared.

The chapter is organized as follows. We begin by describing how INTEL and CI are used in military strategic decision-making. We then describe our methodology and the changes we made to TEMPO and the TEMPO computer system to improve INTEL and CI information handling. Following this, we present our experimental results. Finally, we outline our conclusions and future TEMPO modifications that will further support OS&RL scenarios.

7.2 A Discussion on the Role of Intelligence

The following section elaborates on how intelligence (INTEL) and counter intelligence (CI) are used in military planning. Although most discussions on these topics are from the perspective of the battlefield commander, this research targets the logistics decision maker.

7.2.1 Intelligence (INTEL)

Intelligence provides logisticians with the requisite information needed to make informed decisions about the allocation and distribution of resources required to support the battlefield commander. *Situational development* is a key ingredient of an effective INTEL program. Situational development is essentially a process for analysing information to help the decision maker recognize and interpret enemy intentions, objectives, combat effectiveness, and potential enemy courses of action [64, 65]. The decision maker can then ensure, subject to budget limitations, that the appropriate type and quantity of equipment is procured, and sufficient resources are allocated to store, secure and maintain stockpiles under their control.

Any information about the enemy should be fully integrated into planning decisions; INTEL information serves as a constraint on other logistics decisions. For example, it is unwise to buy a particular type of weapon if INTEL information indicates enemy combat

units have weapons that effectively neutralize it.

Intelligence operations include five basic functions: plan, prepare, collect, process and produce. The first four functions are not discussed in depth here, because they are fully discussed elsewhere [65], and because they are performed by INTEL gathering agencies rather than logisticians. The last function however is worthwhile discussing.

The produce step involves evaluating, analysing, interpreting and combining information and INTEL into a form that supports logistic decisions. Intelligence information from single sources should never be taken at face value, the more corroboration the more believable the INTEL.

7.2.2 Counter Intelligence (CI)

Counter intelligence counters or neutralizes intelligence collection efforts by others. CI includes all actions taken to detect, identify, exploit, and neutralize the multidiscipline INTEL activities of friends, competitors, opponents, adversaries, and enemies [65]. Countermeasures are any actions taken to counteract attempts by others to obtain INTEL.

Either *passive* or *proactive* countermeasures can be used to thwart INTEL attempts, often with a mixture. Passive countermeasures include communications security (encrypting data), computer security (firewalls), physical security (barriers) and camouflage. Proactive countermeasures use deliberate deception to hide one's true intentions by promulgating, misleading, or wrong information.

7.3 Intelligence and Counter Intelligence in the TEMPO Military Planning Game

The game of TEMPO is divided into categories of decisions, with the top level consisting of allocations to weapons or INTEL. Most of the research presented so far has focused on analysing and improving the way the computer player created and used rules for allocating budget to the weaponry. The area of intelligence has been largely left untouched, thus this chapter focuses on this area.

The original version of TEMPO divided the allocation of INTEL resources into three categories: offensive INTEL, defensive INTEL and CI. For each category the player had the option to either buy INTEL at the set price, or go without in that category (a boolean decision). The offensive and defensive categories were used to provide INTEL on what the opposition did in the previous year in the specified category. If a player purchases offensive INTEL in one year, then the next year information on the opposition's total net

OA and OB utils for the previous year is given. However, if the opposition buys CI, and the player purchases INTEL, then the player is only given minimal information about the opposition's utils.

There are three combinations for INTEL to be received. If a player purchases INTEL and the opposition have not bought CI, then the player will receive the full numeric value of the opposition's previous year's results. If the player purchases INTEL and the opposition purchases CI, then the player is only told that utils for the respective intelligence category either *exist* or *do not exist*. If the player does not purchase INTEL then no information is given on the opposition's results, and the resulting value is *unknown*.

The coevolved system currently evolved intelligence rules, but was actively selecting rules that were not being triggered to purchase INTEL. As a result, the rules being developed were overly generalized and purchasing the weapons blind to the opposition's moves.

7.3.1 Deficiencies of Intelligence in TEMPO

It is unclear why TEMPO makes not buying intelligence even an option. The very idea that intelligence activities might not be funded not only lacks common sense, but it is foolhardy and dangerous. No warfighter would even consider implementing some course of action without getting as much information as possible about the enemy's capabilities and intentions. To do otherwise puts men and equipment unnecessarily at risk.

Logistic planning is no different. Successful offensive and defence operations demand an efficacious allocation of logistic resources, i.e. the minimum amount of resources that are correctly configured and placed at the right place, at the right time, to satisfy warfighter's requirements. Intelligence helps to formulate initial logistic plans and it provides the feedback needed to determine if already existing plans are likely to be effective when executed. Without intelligence it is impossible to adapt logistic resource allocations as the strategic and/or tactical situation develops. Military planners, regardless if they are warfighters or logistic planners, never commit resources or execute operations in the dark. Hence, in TEMPO players should always buy INTEL, the only question is *how much*.

Of course any offensive or defensive operation will not be successful if enemy forces know everything about the operational plan, as they could muster effective countermeasures. With respect to logistic planning, if the enemy acquires detailed information about planned purchases, size and operational status of stockpiled equipment, and future distribution plans, they could gain valuable clues about warfighter's intentions. CI activities deny the enemy information. Therefore, the need to purchase INTEL also applies to CI.

If all TEMPO players buy INTEL and CI then the default position is that players get

some information about what their opponent has. Unfortunately past attempts to exploit that information and allow it to influence future logistic planning have been unsuccessful. In part, that is because just knowing something without knowing all details is often not that useful. For example, if a player knows the opponent is buying offensive weapon type B, then it would be wise to purchase defensive weapon type B. However, without knowing how many offensive weapons the opponent has, it is impossible to know how many defensive weapons to purchase and decisions degenerate into mere guesses. Indeed, TEMPO is purposely designed not to reward over-purchasing to counteract an opponent's strategy.

The commander's intent should be supported as the tactical or strategic situation evolves. To achieve this goal means logistic planning rules derived from static fuzzy rule bases should be abandoned in favour of rules derived from continuously evolving fuzzy rule bases, which are better able to anticipate logistic requirements. Intelligent software agents, cognitive decision support, and historical knowledge bases are needed to provide the necessary adaptability [66]. Flexibility is now paramount and far preferable to locally optimized logistics plans.

The way INTEL is handled in TEMPO should be changed. Partial information, currently stated as "exists" cannot be effectively incorporated into fuzzy rule antecedents. One possible improvement is to provide quantified information about an opponent rather than the current three crisp values (no information, some information or 100% accurate information). Tying their effectiveness in blocking information disclosure to the amount purchased might also fuzzify CI expenditures. The more CI purchased, the better information transfer is denied.

7.4 Applying a more Realistic Mechanism for INTEL and CI

For the purposes of this research we decided to break the remodelling of INTEL up into two steps. The first step is to implement the purchase of offensive and defensive INTEL, and the second step – the purchasing of CI. The following section gives a description of the issues with each step, and the implementation choices made.

7.4.1 Redesigning Offensive and Defensive Intelligence

The redesigning of the offensive and defensive INTEL involved two areas: how should a player be able to allocate resources to the INTEL, and how the fuzzy rule system should be adjusted to integrate these changes.

To make the purchasing of INTEL more realistic, we wanted to develop a system that would create a direct correspondence between the amount of INTEL bought and the amount of information received. There are many ways to do this, but we wanted something that would be complex enough to represent a more realistic way of buying INTEL, and yet simplistic enough to give the coevolutionary system a chance of discovering meaningful rules. We achieved the complexity by breaking the purchase of INTEL down further into types (e.g. Offensive A), and allowing the player to purchase a percentage of the total cost. For example, the player can allocate budget to Offensive INTEL A (OIA) up to a certain amount. If there is a maximum OIA expenditure of 400, and the player chooses to allocate 200 out of the budget to this resource, then the player has purchased 50% of OIA. To maintain simplicity we only decomposed INTEL into Offensive and Defensive INTEL types A and B (OIA, OIB, DIA, DIB). We then gave each of the four areas a different maximum budget to spend. It is then up to the player to decide how much of that INTEL to purchase.

This new mechanism means the player can directly influence how much INTEL received in a category. For example, the player may be focusing his or her attention on Offensive A weapons, and chooses to buy a large percent of OIA, and a smaller percent (or even 0%) of INTEL in another area. Therefore, the budget allocations for the four INTEL areas require new allocation strategies by the player.

The result of buying a percentage of the INTEL in these categories is data corresponding to the opposition's total utilities in the area the purchase has been made. In addition, the previous year's INTEL bought is also displayed to the player, to help in analysing the information given for the opposition. These changes can be seen in figure 7.1, which depicts an example of what a year's game-play scenario will be.

The percentage of information bought leads on to one of the major decisions in this chapter: how to create the correspondence between the allocated budget and the information bought. One way is with a direct percentage translation. For example, if the player bought 100% of OIA, and the opposition has 500 OA utilities, then the player is given the full information of 500 opposition utilities. If the player only bought 80% however, then he or she is told the opposition has 400 utilities, and so on. The problem with this direct mapping method, is the obvious inference that is allowed. The player can directly infer how much the opposition actually has by what percentage was bought. This issue of inference made it difficult to decide what information to give the player, and eventually we decided to create a probability distribution that would be influenced by the percentage of INTEL bought. This is discussed further in the section 7.4.3.

To allow the coevolution to use the new values, we needed to extend the INTEL rules

Player's Environment

Year	Pwar	Budget Avail	Budget Left
2	12.00%	11530	11530

Previous Year's Data

Player: Type	Offense	Defense	Enemy: Type	Offense	Defense
A	1518	0	A	200	0
B	200	325	B	0	465

OIA	OIB	DIA	DIB
98.00%	76.00%	0.00%	50.00%

Current Year Allocation:

Intelligence	Cost	Allocated
OIA	300	0
OIB	200	0
DIA	250	0
DIB	150	0

Weapon	MaxAcq	AcCost	Inventory	OperCost	Utils	Opted	Bought	To Opt	To Buy
OA1	15	75	0	150	120	30	0	0	0
OB1	25	50	0	30	20	30	0	0	0
DA1	25	40	0	20	15	25	25	0	0
DB1	25	100	0	60	50	20	20	0	0
OA2	35	75	0	35	60	0	0	0	0
DA3	25	100	0	50	200	0	0	0	0

Figure 7.1: Example screen of a year's game-play with the new inputs and data

to include the new input data as described above, and change the way the INTEL budget is allocated. Previously an evolved cut-off threshold was used, where if the INTEL rule base normalized fuzzy-AND product is greater than the threshold, the INTEL is bought, otherwise not. To make the changes described above, the budget distribution was changed to the same mechanism used for the weapons; for each weapon the rules are examined, and the weapon is allocated a percentage of the budget corresponding to the amount of rules triggered and the strength of the rule outcome. The new INTEL budget is distributed in the same way, with each of the INTEL categories being allocated a ratio of the budget dependent on the triggered rules.

7.4.2 Effects on Counter Intelligence

The second area of investigation is that of CI, and the changes made for the INTEL system directly affects how CI works. Like the old INTEL system, the CI is a boolean decision of buy or don't buy. However, the same reasoning applies to how CI works as to INTEL. For each weapon category being developed and maintained by an opponent, the opponent also has mechanisms in place to stop or skew information on the INTEL getting out. The amount of resources spent on CI is usually directly proportional to its use by the opponent, and/or the strategies the opponent has in play. Regardless, the player should have the decision-making abilities available to change the amount being spent.

To allow the player to assign preventative CI, we mimic the way INTEL has been changed as described in section 7.4.1. Each weapon category/type combination has its

own allocation for CI. We decided to assign the same amount of budget into CI as the opposition does for their intelligence, however we should note that this might not be the most effective way to assign the cost for CI.

When a player purchases CI, the probability distribution standard deviation used to determine what value the opposition gets relative to the utils the player has bought, is spread by the additional percentage of CI bought. This is explained further in section 7.4.3.

7.4.3 Implementation Methods Used

As discussed in section 7.4.1, we decided to implement the way INTEL information is mapped using a probability distribution. Our implementation allocates each intelligence category (OIA, OIB, DIA, and DIB) its own probability density function (PDF). We decided on a normal distribution because it has several features that nicely capture the relative roles of INTEL and CI.

The density function for a normal distribution has an adjustable mean value, which can be set to the true value of the enemy utils for a particular weapon type. For example, if the true enemy force expenditures on weapons system A is 150 utils, then the OIA PDF has a mean value of 150. We are then able to use the standard deviation to change the distribution of what information is given to the opposition. From the previous example where the enemy force is 150 utils, if the player purchases 100% of OIA, he or she then receive the exact information (presuming no CI has been bought). If however, the player only purchases 80%, then the spread of the distribution is increased and the chances of receiving the exact value of 150 become smaller. If a player purchases 0% of the INTEL, he or she then receives the value *unknown*, to differentiate from an enemy utils value of 0.

The standard deviation for the PDF is calculated using the mean (the enemy value), with the percentage of INTEL bought used to determine the sample size. We then use the Box-Muller method (as described in [69, p. 507]) to retrieve a random number selected from the distribution, which is then mapped to the value given to the player.

The above process is depicted in figure 7.2, where we can see the curve changing as the percentage of INTEL bought decreases. The figure depicts the mean at 350 utils, with a maximum enemy utils of 800. The blue curve represents 90% of INTEL bought, the red is 80% and the green is 50%.

As mentioned in section 7.4.1, the amount of INTEL bought in a particular category directly relates to the reliability of the enemy information given. Therefore, it is necessary to display the amounts bought for the previous year in the current year's information, so the player has all the information necessary for decision-making.

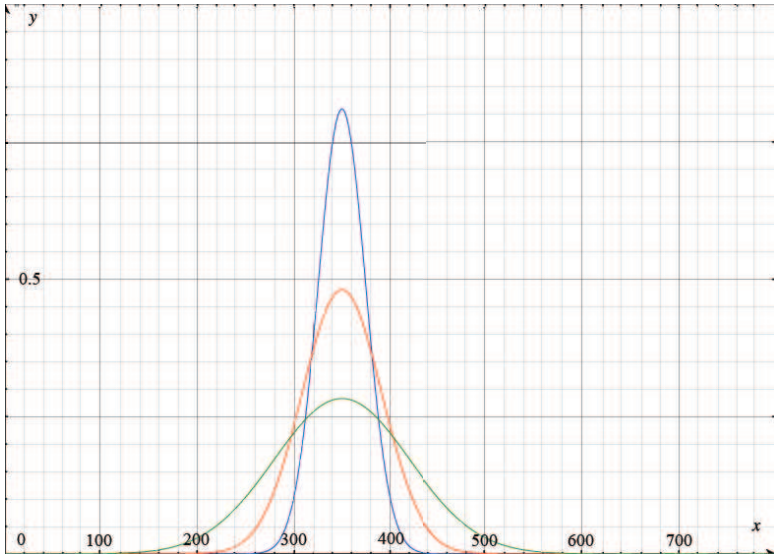


Figure 7.2: Example probability density function used for INTEL.

The standard deviation can also be used for CI to create uncertainty about the received INTEL information. The more CI bought by the enemy player, the bigger the deviation becomes. Let C_I^P be the INTEL bought by the player and C_{CI}^O be the CI bought by the opposition. We are interested in the ratio (assuming some INTEL has been bought)

$$\sigma = \frac{C_{CI}^O}{C_I^P} \tag{7.1}$$

and use this ratio as the standard deviation for the INTEL PDF. This type of standard deviation formula encapsulates the trade-off between INTEL and CI spending. The greater the PDF spread, the greater the uncertainty in the received INTEL information. For instance, if the enemy forces spend more on CI than the friendly forces do on INTEL, then we can expect the INTEL activities do not produce much meaningful information. The INTEL efforts cannot break through the enemy's CI barriers. The opposite effect occurs if friendly force INTEL expenditures exceed the enemy's CI expenditures. Our choice for a standard deviation formula increases the PDF spread whenever CI expenditures are greater than INTEL expenditures, which results in less precise information, and decreases the spread whenever INTEL expenditures are greater than the enemy's CI expenditures.

The addition of CI to this mechanism is intuitively simple. In addition to purchasing INTEL, the players also have the opportunity to purchase CI in the same categories as the INTEL available, and with the same costs as what the opposition has available to purchase INTEL. When a player purchases INTEL, the PDF is determined as described above. When the opposition purchases CI, this then affects the PDF by increasing the distribution.

For example, if a player purchases 80% of OIA, and the opposition purchases 40% of CI for OIA, then the spread of the PDF generated to obtain the opposition's OA data should be further increased relative to the amount of CI bought. The way to actually apply this however is more complicated.

Currently, with no CI being used, the function used to determine the distribution range is simply $100\% - \textit{the INTEL\%}$. Thus, when the player purchases 80% INTEL, the distribution range becomes 20%, and the chance of getting the exact opposition utils is relatively high. Now we wanted to change the function to incorporate CI. One way investigated was to directly apply the CI as is, so if the enemy purchases CI, simply add the CI amount to the distribution (with a 100% range cutoff). In the above example, this would give an original distribution range of $100\% - 80\% = 20\%$ (a relatively small range and likely to give a value close to the actual enemy utils), and an adjusted distribution of $(100\% - 80\%) + 40\% = 60\%$ (a much larger range, and less likely to give a value close to the enemy utils). This is not very meaningful, as it effectively cancels out any INTEL purchase made that is of equal or less value, and greatly decreases larger INTEL values. This was deemed too great a penalty to the purchase of INTEL.

Investigation was taken into a function that could be used to integrate CI with INTEL meaningfully. The function was developed so that $f(INTEL, CI)$ would give the range to apply to the PDF. The function $f(p, q)$ is shown below, where INTEL is represented as p and CI as q, both in the range $[0, 1]$. This function takes into account the addition of the CI to the PDF distribution range created by the INTEL (represented as $(1 - p)$), but also incorporates the average of the two values.

$$f(p, q) = (1 - p) + (1 + q)\left(\frac{pq}{2}\right) \tag{7.2}$$

Some example values applied for the function and the original $(1 - p)$ function can be seen in table 7.1. This table depicts the trends over a range of possible values.

To incorporate the CI component for the individuals, we added a third rule base for generating CI based rules. This meant that there were now independent rule bases for weapon, INTEL and CI rules. The CI rule base was given the same input parameters as the INTEL rule base, and functioned in the same way.

To investigate the effect of incorporating CI into the system, we ran experiments with and without the CI included in the system. The next section describes the experiments run to investigate the addition of INTEL, followed by the addition of CI.

Table 7.1: Example values and function results

p	q	$f(p) = (1-p)$	$f(p,q) = (1-p) + (1+q)\left(\frac{pq}{2}\right)$
1	0	0	0
1	0.25	0	0.16
1	0.5	0	0.38
1	0.75	0	0.66
1	1	0	1
0	1	1	1
0.25	1	0.75	1
0.5	1	0.5	1
0.75	1	0.25	1
1	1	0	1
0.1	0.9	0.9	0.99
0.25	0.75	0.75	0.91
0.5	0.5	0.5	0.69
0.75	0.25	0.25	0.37
0	0	1	1
0.1	0.1	0.9	0.91
0.25	0.25	0.75	0.79
0.5	0.5	0.5	0.69
0.75	0.75	0.25	0.74
0.9	0.9	0.1	0.87

7.5 Experiments and Results

Each experiment was conducted for ten separate runs, all with the same environmental and evolutionary configuration. The system was run each time for 10,000 generations with a population size of 100 for both populations. Later experiments with CI have 50,000 generations, but all other parameters are the same. The coevolution is performed as previously, using the short and long term memory selection distribution as described in section 6.6, with the same evolution parameters.

The results show the won ratio against the expert, measured as the average over 100 games with the best individual from the population (A or B) for the generation. The y axis displays the average won ratio over the ten runs. The x axis displays the generation number. Additionally, the results are also shown with the Standard Deviation (SD) range. The SD results have been smoothed using a Bezier curves function to accentuate the changes.

7.5.1 Experiments with the new INTEL Mechanism

The changes to INTEL in the TEMPO game were done for two purposes: to present a more realistic game-play, and to try and prompt the evolved individuals into using the INTEL to make their decisions. Previously, although rules regarding INTEL were being evolved, the evolved rules were rarely being used to purchase any INTEL, and were essentially useless. By including a more realistic way of getting information from the enemy, we were hoping

the evolutionary system would make use of this.

Our first experiment involved implementing the INTEL changes described in section 7.4.3, and analysing the rules and the purchases made from the rules. It was immediately apparent that the rules being evolved were now buying INTEL, but still nowhere near regularly. It appeared that the rules that bought INTEL were largely disregarded by the evolution, as the INTEL bought was not being used by the weapon rules being evolved. Instead the weapon rules were being evolved to be generic rules that bought the most weapons for a given situation regardless of what the enemy was doing.

To give the evolutionary process some encouragement to use the INTEL rules, we decided to implement the following changes. First, we made each individual purchase the full INTEL for each category for the first iteration of the game. This was to encourage the weapon rules to be developed using this information. We then lowered the cost of buying the INTEL, as we reasoned the system was deliberately avoiding the 'useless' INTEL costs and instead focusing on just buying the weapons.

The last change made was to the fitness function. As it was, the fitness was rewarding the rules for purchasing weapons only, as it was calculated using the total net utilities left at the end of each game. This meant that although we as humans can see a direct correlation with the importance of buying INTEL, the evolutionary system was not picking up on this and was instead focusing on rules that were buying the most weapons – regardless of what the enemy was doing. To try and encourage the use of INTEL, we added a reward to the fitness for each used INTEL rule created.

The results given from the above changes gave an increase in the number of INTEL rules and the corresponding INTEL bought. However, after analysing the outcome of some of the results, it appeared that the INTEL being bought was still not being used by the weapon rules being evolved. There was no coordination between the evolution of the two sets of rules, even after 10,000 generations. Hence, we wanted to test the importance of using INTEL in the rules, and developed a rudimentary expert that would play a strategy using rules that take advantage of the INTEL bought. The rules were as follows:

RULE 1:
if [EnemyOA IS Very High]
[Category IS Defensive]
[Type IS A]
then [Evaluation IS Very High]

RULE 2:
if [EnemyOA IS High]
[Category IS Defensive]

[Type IS A]
then [Evaluation IS High]

RULE 3:
if [EnemyOA IS Med]
[Category IS Defensive]
[Type IS A]
then [Evaluation IS Med]

and the same for low and very low. Then for EnemyDA:

RULE 6:
if [EnemyDA IS Very Low]
[Category IS Offensive]
[Type IS A]
then [Evaluation IS Very High]

RULE 7:
if [EnemyDA IS Low]
[Category IS Offensive]
[Type IS A]
then [Evaluation IS High]

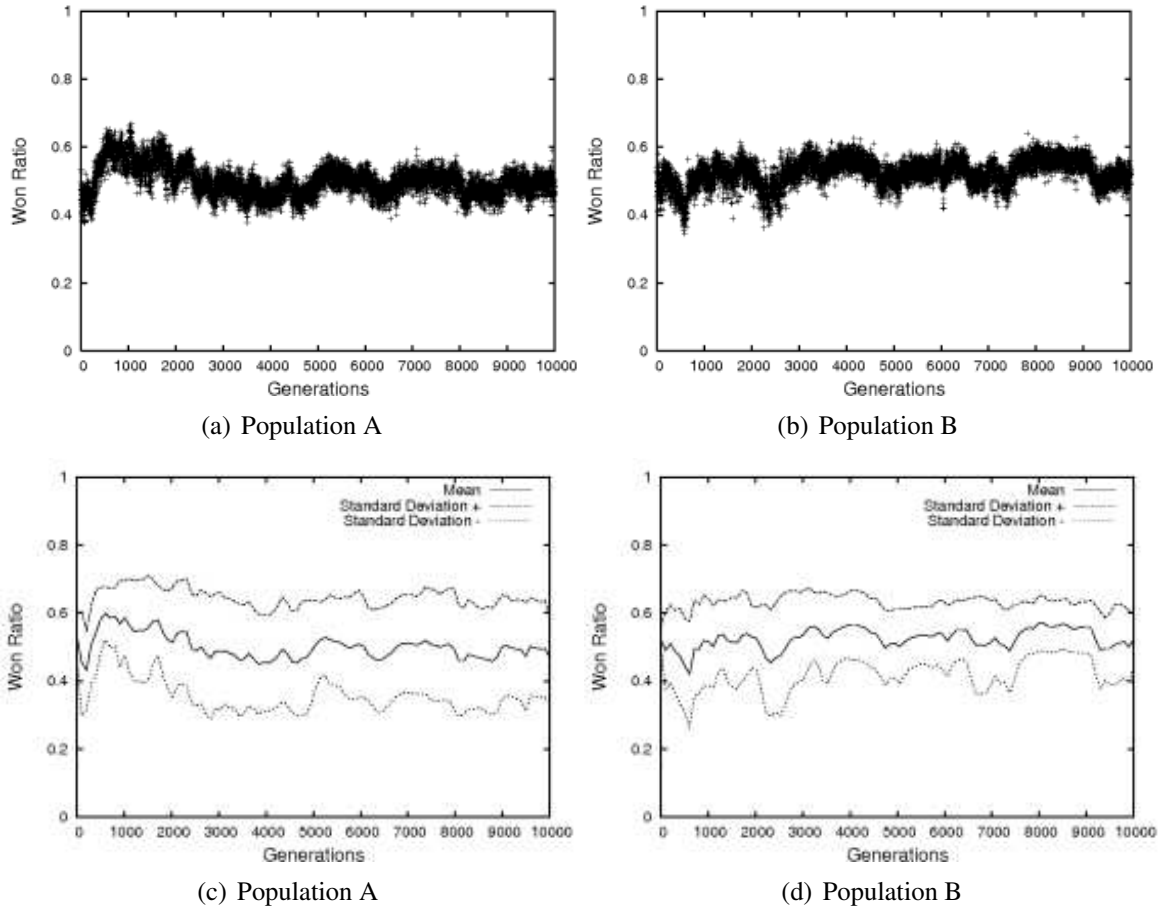
and so on, with the same rules again for Type B.

The basic strategy behind these rules was to directly counter whatever the opposition were doing. If the opposition is buying up in offensive weapons in a particular type, then buy defensive weapons in the same type to counter this activity. If the opposition is buying lots of defensive weapons of a category/type, then do not waste money by buying offensive weapons in this type (as these are countered by the defensive weapons). However, if the opposition has low defences in a particular type, then buy up offensive weapons of this type.

This expert was then placed into the evolutionary process to see how it would fare. It showed a distinct dominance in the evolution, and some of its traits (sections of the rules) could still be seen in most runs even after thousands of generations. This showed us that the use of INTEL was important when developing rules through the coevolutionary process. We decided to use the expert as a benchmarking measurement and performed the previous experiments mentioned again, measuring against this expert.

The results of running the system against the expert as a benchmark (and with the expert still inserted in the populations) can be seen in figure 7.3.

The results showed that even after 10,000 generations, almost every run had the best



(a) Population A (b) Population B

(c) Population A (d) Population B

Figure 7.3: Success ratio against the 'intelligent' expert with same expert inserted in populations

individual still buying a large percentage of INTEL, and actually using it to buy weapons.

Analysing the rules developed is very interesting, as the rule bases that are successful against the expert seem to have a strategy of focusing on one aspect of the allocation. For example, one of the individuals that won 70% of its games against the expert focused only on weapons of type B. Of the ten INTEL rules evolved, six were regarding type B, with only two rules (that were never triggered) for type A, and another two generic rules. The weapon rule base then focused mainly on weapons of type B, often taking into account the INTEL received. For example following are the first three rules from the weapon rule base for the individual:

RULE 1:
if [Category IS Offensive]
[Type IS B]
then[Evaluation IS high]

RULE 2:

if [PWar IS Low]
[Budget IS Very Low]
[SUBTYPE IS 2]
[Inventory IS Low]
[MaxAcquisitonUnits IS Very High]
[Utils IS Medium]
[UtilsPerAcquisitionCost IS Medium]
[YearAvailable IS Medium]
[EnemyOffUtilsB IS Very Low]
[EnemyDefUtilsB IS Very Low]
then [Evaluation IS medium]

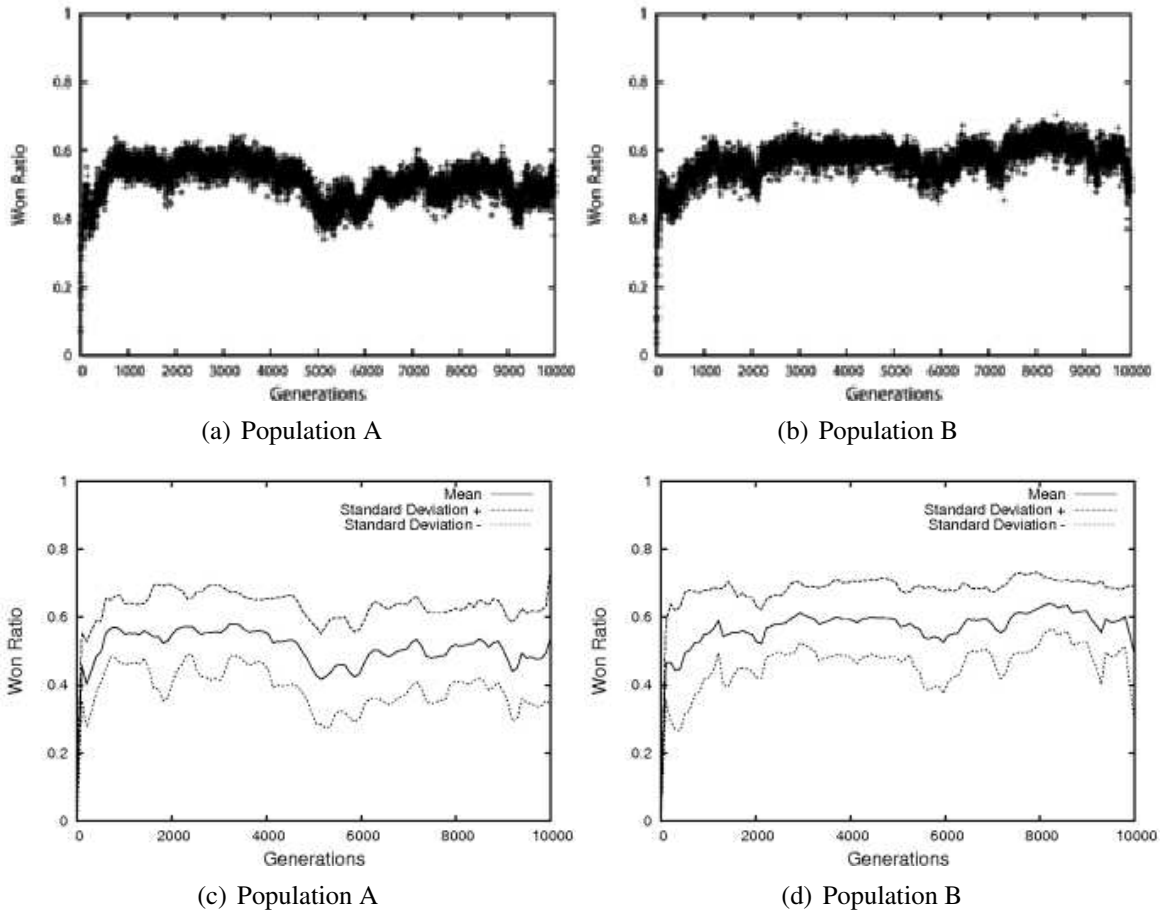
RULE 3:

if [PWar IS High]
[Budget IS High]
[OperationCost IS Very High]
[UtilsPerOperationCost IS Low]
[YearAvailable IS Very High]
[EnemyOffUtilsB ISLow]
then [Evaluation IS very high]

We also performed the same run without the expert inserted into the populations. These results can be seen in figure 7.4, and show the coevolutionary system performs quite well against the expert. The rules being evolved also show that the majority of best individuals were actually purchasing and using a large degree of INTEL.

We measured the new system of INTEL against the very simple utils/operation cost expert used in the previous experiments. The results of the new system against the expert (without the new expert inserted into the populations) can be seen in figure 7.5. Previously, the won ratio for the system (using the Short and Long Term memory mechanism) against the old expert had an average of around 60%. We can see here that the system has achieved greater results with the redesigned INTEL.

The last experiment performed was to attempt a stronger correlation between the weapon and INTEL rule base. We wanted to reward weapon rules that bought weapons using information provided by the INTEL rules. To do this, we separated the fitness function into: the fitness for the INTEL rule base, and the fitness for the weapons. The fitness for the INTEL rule base included the total INTEL bought by an individual, and an added reward for when the INTEL bought was used to purchase a weapon by the weapon rule base. The weapon



(a) Population A (b) Population B

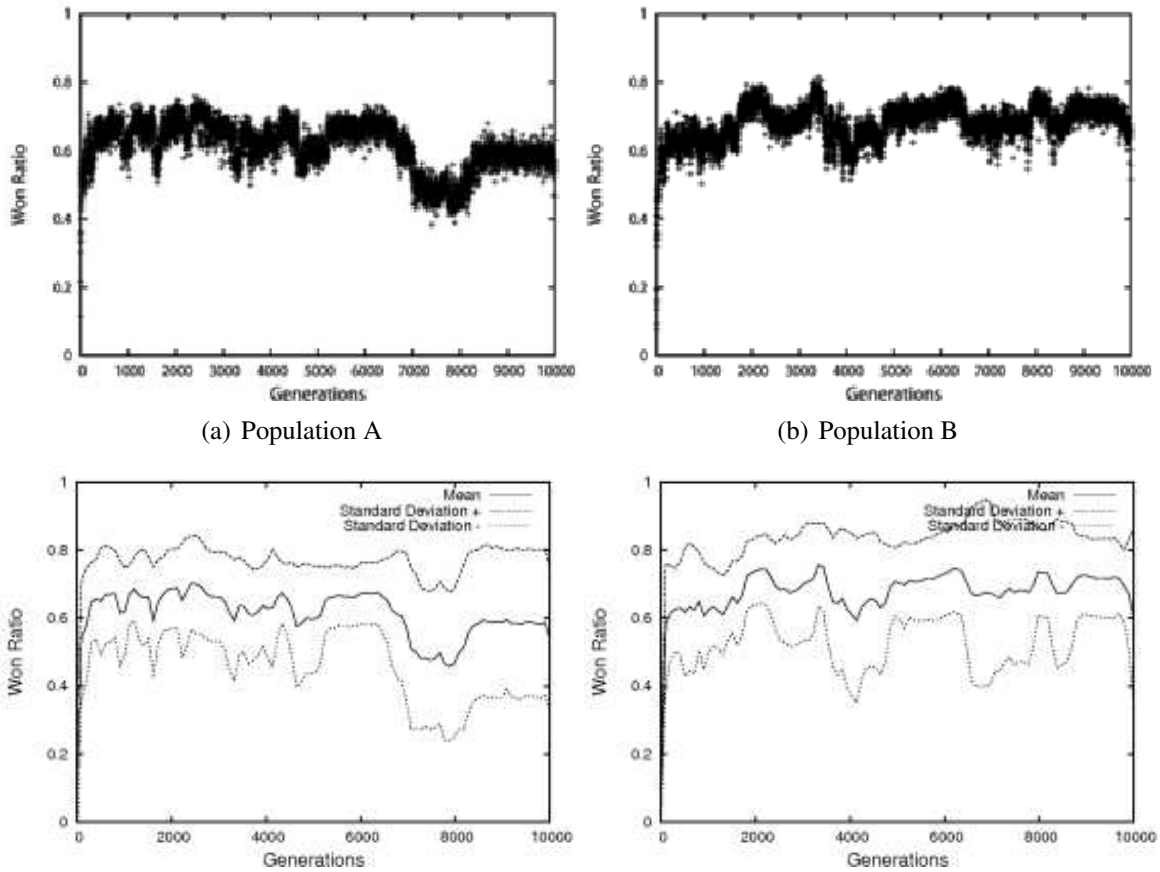
(c) Population A (d) Population B

Figure 7.4: Success ratio against the 'intelligent' expert with no expert inserted into the populations

fitness was adjusted to reward each time a weapon was bought. These finesses were then combined and added to the total fitness of the individual – calculated as the won ratio over all games played.

The results of this experiment against the expert can be seen in figure 7.6. The results show a slight overall improvement in the performance against the 'intelligent' benchmarking expert. The individual runs using the new fitness function showed more individuals that were not buying INTEL than in previous experiments. However, of the runs where the leading individuals were buying INTEL, these individuals were performing much better against the static expert then previously.

Figure 7.7 shows the same experiment re-run without the expert inserted into the populations. Once again we can see a slight improvement in the success ratio against the expert.



(a) Population A (b) Population B

(c) Population A (d) Population B

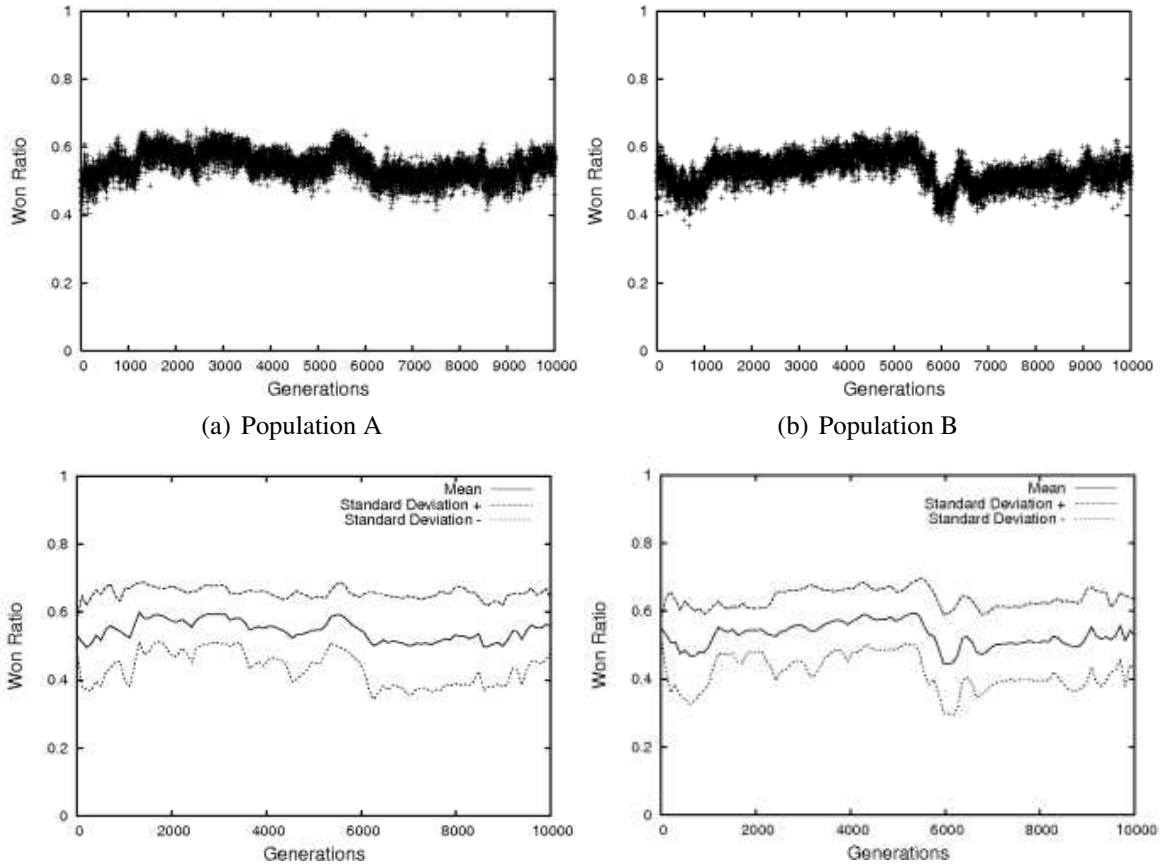
Figure 7.5: Success ratio against the old expert with no expert inserted into the populations

7.5.2 Addition of Counter Intelligence

The next step in the experiment was to assess the CI as described in section 7.4.3. The experiments were performed by running the same system as for the INTEL experiments. Each experiment had the same evolutionary parameters, and the system was run each time for 50,000 generations unless otherwise noted.

This section contains results graphs that are slightly different from previous experiments. Both populations are plotted on the same graph, so that differences in evolution can be easily seen. As mentioned previously, the experiments were run 10 times, and the average of the runs for each population was calculated for plotting purposes. For the graphs in this section, this average was smoothed using the Bezier curves function to approximate the data trend.

For the first experiment, we wanted to observe the impact of including the CI component. To truly observe the difference, we ran the coevolution with one population allowed



(a) Population A (b) Population B

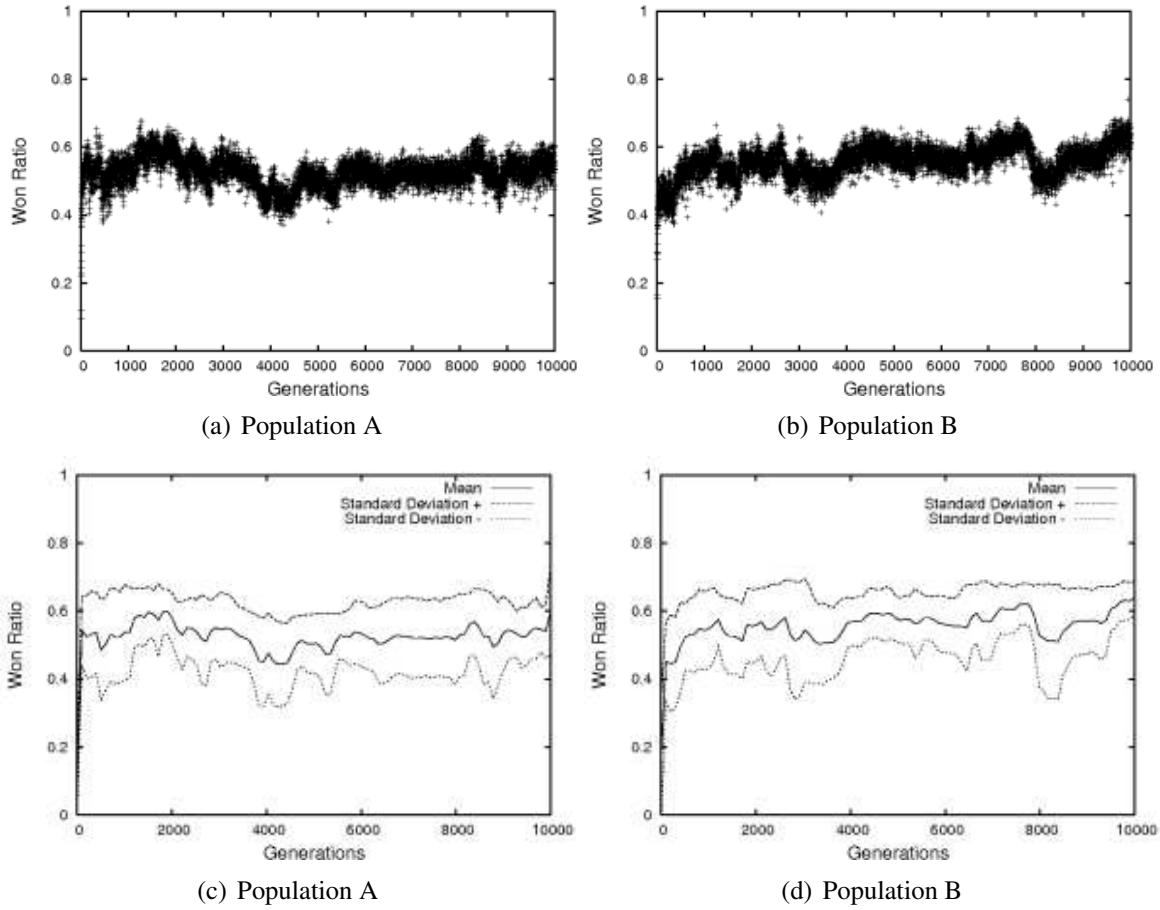
(c) Population A (d) Population B

Figure 7.6: Success ratio against the 'intelligent' expert with new fitness function with the expert inserted into the population

development of CI rules, and the other population restricted to weapon and INTEL rules only. The baseline was measured with the old expert, and the results can be seen in figure 7.8, with figure 7.9 displaying the populations on the same graph.

The results show that the addition of CI to the system did not have much further impact from the INTEL implementation. However, we noted that the rules being developed were not purchasing much INTEL or CI. To investigate this further, we performed the experiment again, but this time having population A with no INTEL or CI rules, and population B with only INTEL rules. These results can be seen in figure 7.10, with figure7.11 showing both populations on the same graph.

The same trends in results were seen against the 'intelligent' expert. These results proved very interesting, as the population with no INTEL or CI rules (popA) dominated over the population with INTEL rules (popB). It would appear that the addition of INTEL rules hampered the system somewhat, but the further addition of CI did not create addi-



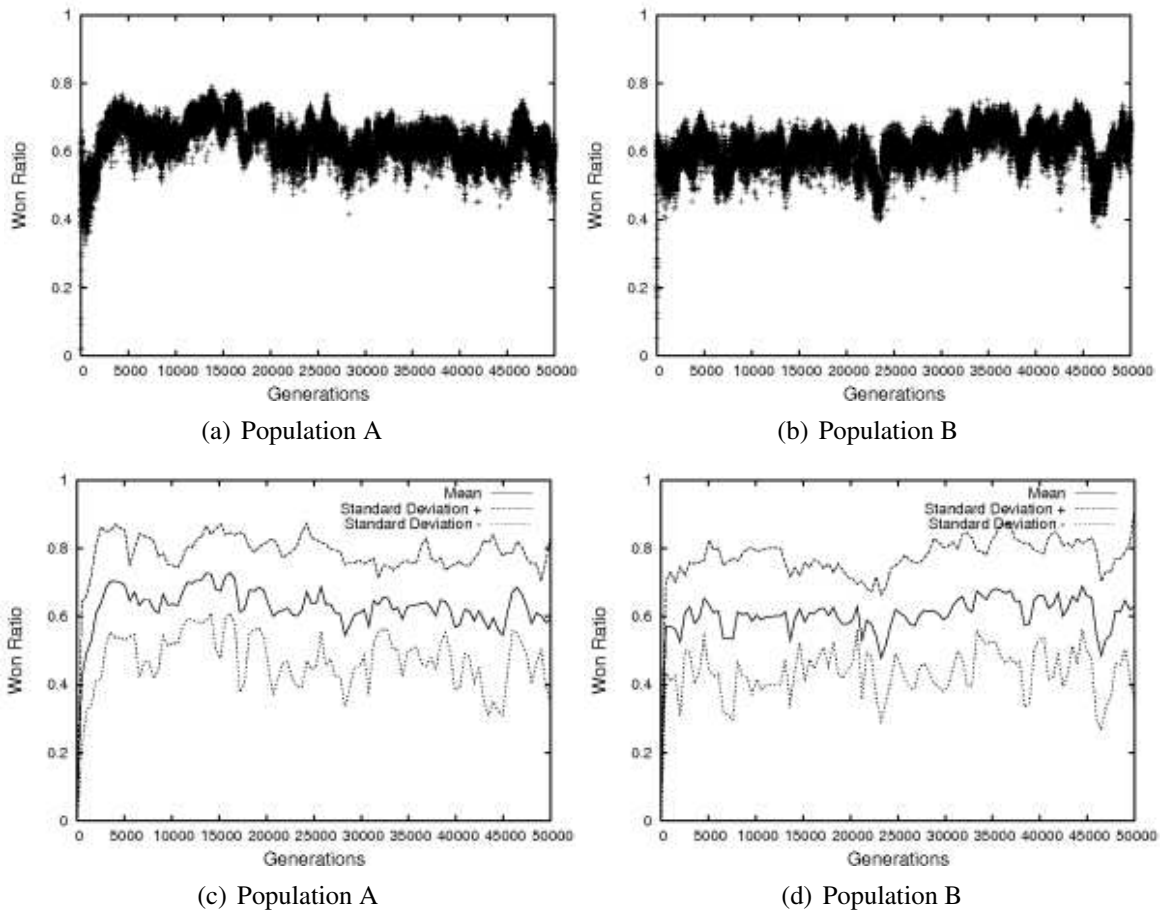
(a) Population A (b) Population B

(c) Population A (d) Population B

Figure 7.7: Success ratio against the 'intelligent' expert with new fitness function without the expert inserted into the population

tional drag, which meant that our first instinctive hypothesis that the extra rule bases were affecting the evolution was incorrect. To investigate the reasons for this, we devised two hypotheses. One was that the additional INTEL rules were causing the fitness of the individuals to be lowered due to the mechanism we used to apply the Ockham's razor principle. The second was the cost of purchasing INTEL was deemed an unnecessary hindrance for the coevolutionary system.

Changing the way we applied the Ockham's razor principle to the INTEL rule base tested the first hypothesis. Each individual has three rules bases, one each for the weapon, INTEL and CI rules. Each of these rule bases then have the Ockham's razor principle applied to them, penalizing the fitness by the number of rules and inputs used. We changed this to allow the first five rules in the INTEL rule base to exist without penalty. The same experiment was then run again, but with this new mechanism in place. The results showing the bezier smoothed averages for both populations can be seen in figure 7.12, with both



(a) Population A (b) Population B

(c) Population A (d) Population B

Figure 7.8: Success ratio with population A only allowed weapon and INTEL rules, and population B also allowed CI rules, measured against the old baseline expert

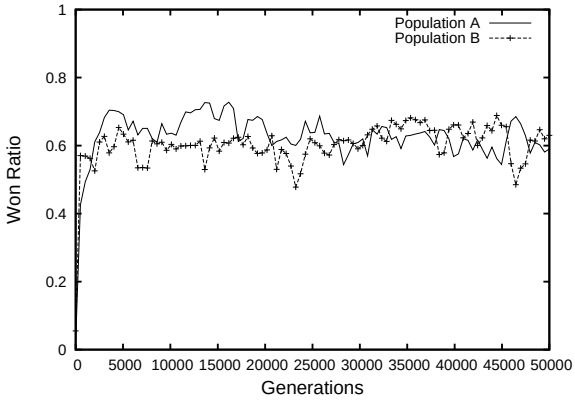
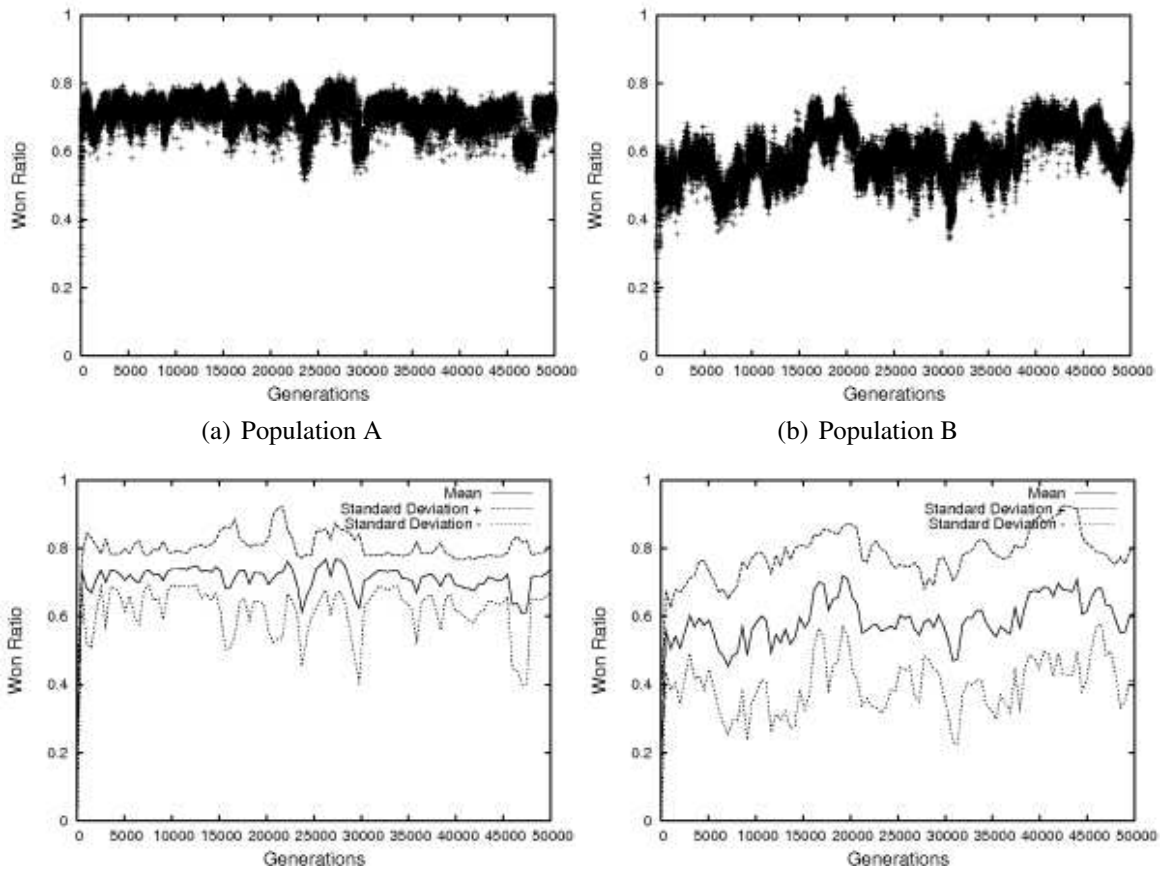


Figure 7.9: Success ratio with population A only allowed weapon and INTEL rules, and population B also allowed CI rules, measured against the old baseline expert.



(a) Population A (b) Population B

(c) Population A (d) Population B

Figure 7.10: Success ratio with population A only allowed weapon rules, and population B also allowed INTEL rules, measured against the old baseline expert

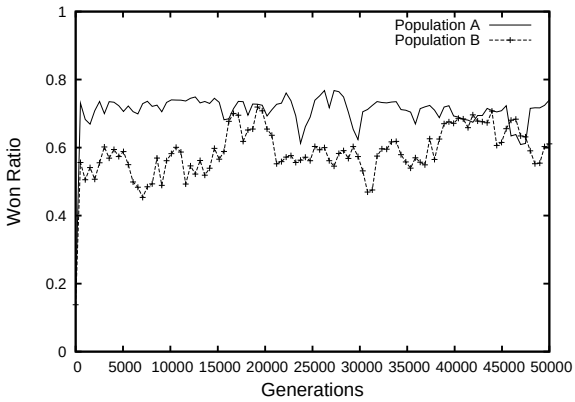
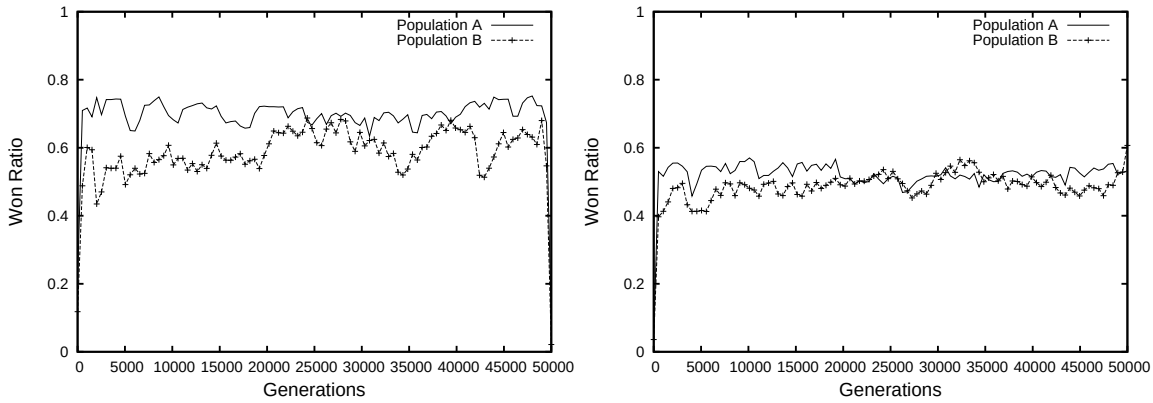


Figure 7.11: Success ratio for both populations with population A only allowed weapon rules, and population B also allowed INTEL rules, measured against the old baseline expert



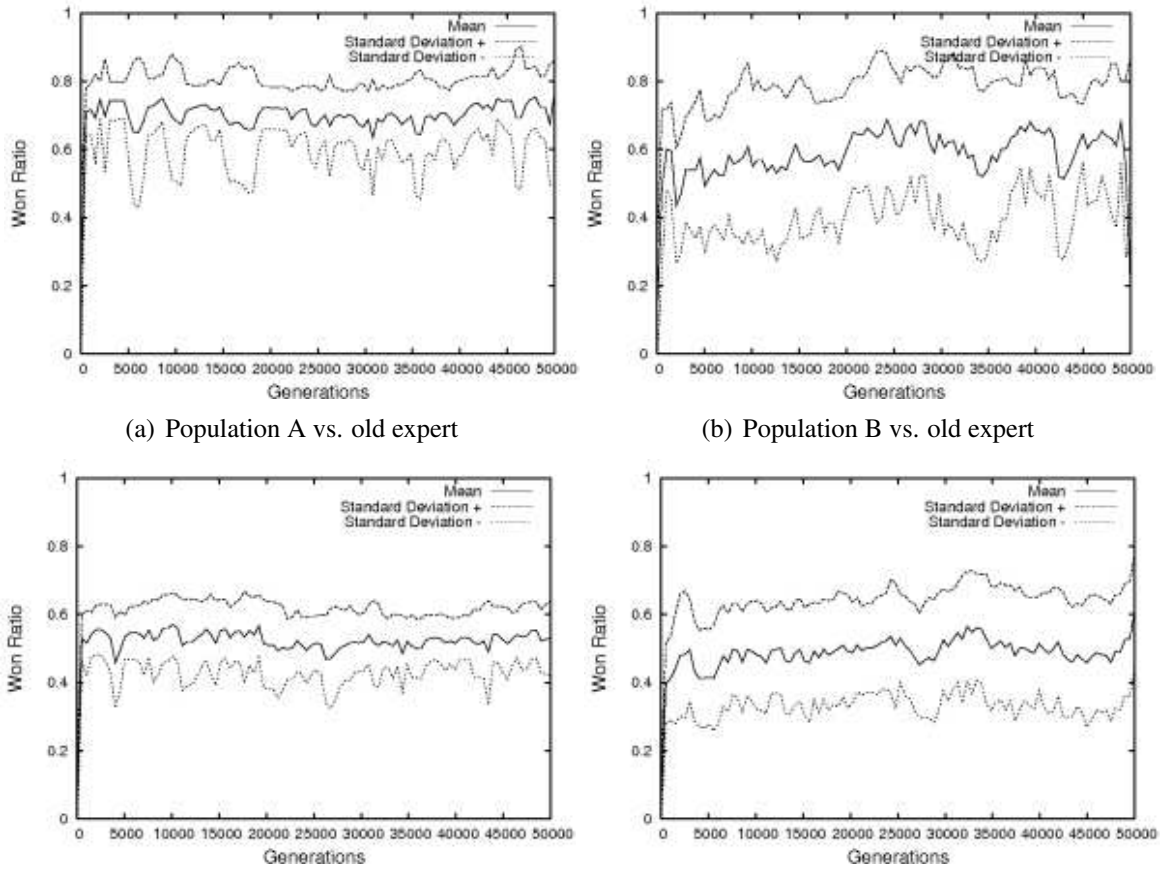
(a) vs. old expert (b) vs. 'intelligent' expert

Figure 7.12: Success ratio for both populations with no penalty to the first five INTEL rules. Population A only allowed weapon rules, and population B also allowed INTEL rules.

experiments shown. The mean and standard deviation results for each population in both experiments can be seen in figure 7.13.

Once again, the results showed that when using the old static expert, the population without INTEL or CI (popA) dominated over the population with INTEL rules (popB). However, the results against the 'intelligent' static expert were different, with a much closer average. It would appear that against the baseline that used INTEL rules to decide on strategy, population B was able to keep up with population A once the rule penalty was restricted.

The next experiment was performed to test the second hypothesis, that the cost of the INTEL played a part in the dominance of the population without INTEL rules. It was thought that the coevolutionary system was trying to find a good strategy by relying only on the weapons, regardless of what the opposition was doing. The purchase of INTEL just reduced the amount of budget that could be spent on weapons, and after all, it is the weapons that are needed to win the game. To test this, we dropped the price of the INTEL purchases. Previously the starting cost for each of the categories of INTEL had been between 10-50 (with a starting budget of about 8,000). This cost then grew an average of 0.05% for each year of game-play (as did the budget). For this experiment we dropped all starting costs for the INTEL down to 10, and the growth to 0.03. We then ran experiments to test the hypothesis. The results of these can be seen in figure 7.14, showing both populations in the same graph for each experiment. The mean and standard deviation results for all populations can be seen in figure 7.14 These results depict two different experiments, both against the 'intelligent' expert. The first experiment had population A allowed no INTEL or CI, and population B allowed INTEL. The second experiment then depicts population A



(a) Population A vs. old expert

(b) Population B vs. old expert

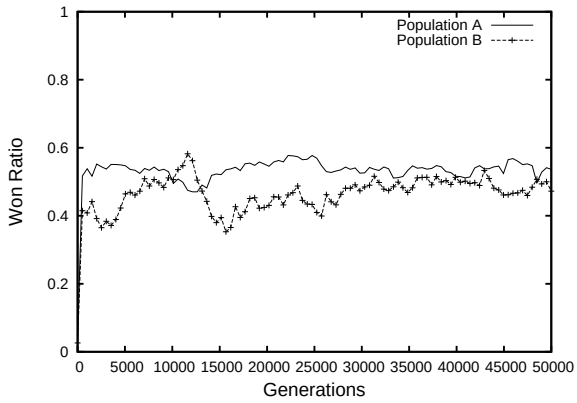
(c) Population A vs. 'intelligent' expert

(d) Population B vs. 'intelligent' expert

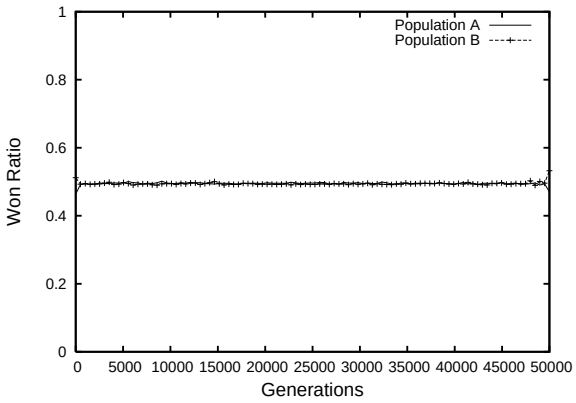
Figure 7.13: Mean and standard deviation for each population with no penalty to the first five INTEL rules. Population A only allowed weapon rules, and population B also allowed INTEL rules.

allowed INTEL, and population B allowed INTEL and CI.

The effect of lowering the cost for the experiment in figure 7.14(a) depicts a slight improvement in the use of intelligence. The results from figure 7.14(b) also show a definitive difference. This is also witnessed when looking into the individual rules being evolved, as the majority of rules in population B for the second experiment (b) are using both INTEL and CI effectively. While previously population A had a very slight dominance in experiments run, both populations are now equal in ability. This indicated that our hypothesis that the coevolutionary system deemed the INTEL unnecessary was true, as results improved once costs were dropped.

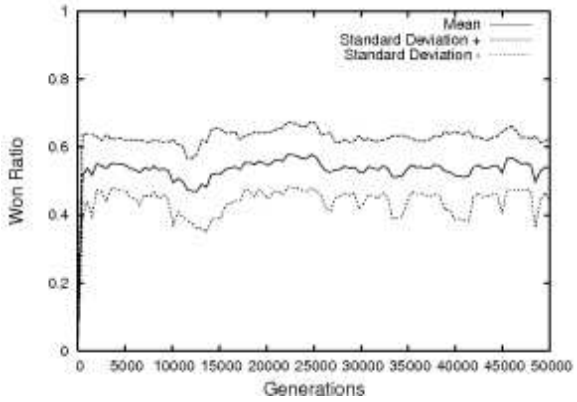


(a) PopA no INTEL or CI, PopB INTEL no CI

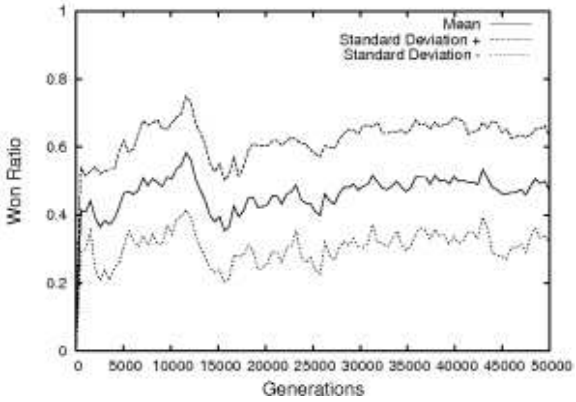


(b) PopA INTEL no CI, PopB INTEL and CI

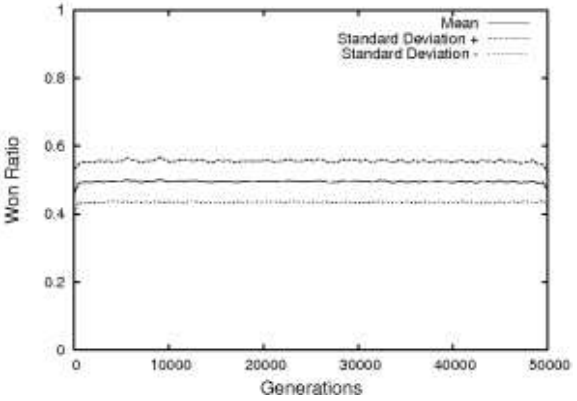
Figure 7.14: Success ratio for the experiments with lowered INTEL costs, against the 'intelligent' expert



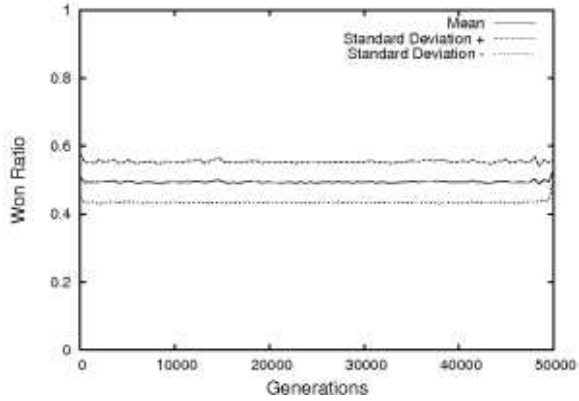
(a) Population A with no INTEL or CI



(b) Population B with INTEL and no CI



(c) Population A with INTEL and no CI



(d) Population B with INTEL and CI

Figure 7.15: Success ratio for the experiments with lowered INTEL costs, against the 'intelligent' expert

7.6 Conclusions and Future Work

Our experiments have shown that it is difficult to prompt the evolutionary process to appreciate the value of knowing what your opponent is doing. It takes the easy route of finding solutions that work without needing the intelligence. Due to it being a coevolutionary approach, it rarely needs to overcome an enemy that effectively uses intelligence. With the changes made in this research however, we have made improvements in the use of intelligence by the system, and have succeeded in developing a coevolutionary system that evolves rules that now take into account the enemy's movements.

By incorporating our "INTEL" expert into the system, we determined that an enemy that *does* effectively use the intelligence dominates the other individuals. Therefore, for an effective individual to be developed from the coevolutionary process, it needs to develop mechanisms to use the INTEL effectively. We have performed a number of different experiments to activate the coevolutionary process to use INTEL, and have had some success in doing so. There is still a degree for improvement however, and further ways of encouraging the individuals to make use of the intelligence they are purchasing should be examined.

Lastly, TEMPO requires several modifications before it can support S&R training. Logisticians must now prepare responsive and adaptive logistic support plans to meet the commander's intent [66]. One challenge is to modify the fuzzy rules, and their evolution, to specifically support the commander's intent. Something that TEMPO currently cannot do. The first step in this process is to set the initial stockage levels, in terms of type and quantity of equipment to support the commander's initial mission. Fuzzy rules must adapt as needed to support changes in the commander's intent as the mission evolves prior to the start of war. INTEL information plays a key role here as well. Indeed, some fuzzy rules may be purged if the commander's intent changes or INTEL information renders rules useless. Cognitive decision support tools must be also be designed to show risk and to help managers develop different logistic courses of action, especially in support of predicted or anticipated logistic support needs. Finally, fuzzy rules must be initialized and evolved in ways that exploits the logistics knowledge base and lessons learned.

We have made some very important steps in improving the INTEL and CI for the TEMPO game, and have had success doing so. However, for the game and computer player to be truly challenging for strategy creation in this area, more work is needed.

The next chapter provides the finale for our research. We combined the previous implementation mechanisms, into developing a system that adapts to a single human player. We describe this system, and the user study tests performed with it.

Chapter 8

Adapting to Human game-play

No matter how good a computer player is, given enough time human players may learn to adapt to the strategy used, and routinely defeat it. A challenging task is to mimic this human ability, and create a computer player that can adapt to its opposition's strategy. Having such a computer player provides a challenge that is ongoing. Additionally, a computer player that adapts specifically to an individual human provides a more personal and tailored game-play experience. To address this need we have investigated the creation of such a computer player. By creating a computer player that changes its strategy with influence from the human strategy, we have shown that the holy grail of gaming – an individually tailored gaming experience, is indeed possible.

8.1 Introduction

A common method of representing a computer player is by a static strategy for game-play. The representation of the strategy could be a neural network, a set of *IF – THEN* rules (fuzzy or not), decision trees or many other means. Regardless of the representation, the use of a static strategy results in a computer player that becomes obsolete once the human player adapts to it. When a computer player ceases to challenge the human player, it is no longer fun to play against. Thus, a game-play experience that is unique depending on the circumstance, and the human game strategy used, has been of significant importance in recent years. Games such as Fable (Lionhead Studios) and Star Wars: Knights of the Old Republic (BioWare), which change the game scenario dependant on the choices the player makes (to be a good, or evil character) have been very successful. The 'choose your own adventure' style has proven to be a lucrative venture, and adaptive adversaries are key to continued success. It seems that the ability for a computer player to adapt to an individual human player is the holy grail of game-play.

Additionally, the ability to provide an adaptive computer player that tailors itself to the human player has a lot of potential in the training industry. It is now fairly well accepted that playing computer games is beneficial for teaching purposes [17]. The ability to provide a fun and challenging way of learning has clear benefits. The problem lies in finding a way to provide the individualistic training required for each student. Currently this lies in providing a level rating (e.g. easy, normal, and hard) that the player can choose. This system is inadequate however, as the classification of student levels into (normally) 3 levels of expertise has obvious disadvantages. Instead, by adapting to the individual human, the game play is no longer standardized for a wide range of players. As identified by Charles et. al. [27], in relation to adaptation in games:

> "Adaptation as such is strongly connected to learning and we may use it to learn about a player in order to respond to the way they are playing, for example by adjusting a computer opponent's strategy so as to present a more appropriate challenge level."

By providing an experience that is tailored at a specific human player, the player can experience a game that simultaneously gains in difficulty as their experience and skill in the game increases.

When the human player starts off playing the game, the strategy creation for the game is fairly weak. The human player then starts to increase in strength as understanding of the tactics increase, and more complex strategies are developed. The same can be said for the coevolutionary algorithm. The initial generations of a coevolutionary computer player create very basic and not overly intelligent rule development. Then, as each game is played, it has a chance to encompass and counter the opposition's strategy of game-play and create better rules. We wanted to use this potential of the coevolutionary system to create a computer player that coevolves to adapt to a human player relative to the human's capability. By including the human strategy into the coevolutionary system, the system can evolve strategies that have adapted to the human strategies. As the human gains more experience in playing the game, so does the computer player.

To achieve the goal of creating an adaptive computer player, we needed to create a way of including the human strategy in a form the coevolutionary system could understand and use. This essentially involved creating a method of reverse engineering the human strategy from the outcome of the game-play. We decided to create an iterative system that would record the data (the human choices and the game environment) from each game played, and use that data to create a model of the human. The model would then be in a form that could be included in the coevolutionary process. To create the model for our

coevolutionary system, it would need to be of the same fuzzy logic rule base structure as the other individuals. The human model is then added as a supplementary individual to the coevolutionary populations, so that the coevolution can evolve with, and against the model.

By reverse engineering the human's strategy into a set of rules and adding the model created to the coevolutionary system, we are able to influence the coevolutionary process. At the beginning of the human's learning progress, he or she will not have any experience in playing the game, and are likely to have minimal strategy development. This stage is probably the most difficult to reverse engineer, as the human is more random in strategic choices. Consider the idea of beginner's luck in games such as poker; expert players can have trouble reading beginners, as the beginner makes seemingly random choices due to inexperience. However, even the very general (and probably not effective) rules reverse engineered at this stage have a chance of affecting the coevolution. It is unlikely that the human rules will be considered the best individual in the population for elitism, but they still have a chance of effecting the next generation through selection and crossover.

As the human starts to gain experience in the game, he or she develops 'winning' strategies. It is likely that the human will then repeat the same or similar strategies over concurrent games if the strategy continues to work. It is here that the adaptive coevolutionary system really comes into play. As the humans form a clearer pattern for reverse engineering, the rules being modelled and added to the coevolutionary system have a greater impact. Now the coevolution can act to directly overcome the human strategy, and create new strategies that are a greater challenge to the human player. This allows us to develop a system that grows and improves along with the human the computer player is competing against; a tailored system that provides the best challenge for the individual human.

This chapter discusses the issues involved with creating a computer player using coevolution that can adapt to humans, and the methods we used to create this system. We begin with a brief background discussion on the area and the difficulties involved. We then discuss the mechanism used to create our adaptive computer player. After implementing this system, we ran a user study to observe effectiveness of the system, and the way human players interact with the computer player. The results of this user study are provided, along with discussion and analysis. We conclude the chapter by discussing the findings, and the areas of research that have been identified for future work.

8.2 Coevolving with Humans

There has been very little research regarding the adaptation of a coevolved computer player to a human player. The main research performed has been conducted by Louis et. al. [60–

63] and Ponsen [79–81] (see section 4.5 for details), but there has currently been no research on the creation of new rules that adapt to a specific human player, during game-play.

It is commonly known that playing a static coevolved player against the same human repeatedly, allows the human to determine a counter-strategy that is dominant over the static computer player. The first time the human plays against the strategy however, it is unknown and could possibly be difficult to beat. The question then arises as to whether the need for adaptation is indeed necessary. Could we not just continue the coevolutionary process, and pick different individuals to play against the human each time? There could well be enough difference in strategy represented through the coevolutionary process itself to provide a challenge for each new game played.

While there has not been much recorded research on this topic, intuitively it would seem that a similar occurrence to the static scenario would eventuate. This reasoning is based on observation through our previous research, where the coevolution reached a plateau. This plateau is visible through the baseline measurement technique used and analysis of evolved rules, where the results show only small change in the best individuals over time. Even though it is constantly changing and undulating, the evolution does not tend to make great leaps in development. Thus, even though the human player would be playing a different player each time, the strategies being developed by the player are similar in strength and strategy to previous ones, and the human player could learn to overcome them. By incorporating the human model, a possibly new 'best' individual is included in the population, and new strategies to beat it are created. Thus, allowing the evolving individuals to directly counter the human strategy making and provide a greater individual challenge. In teaching terms this is a great advantage.

There are a number of ways that the human models could be used in the coevolutionary process. Originally we thought of having a separate human population, similar to the memory population in previous research. However, the extra processing time required for selecting and evaluating a separate population was deemed excessive for the purpose. Additionally, if evaluation were the only influence the human model had on the populations, poorer human strategies would have little to no effect when they are constantly beaten. Including the human models into the populations has a direct effect on the individuals being created (through selection and crossover).

To include the human in the process, we needed to find a way to coevolve against the human model and the other randomly evolving players. This allows the system to create players that are still finding randomly evolving strategies that can take into account, and counter, the human player's actions. By including the human model in the coevolutionary process, when a new 'best' individual is chosen from the system to play against a human

player, this individual has been able to incorporate and counter the stronger elements of the human strategies. Now the individual can provide a new challenge for the human.

8.3 Representing Human Strategies

Representing (modelling) humans is a research field in itself and can be very difficult to do. To minimize this issue, we chose to very loosely reverse engineer the human choices as a model of the strategies used. The model used would also need to be of the same format as the coevolutionary individuals in the TEMPO system. Doing this allows the human model to be directly inserted into the coevolutionary system for the process described above.

The reverse engineering of the human works as follows. When a human plays a game against the computer player, the data of the game is recorded. This data includes the choices the human made, and the environmental data for each game year. The data is then used in an evolutionary system to find individuals that model the human by mimicking the human choices. To evolve the human model, individuals represented in the same way as the co-evolved individuals are randomly initialized. Each individual then plays the exact same game as the human, against the same computer player as the human played. The individual is evaluated as the difference in outcome and allocated resources to what the human achieved. The closer the individual comes to the same results as the human, the better the fitness.

Additionally we added changeable constant weightings to the evaluation. The weightings were applied to the differences in the outcome, weapons bought and intelligence/counter intelligence bought between the individuals and the human value. By adding weights, we are able to sway the evaluation importance of each of the parameters for the purpose of creating more realistic rules. For example, it might be that getting a closer outcome (total offensive utils at the end of the game) to the human was more important than creating rules that allocated the resources in the same manner as the human, and vice versa. Hence the evaluation function has the following evolutionary variables:

1. *human_NetUtils* – the total net offensive utils for the human player at the end of their game.

2. $individual_i$*_NetUtils* – the total net offensive utils the individual scored (when playing the same game).

3. *Years* – the total number of years the game played for before war broke out.

4. *human_IntelChoice*, *human_OptChoice*, and *human_BgtChoice* – the data arrays of human allocations made for intelligence, weapons operated and weapons bought respectively for the year.

5. $individual_i$*_IntelChoice*, $individual_i$*_OptChoice*, and $individual_i$*_BgtChoice* – the allocations the individual made for the year corresponding to the human allocations.

6. *NetUtilsWeight*, *IntelWeight* and *WeapWeight* – the constant weights applied to the different evaluation areas as described.

The described evaluation function *eval* is implemented as

$$\begin{aligned} eval(individual_i) = {} & abs(human\_NetUtils - individual_i\_NetUtils)\, NetUtilsWeight + \\ & \sum_{t=1}^{Years}((abs(human\_IntelChoice_t - individual_i\_intelChoice_t)\, IntelWeight) + \\ & (abs(human\_OptChoice_t - individual_i\_OptChoice_t)\, WeapWeight) + \\ & (abs(human\_BgtChoice_t - individual_i\_boughtChoice_t)\, WeapWeight)). \end{aligned}$$

We experimented with different values for the constants, with different preference weights for the resources and outcome. For the final process, the outcome constant was assigned the highest preference with a weight of five, followed by the weapon constant with a weight of three. The INTEL/CI constant was given a weighting of one.

We also experimented with selection operators, and used ranked selection with elitism in the final process. All individuals in the population were used as potential parents. The variation operators used were two point crossover and mutation, where chosen genes were replaced with a random value. Crossover was run on the parent population first, followed by mutation. To avoid premature convergence on a suboptimal solution, we also forced the individuals to have unique genotype.

After much experimentation and manual changing of the parameters to determine a good result, the final evolutionary parameters chosen were as follows. The process ran for 150 generations, with a population of 100 individuals. An elitism ratio of 5% was used, with a 50% crossover ratio, and a 10% mutation ratio. No rule penalty was applied to minimize the rules used in the rule bases, as this seemed to occur naturally.

The rules evolved using this method give a rough estimation of possible strategies the human used. It by no means represents the human strategy exactly, which is in many ways a good thing. Our task is not to try and create an optimized computer player against a human player, but to create a computer player that is *challenging* for the human player. Even if it is only evolving against a rough estimate of the human player, for a single gameplay situation, the evolutionary process is still given the opportunity to counter the human strategies.

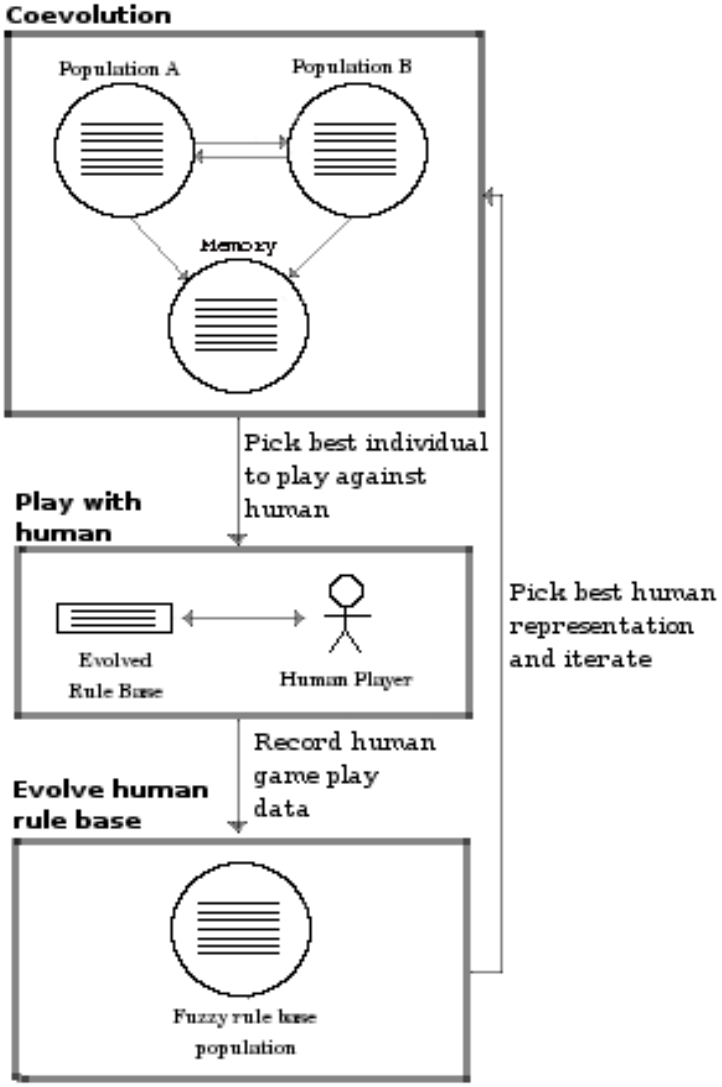


Figure 8.1: The human adaptive coevolutionary process

8.4 The Human Adaptive Coevolutionary Process

To incorporate all the ideas described above, we needed to develop an entire system for game-play. We named the system the Human Adaptive Coevolutionary Process (HACP).

HACP incorporates the coevolutionary system from previous chapters with a graphical user interface (GUI) to play against human players. This was then combined with the human reverse engineering (modelling) system described in section 8.3. The process flow can be seen in figure 8.1.

The process consists of the following steps:

1. The process begins by coevolving two populations against each other and the mem-

ory population, using the STM and LTM as described in section 6.6.

2. After a set number of generations, the best individual from the currently winning population is chosen and played against the human.

3. The data from this game is then recorded and passed to the evolutionary human modelling system.

4. The modelling system then evolves a rule base that mimics the actions the human made. This system runs for a set number of generations before the best individual is selected.

5. The best individual is then placed into the coevolutionary system, replacing the worst individuals from each population, and the whole process iterates again from the beginning.

To ensure the human model affects the coevolutionary populations, the population size was cut down to 15 individuals. Thus, even if the human model has a weaker fitness than the individuals in the population, it still has a probable affect through selection. The first time the coevolution is run, it runs for 300 generations. This is enough to develop a beginner level player that buys some form of weaponry. The consecutive iterations of coevolution then only have 100 generations to coevolve a new player, which allows reasonable time (generally $<$ 1 minute) in between games with the human.

The number of sample games played against the opposition and memory for evaluation was also cut down to decrease the time taken. For this process, the individual is evaluated by playing $r_1 = 5$ games against the opposition, $r_{2S} = 5$ against the Short Term Memory, and $r_{2L} = 5$ against the Long Term Memory. All the other evolutionary parameters remained the same as for previous experiments. Crossover was applied at rate of 30%, and if crossover was not applied to the individual, mutation was applied for each gene at a rate of 30% with a 10% chance of a large mutation.

The final component of the process is the human modelling system. This is added using the evolutionary process described in section 8.3 above. The human modelling system added on average an extra minute to the entire process.

The GUI was coded in Java and communicated to the C++ code through http sockets. Figure 8.2 shows a screen shot of the GUI mid game.

The environment section shows the current year, the chance of war breaking out at the end of the current year (the *pwar*), the budget for the year, and the amount of the budget left as the user allocates to weapons and INTEL/CI. The previous year's data section shows the total OA, OB, DA and DB utils left over from the previous year (once the opposition's

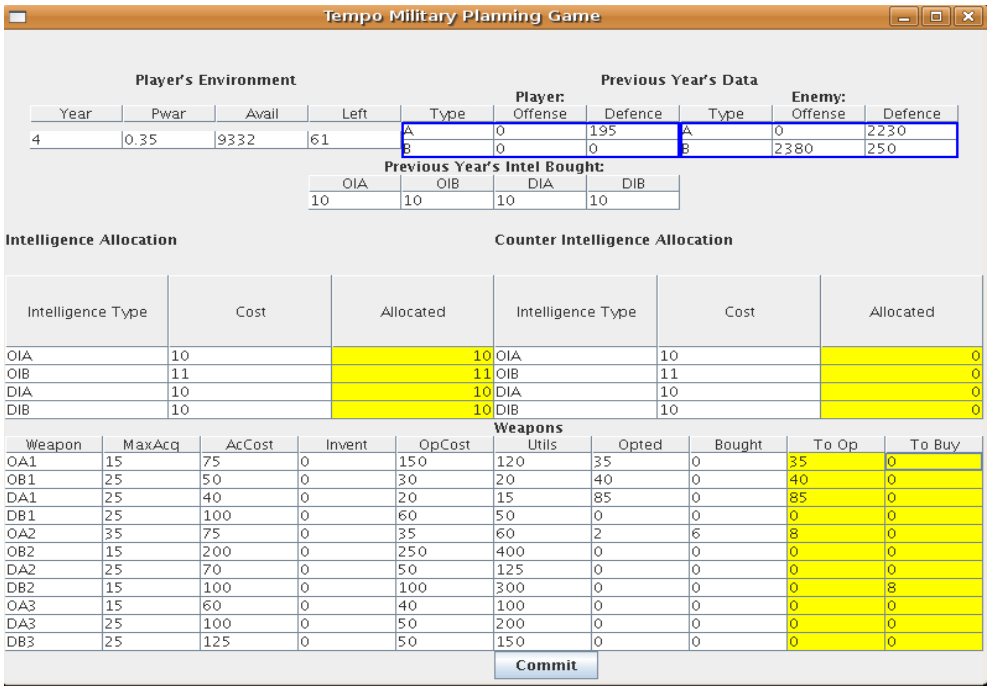


Figure 8.2: Screenshot of the TEMPO GUI

corresponding utils have been subtracted) for both the player, and the opposition. The opposition utils are only shown if INTEL has been purchased, otherwise "UNKNOWN" is displayed. If the opposition has purchased CI, then the value shown in the enemy's previous year's data may be incorrect to some degree as described in 7.4.3. The previous year's INTEL section displays the amount of INTEL bought by the player for the previous year.

The Intelligence Allocation table and Counterintelligence Allocation table both have three columns. Each row in the table represents a different type of INTEL/CI category, with the associated cost. The last column in each table is the user entry field, where the user can enter the budget amount allocated to the category. The cost of INTEL/CI is deliberately low to encourage purchase.

The Weapons table displays all the weapons available for the year. Each row represents a different weapon with the corresponding attributes for the weapon. The first column gives the name of the weapon, consisting of the category (Offensive/Defensive), type (A/B) and number (1,2,3). The second column shows maximum number of weapons that can be acquired (bought) each year. The third column is the cost to purchase a single unit of the weapon. The fourth column represents the inventory for the weapon – the number of weapons given to the player for 'free' at the beginning of the game. The fifth column gives the cost to operate (use) a unit of the weapon for a year. The sixth column gives the

number of utils the weapon has (the power ability of the weapon). The seventh and eighth columns show the weapons that have been opted and bought (respectively) in the previous year. Finally, the ninth and tenth columns are the user input columns to allocate the budget to opt previously bought and opted (or available in inventory) weapons, and purchase new weapons for the coming year.

Once a player has made their allocations, the commit button is pressed and play either continues into the next year, or war breaks out and the game results are displayed.

8.5 User Study

Using the HACP system, we ran a user study to test the effectiveness with humans. The purpose of the user study was to obtain users with no experience of TEMPO, and use the HACP system as a way of training them. We also wanted to test the effectiveness of using a system that adapts to a human, as opposed to a static player or a coevolving player with no knowledge of the human strategy.

To achieve this experiment, we created an application with three consecutive stages. Each stage would involve the human playing 4 consecutive games against a computer player, with the results for each game recorded. We chose 4 games due to time constraints, but ideally more games would be beneficial. The first stage involved running the user against the same static computer player for all 4 games. The static player was previously evolved and consisted of 19 weapon rules, 8 intelligence rules and 8 counter intelligence rules, with the two most active rules as:

1. IF Category IS Offensive and Type IS B
   THEN Evaluation IS medium

2. IF OperationCost IS Very Low
   THEN Evaluation IS high

The intelligence and counter intelligence rules were rarely activated. After playing the 4 games against this static player, the human was then informed that the next stage was about to start.

Stage 2 consisted of the human playing 4 games against the coevolutionary system. The system was first run for 300 generations (with the same evolutionary parameters as described in section 8.4), and the individuals for both populations in the final generation were saved. This was the starting point for all the human players, and the best individual from this coevolutionary run was chosen as the starting individual to play against the human for Stage 2. Once the first game was completed, the coevolutionary system was then

restarted from where it left off, and another 100 generations were run. The best individual from the best population (according to fitness) was then chosen to play against the human, and this process was then iterated for the rest of the games. While the first coevolved player in Stage 2 was the same for each test subject, consequent players were different due to the coevolution.

Stage 3 was conducted in the same manner as Stage 2. However, in this stage, the HACP system was used, and the additional steps of finding a human model and including it in the coevolutionary system were applied.

8.5.1 User study results

The results for Stage 1 are depicted in table 8.1. The table shows the results for each human player, for each game played, against the static computer player. The score is the total net utils for the human player at the end of the game. Positive results show a win, while negative ones are a loss. The number of years the game played for is also recorded for comparison purposes. The final column in the table shows the total number of wins each player had in the stage. The last row in the table gives the average score and years played for each game, and the average games won for Stage 1.

As expected, the results show that the players had an overall average loss for this round. The average loss does however decrease over the progression of the stage, depicting player learning. From analysing the results and questioning the players, we found that the average games played before the players felt confident in their game-play were 4–5 games. We also note that there were some players who developed good strategies from Stage 1, and performed well throughout the stages.

The Stage 2 results are depicted in table 8.2, in the same format as the stage one results. By this stage, most of the players are confident in the game-play and begin to develop some strategies to win. There is a dramatic increase in the average scores for the entire stage, although there is still a slight learning curve in some participants. To note, the first game played against is actually a simpler (but different) player than the static one played in Stage 1, so the number of negative scores tends to show some players are still developing their strategies. By the end of this stage we only have a single player that has not won a game, with the majority of players winning at least two games. The players are beginning to play around with strategies, or have found a 'winning' strategy they continue to use.

The Stage 3 results are shown in table 8.2, and follow the same format as before. The first game has the same player as Stage 2, Game 1 (the same coevolutionary starting point), but this time round the majority of players win. At Stage 3, Game 2, the first round of the

Table 8.1: User Study Stage 1 Results

Stage 1									
	Game 1		Game 2		Game 3		Game 4		
Player No.	Score	Years	Score	Years	Score	Years	Score	Years	Wins
1	-135	4	655	8	0	2	0	3	1
2	-1920	7	1165	2	2480	8	2160	6	3
3	0	3	-4275	8	-2750	4	-50	2	0
4	460	4	-7100	7	2370	8	425	3	3
5	-6661	9	3230	3	-6440	7	-6012	8	1
6	-4340	7	1660	5	0	7	-2000	6	1
7	0	7	-620	5	-1640	5	0	7	0
8	2465	8	2000	5	2610	10	1400	6	4
9	0	7	-1090	5	-99	2	1165	2	1
10	0	8	2713	3	-1290	6	-1550	4	1
11	-1190	6	-3890	7	-1820	7	2730	6	1
12	2260	5	-4820	5	-4640	5	-4940	5	1
Average	**-1029.18**	**6.36**	**-504.73**	**5.27**	**-598.09**	**6.00**	**-157.45**	**4.82**	**1.45**

Table 8.2: User Study Stage 2 Results

Stage 2									
	Game 1		Game 2		Game 3		Game 4		
Player No.	Score	Years	Score	Years	Score	Years	Score	Years	Wins
1	3860	5	-540	4	-495	4	1300	3	2
2	8940	7	4412	8	6589	9	-3004	6	3
3	-2518	7	1200	2	2955	7	-2599	8	2
4	-40	3	6100	7	5039	7	1092	5	3
5	-1155	4	-3358	5	659	6	2940	3	2
6	0	8	3260	6	700	4	0	9	2
7	-1440	3	3070	5	-840	3	0	7	1
8	1065	5	2910	5	615	5	2965	4	4
9	5188	9	1210	7	1278	5	3840	2	4
10	-1236	4	4800	8	289	4	4640	2	3
11	2965	6	4070	5	2604	7	0	3	3
12	-4620	5	-7656	7	-9316	8	-9186	5	0
Average	**1420.82**	**5.55**	**2466.73**	**5.64**	**1763.00**	**5.55**	**1015.82**	**4.73**	**2.64**

Table 8.3: User Study Stage 3 Results

Stage 3									
	Game 1		Game 2		Game 3		Game 4		
Player No.	Score	Years	Score	Years	Score	Years	Score	Years	Wins
1	2785	6	-649	10	-2600	4	0	5	1
2	3960	6	-490	6	-3290	6	2780	6	2
3	-4601	5	-3085	6	730	4	-1899	9	1
4	3765	6	442	9	582	7	2926	8	4
5	47	3	428	3	316	3	-342	4	3
6	-1440	3	3140	4	1640	6	-2090	4	2
7	230	4	1500	6	2148	2	5750	7	4
8	2455	6	3695	7	1019	8	400	3	4
9	4705	8	-586	7	1500	1	110	6	3
10	4995	7	445	7	1500	1	5630	8	4
11	3335	6	2450	6	3210	4	1740	5	4
12	-6418	6	-4894	7	-1180	3	-5050	6	0
Average	**1839.64**	**5.45**	**662.73**	**6.45**	**614.09**	**4.18**	**1364.09**	**5.91**	**2.91**

HACP process has been performed, with the coevolution taking into account the human game-play. The Game 2 results for this stage show an overall success with the HACP system. A large amount of the players who won in Stage 2, Game 2 either lost the game or had the amount of utils won by greatly decreased. The 3rd game in Stage 3 shows this same trend happening with some players, while others succeed with a new winning strategy. The same thing is seen in Stage 3, Game 4.

The other notable thing from Stage 3 is with player 12, who was the only player that had difficulty learning the game. The results from this stage show that even though the losses were continuing, they were not by as much. It appears that even in this case the game-play was slowly helping. Whether this is due to a longer learning curve or the HACP system however is questionable.

8.6 Conclusions and future work

The total wins for Stage 3 were slightly higher than Stage 2, which was expected due to the player learning curve continuing through into Stage 2. However, the results clearly show that there was not a large jump in results between Stage 2 and 3, and that the system did in fact continue to challenge and teach people. This conclusion was reinforced through the informal verbal feedback process at the end of the user study. The majority of the users stated that they found Stage 3 more challenging than Stage 2 when asked if they noticed

any difference between the two stages. There were also some users who specifically stated that strategies they had developed and used in Stage 1 and 2 needed re-evaluation in Stage 3 when the computer player overcame their strategy. The human players were then forced to think of different strategies.

There were also players who were able to dominate the game from Stage 1. This is likely due to the choices made to cater for average users. For example, to allow for an easier starting point, we only evolved the starting computer player for 300 generations. With a population of 15 individuals this only allows creation of a very simplistic player. Additionally, the short number of games played meant that the HACP system in Stage 3 did not have much time to evolve against more complex human strategies. The human modelling system also struggled more to create these strategies. Even with these deficiencies however, the human players with stronger strategies noted that they thought Stage 3 was beginning to increase in difficulty, and future research should test the impact of additional games.

While the HACP system does seem to have considerable benefit for the majority of users, the extreme ends of the learning curve do not seem to benefit as much. One way to address this problem is to adapt the evolutionary parameters to the capability of the human player. The idea here is to adaptively restrict or encourage the coevolutionary process to match the proficiency of the individual human player.

If a human player is doing particularly poorly and losing every game, he or she soon feels discouraged and stops playing. Hampering the success of the coevolutionary process can address this issue, be it through a reduction in generations or population size, or applying an additional weight to the evaluation function. The weight could change depending on how much the individual won by against the human player, with the result of allowing 'lesser' individuals to obtain a higher fitness. The human players who continually win against the computer player also need a mechanism to make the play more interesting. The same concept could be applied for these human players, but in reverse. When a computer player loses to a human, the generations and/or population size could adaptively *increase*. The evaluation weighting could be applied to progressively increase the loss penalty for successive losses. There are also many other possibilities that could be applied.

The other area we would like to improve is the development of the human model. Each time a new game is played, a new model is reverse engineered with no reference to previous games. This process wastes a vast amount of data that could be used to refine the model, as usually humans build strategies from previous game-play. One mechanism we have thought of applying is to use the results of the previous year to influence the fitness of the current models being evolved. If a model has some similar tactics to the previous year, then it is rewarded. This concept also has inadequacies however, as it only forms a single link to the

previous year, and does not take into account long term strategy.

Overall, we had a great deal of success with the implementation of the HACP system, with a number of users stating that they had fun trying to beat the computer player. Being able to learn new and better strategies of resource allocation on their own time allows students individual training that caters for their own needs. The HACP system provides a good mechanism for applying such training. The HACP system also has benefits for creation of computer players in many commercial games (such as turn based strategy games), where each turn the strategy could be updated to incorporate the human player's strategy. The applications are many, and this research is just the beginning. We have shown that it can work, and making it work in other games is an exciting challenge.

Chapter 9

Conclusions and Future Work

The research presented throughout this thesis has addressed the need for an adaptive computer player for resource allocation games. We conclude the thesis by reiterating the hypothesis and objectives, and how we achieved these goals. We also discuss inadequacies that remain in the system, and possible future directions of research.

As stated at the beginning of the thesis, our hypothesis was that *"Coevolutionary algorithms are an effective mechanism for the creation of a computer player for strategic decision-making games"*. In particular we defined our three main objectives as:

1. The computer player must be able to allocate resources effectively in the game of TEMPO, for the purpose of being competitive against human players.

2. The computer player and the TEMPO game should provide the necessary scenario for the human player to learn the task of resource allocation.

3. The challenge given by the computer player should be tailored to the individual human's ability level.

We now provide a summary of how each of these objectives was met through the research. The first objective was addressed partially by the research of Johnson et. al. [55]. Deficiencies were found however in the efficiency of the computer player. To increase the effectiveness of the computer player, we included the use of a memory in the coevolutionary algorithm. Additionally, we researched the mechanism of selection from the memory.

We developed a novel selection technique based on the human short and long term memory structure. The technique was used to select individuals from the memory that were used to evaluate the coevolutionary individuals. We found that the different forms of selection pressure applied through probability distribution selection had an impact on the effectiveness of the memory. In particular, we determined that high selection pressure

on the most recent individuals in the memory worked the best. This pressure was applied through a selection window of the top ten individuals in the memory, which we called the short term memory. Selection was then also applied to the entire memory, which we called the long term memory, through a linear time distribution.

The use of the short and long term memory provided very good results, and further research was conducted into different ways to use the long term memory. We had success with a ranked selection technique, called gladiator selection, which out-performed the linear long term selection technique. We also attempted to cluster the long term memory into groups representing rules for the current situation in the game. These clusters were then used for selection, dependant on the yearly status of the game. The clusters required a different mechanism for coevolving the individuals, as instead of playing a complete game against an opponent for evaluation, the individual played a complete game against changing opponents for evaluation. This technique provided some interesting results, and more research is needed for the full benefit to be analysed.

The use of memory allowed the development of strong static rule bases. When played against human players, the computer player was now much more competitive, and harder to beat. There were still difficulties however in developing computer players that were efficiently using the intelligence (INTEL) and counter intelligence (CI) components of the game. It was with this in mind that we decided to re-evaluate the use of INTEL and CI in the TEMPO game. In collaboration with Garrison Greenwood, we identified inadequacies to the game that needed to be addressed. These inadequacies directly affected the second objective of our research, as we found the TEMPO game was not currently providing the scenario necessary for its purpose.

To address this deficiency, we changed the INTEL and CI from a boolean allocation into a percentage allocation. The previous mechanism allowed players to either buy or not buy INTEL and CI. The new mechanism allowed players to buy a degree of INTEL and CI up to a specified amount. We also divided the INTEL and CI into the category/type units (OIA, DIA, OIB and DIB) instead of the previous Offensive and Defensive categories, with CI in a separate category. We presented a novel way of implementing this new INTEL/CI mechanism, using a probability distribution function to map the amount of INTEL bought to an appropriate degree of information given. The CI was also implemented using the probability distribution, spreading the function to a degree.

The new INTEL/CI mechanism was a more realistic scenario for resource allocation, and additionally the computer players started to develop INTEL rules that purchased INTEL for the first time. The use of INTEL by the computer players was however still lacking, and further experimentation was carried out to improve on this. While we were able to

make small improvements to the INTEL rule bases being developed, there is still a lot of improvement to be made. It may be possible to achieve this improvement through a mechanism to link the separate rule bases in the evaluation. The link could be achieved through a hierarchy mechanism, or possibly some form of multi-objective task with evolved weights determining the importance of weapons vs. INTEL.

Further improvements are also possible for the INTEL/CI in the game of TEMPO. While the breakdown into the separate category/type units is more realistic, further breakdown is necessary for true usefulness. For example, consider a situation where CI is obscuring the information being given about the opposition's utils. It is now difficult to conclude an appropriate action when only a single INTEL source is available, and realistically INTEL should never be believed until it is corroborated. This should be performed by using information from multiple, independent sources and then merging the data, looking for corroborating evidence to support an analytical conclusion. This process, called information fusion, adds credibility to the analytical conclusions; the more corroboration, the higher the believability [32]. In practice INTEL information comes from a variety of sources including human intelligence (where data is collected through interpersonal contacts between individuals), signal intelligence (where data is collected by way of intercepted communications) or data transmissions imagery intelligence (where data is collected from space-based electro-optical, radar or infra-red images). For each type of INTEL activity there is also a corresponding CI activity to counteract it. It is this breakdown in the INTEL/CI mechanism that we would also like to represent in TEMPO.

While the research into INTEL and CI for the TEMPO game provided many insights and improvements, there is still much to do. In this regard, the second objective was met to a degree. The computer players and the TEMPO game do create a good scenario for the task of learning resource allocation, but more research is required for the full potential to be met.

The third objective was addressed by our development of the Human Adaptive Coevolutionary Process (HACP). This combined the work done with memory to improve the static rules developed, and the additions to the INTEL/CI mechanism, and created an additional step to adapt to a human player. The additional step used data recorded from a game with a human player to reverse engineer a model of the human's strategy. This strategy was then placed in the coevolutionary system to continue evolving against. The next game played against the human would use a static strategy that had been coevolved with the human model, thus providing an adaptive computer player that tailors itself to the human.

Through a user study, we were able to test the usefulness of HACP system against human players. We found a definite advantage of the HACP to provide a challenge for

human players. The human players began game-play against a static unchanging computer player, and then played against a dynamic computer player with no adaptation, followed by the adaptive HACP computer player. There were distinct advantages noticeable when playing against the HACP computer player, which were also noted by the human players.

While we have met the third objective with the HACP system, there is still room for further investigation, especially for human players at the extreme ends of the learning curve. It is possible however that the HACP system could be tailored for these players, through an adaptive process aimed at the specific skill level of the player. The possibilities for further development and improvement are an exciting area for future research.

There are many possible future directions for research into a computer player for TEMPO. One possibility is to increase the complexity of the game through the addition of more weapon types (A, B, C etc.). This could provide a more interesting and challenging scenario, and could allow further investigation for the use of INTEL/CI in the game. Another area of investigation is the inclusion of the research and development component. This would require analysis for impact on the game, as well as the computer players. However, it would also increase the interest and strategy making challenge for the game. For further realism in the game, it could also be beneficial to change the pwar environmental parameter to change for the circumstance. If players both start purchasing large quantities of offensive weaponry, the chances of war breaking out could increase. Another possibility is to incorporate more than two players in the game. Providing additional players changes the dynamics of the game, where alliances provide a cooperative scenario in addition to the competitive one.

Other directions of research are to use the HACP system, or the other research components presented in this thesis, and apply them to different domains. While our scenario involved a competitive resource allocation game, the same techniques could be applied for other scenarios such as cooperative games. Additionally, the concept of evolving a strategy has any number of forms and applications.

Finally, through the HACP system we were able to achieve our overall hypothesis and show that coevolutionary algorithms are indeed an effective mechanism to create a computer player for strategic decision-making games. We have shown that human players respond well to a computer player that adapts to their strategies, and that it is useful for learning the development of new strategies. In turn, the human players develop increased strategy making skills. We have succeeded in creating a system that achieved what we set out to do, and have provided a number of novel techniques to apply coevolution for the development of strategic computer players. Our research provides many exciting possibilities for future work, and we look forward to seeing these applied.

Bibliography

[1] *Websters Dictionary of the English Language*, volume 1.

[2] The Planning, Programming, and Budgeting System (PPBS) directive. Technical report, USA Department of Defense, May 22 1984.

[3] *The Macquarie Dictionary*. The Macquarie Library Pty. Ltd., 1996.

[4] TEMPO military planning game – explanation and rules for players. Technical report, 2003.

[5] D. Aha, M. Molineaux, and M. Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In *Case-Based Research and Development*, volume 3620/2005, pages 5 – 20, 2005.

[6] R. Atkinson and R. Shiffrin. Human memory: a proposed system and its control processes. In *The Psychology of Learning and Motivation: Advances in Research and Theory*, volume 2, pages 89 – 195. Academic Press, New York, 1968.

[7] P. Avery, G. Greenwood, and Z. Michalewicz. Coevolving strategic intelligence. In *IEEE Proceedings for Congress on Evolutionary Computation*, Hong Kong, China, 2008.

[8] P. Avery and Z. Michalewicz. Static experts and dynamic enemies in coevolutionary games. In *IEEE Proceedings for Congress on Evolutionary Computation*, pages 4035 – 4042, Singapore, 2007.

[9] P. Avery, Z. Michalewicz, and M. Schmidt. A historical population in a coevolutionary system. In *IEEE Symposium on Computational Intelligence and Games*, pages 104 – 111, Honolulu, Hawaii, USA, 2007.

[10] P. Avery, Z. Michalewicz, and M. Schmidt. Short and long term memory in coevolution. In *International Journal of Information Technology and Intelligent Computing*, volume 3, 2008.

[11] R. Axelrod. More effective choice in the prisoner's dilemma. In *Journal of Conflict Resolution*, volume 23, pages 379 – 403, 1980.

[12] A. D. Baddeley. The psychology of memory. In *The Essential Handbook of Memory Disorders for Clinicians*, pages 1 – 13. John Wiley and Sons, 2005.

[13] A. Bader-Natal and J. B. Pollack. A population-differential method of monitoring success and failure in coevolution. In *Genetic and Evolutionary Computation Conference*, volume 2723 of *Lecture Notes in Computer Science*, pages 585–586. Springer, 2004.

[14] L. Barone and L. While. Evolving adaptive play for simplified poker. In *IEEE World Congress on Computational Intelligence.*, pages 108 – 113, May 1998.

[15] L. Barone and L. While. Evolving computer opponents to play a game of simplified poker. In *International Conference on Evolutionary Computation*, pages 153 – 160, 1998.

[16] A. Barzel. The perplexing conclusion: The essential difference between natural and artificial intelligence is human beings ability to deceive. In *Journal of Applied Philosophy*, volume 15, pages 165 – 178, 1998.

[17] K. Becker. Teaching with games: the minesweeper and asteroids experience. In *Journal of Computing Sciences in Colleges*, volume 17, pages 23 – 33, USA, 2001. Consortium for Computing Sciences in Colleges.

[18] J. Beusmans and K. Wieckert. Computing, research, and war: If knowledge is power, where is responsibility? In *ACM Social Aspects of Computing*, volume 32, pages 939 – 947, 1989.

[19] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. In *Artificial Intelligence*, volume 134, pages 201–240, 2002.

[20] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in Poker. In *Proceedings of the 15th National Conference on Artificial Intelligence)*, pages 493 – 498. AAAI Press, 1998.

[21] J. Bracker. The historical development of the strategic management concept. In *Academy of Management Review*, volume 5, pages 219 – 224, 1980.

[22] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Computation*, pages 1875 – 1882, Washington, USA, 1999.

[23] L. Bull. On coevolutionary genetic algorithms. volume 5, pages 201–207, 2001.

[24] M. Buro and T. M. Furtak. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference*, pages 51 – 58, 2004.

[25] R. Caillois. *Man, Play and Games*. University of Illinois Press, 2001.

[26] S. Carberry. Techniques for plan recognition. In *User Modeling and User-Adapted Interaction*, volume 11, pages 31 – 48. Kluwer Academic Publishers, 2001.

[27] D. Charles, M. McNeill, M. McAlister, M. Black, A. Moore, K. Stringer, J. Kucklich, and A. Kerr. Player-centered game design: Player modeling and adaptive digital games. In *DiGRA 2005 Conference: Changing Views – Worlds in Play*, 2005.

[28] K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. In *Proceedings of the IEEE*, volume 87, pages 1471 – 1496, 1999.

[29] K. Chellapilla and D. B. Fogel. Evolving an expert checkers playing program without using human expertise. In *IEEE Transactions on Evolutionary Computation*, volume 5, pages 422 – 428, 2001.

[30] B. Chin. Sense-And-Respond logistics evolving to predict-and-preempt logistics. In *Army Magazine*, volume May, pages 61 – 72, 2005.

[31] S. Chong, D. Ku, H. Lim, M. Tan, and J. White. Evolved neural networks learning Othello strategies. In *The 2003 Congress on Evolutionary Computation*, volume 3, pages 2222 – 2229, Newport Beach, California, USA, 2003.

[32] K. Chopra and C. Haimson. Information fusion for intelligence analysis. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, page 111, 2005.

[33] D. Cliff and G. F. Miller. Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulation. In *European Conference on Artificial Life*, pages 200 – 218, 1995.

[34] F. Craik and R. Lockhart. Levels of processing: a framework for memory research. In *Journal of Verbal Learning and Verbal Behavior*, volume 11, pages 671 – 684, 1972.

[35] F. Craik, A. Routh, and D. Broadbent. On the transfer of information from temporary to permanent memory. In *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, volume 302, pages 341 – 359, 1983.

[36] I. L. Davis. Strategies for strategy game AI. In *AAAI 1999 Spring Symposia*, pages 24 – 27, 1999.

[37] E. D. de Jong. The incremental pareto-coevolution archive. In *Genetic and Evolutionary Computation – GECCO 2004*, pages 525 – 536, 2004.

[38] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[39] K. A. De Jong. *Evolutionary Computation, A Unified Approach*. The MIT Press, 2006.

[40] K. A. De Jong and M. A. Potter. Evolving complex structures via cooperative coevolution. In *Evolutionary Programming*, pages 307 – 317, 1995.

[41] A. K. Dixit and B. J. Nalebuff. *Thinking Strategically: The Competitive Edge in Business, Politics, and Everyday Life*. W.W. Norton & Company, 1991.

[42] J. Doyle and T. Dean. Strategic directions in artificial intelligence. In *ACM Computing Surveys*, volume 28, pages 653 – 670, 1996.

[43] H. M. Eisenhardt and M. J. Zbaracki. Strategic decision making. In *Strategic Management Journal*, volume 13, pages 17 – 37. John Wiley & Sons, Ltd., 1992.

[44] S. L. Epstein. Game playing: The next moves. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 987 – 993. AAAI Press, 1999.

[45] S. G. Ficici and J. B. Pollack. A game-theoretic memory mechanism for coevolution. In *GECCO 2003, Lecture Notes in Computer Science*, volume 2723/2003, pages 286 – 297. Springer Berlin/Heidelberg, 2003.

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

[47] J. C. Giordano, P. F. R. Jr., and D. C. Brogan. Exploring the constraints of human behavior representation. In *Proceedings of the 2004 Winter Simulation Conference*, pages 912 – 920, 2004.

[48] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[49] T. D. Gwiazda. *Genetic Algorithms Reference; Volume I – Crossover for single-objective numerical optimization problems*. Tomasz Dominik Gwiazda, 2006.

[50] T. D. Gwiazda. *Genetic Algorithms Reference; Volume II – Mutation operator numerical optimization problems*. Tomasz Dominik Gwiazda, 2007.

[51] S. P. Hargreaves Heap and Y. Varoufakis. *Game Theory: A Critical Introduction*. Routledge, 1995.

[52] W. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In *Emergent computation*, pages 228 – 234. MIT Press, Cambridge, MA, USA, 1991.

[53] E. Hurwitz and T. Marwala. Learning to bluff. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1188 – 1193, 2007.

[54] IBM. Deep blue. Online, 20 September 2005. `http://researchweb.watson.ibm.com/deepblue/`.

[55] R. W. Johnson, M. E. Melich, Z. Michalewicz, and M. Schmidt. Coevolutionary optimization of fuzzy logic intelligence for strategic decision support. In *IEEE Transactions on Evolutionary Computation*, volume 9, pages 682 – 694, 2006.

[56] R. W. Johnson, Z. Michalewicz, M. E. Melich, and M. Schmidt. Coevolutionary tempo game. In *Congress on Evolutionary Computation*, volume 2, pages 1610 – 1617, Portland, Oregon, 2004.

[57] H. Juille and J. B. Pollack. Dynamics of co-evolutionary learning. In P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4*, pages 526 – 534, Cape Code, USA, 1996. MIT Press.

[58] G. Kendall, R. Yaakob, and P. Hingston. An investigation of an evolutionary approach to the opening of go. In *2004 Congress on Evolutionary Computation*, volume 2, pages 2052 – 2059, Portland, Oregon, USA, 2004.

[59] S. J. Louis and J. Johnson. Solving similar problems using genetic algorithms and case-based memory. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283 – 290, San Francisco, CA, 1997. Morgan Kaufmann.

[60] S. J. Louis and J. McDonnell. Learning with case-injected genetic algorithms. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 316 – 328, 2004.

[61] S. J. Louis and J. McDonnell. Playing to train: Case injected genetic algorithms for strategic computer gaming. In *GECCO*, 2004.

[62] S. J. Louis and C. Miles. Combining case-based memory with genetic algorithm search for competent game AI. In *ICCBR Workshops*, pages 193 – 205, 2005.

[63] S. J. Louis and C. Miles. Playing to learn: Case-injected genetic algorithms for learning to play computer games. In *IEEE Transactions on Evolutionary Computation*, volume 9, pages 669 – 681, 2005.

[64] Chairman of the Joint Chiefs of Staff. Doctrine for logistic support of joint operations. In *Joint Publication 4-0*, April 2000.

[65] Headquarters Department of the Army. Intelligence, field manual FM 2-0, May 2004.

[66] Office of Force Transformation, Department of Defense USA. Operational sense and respond logistics: Coevolution of an adaptive enterprise capability. In *Sense and Respond Logistics Metrics Overview*, 15 October 2007.

[67] J. McCarthy. Some expert systems need common sense. In *Proceedings of a symposium on Computer culture: the scientific, intellectual, and social impact of the computer*, pages 129 – 137, New York, NY, USA, 1984. New York Academy of Sciences.

[68] L. Messerschmidt and A. Engelbrecht. Learning to play games using a PSO-based competitive learning approach. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 280 – 288, June 2004.

[69] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer -Verlag, second edition, 2004.

[70] C. Miles, S. Louis, N. Cole, and J. McDonnell. Learning to play like a human: case injected genetic algorithms for strategic computer gaming. In *Congress on Evolutionary Computation*, volume 2, pages 1441 – 1448, 2004.

[71] C. Miles, J. Quiroz, R. Leigh, and S. Louis. Co-evolving influence map tree based strategy game players. In *Computational Intelligence and Games*, pages 88 – 95, 2007.

[72] R. W. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Springer-Verlag, 2004.

[73] J. Murdoch. Bioshock. Online, May 2007. `http://www.gamerswithjobs.com/node/32465`.

[74] J. F. Nash. *Non-cooperative Games*. PhD thesis, Princeton University, 1950.

[75] M. Nerome, K. Yamada, S. Endo, and H. Miyagi. Competitive co-evolution model on the acquisition of game strategy. In *Simulated Evolution and Learning*, pages 224 – 231, 1996.

[76] W. C. Oon and Y. J. Lim. An investigation on piece differential information in co-evolution on games using Kalah. In *The 2003 Congress on Evolutionary Computation*, volume 3, pages 1632 – 1638, Newport Beach, California, USA, 2003.

[77] M. J. Osborne. *An Introduction to Game Theory*. Oxford University Press, 2004.

[78] S. D. Pinson, J. A. Louy, and P. Moraitis. A distributed decision support system for strategic planning. In *Decision Support System*, volume 20, pages 35 – 51, 1997.

[79] M. Ponsen, H. Muoz-Avila, P. Spronck, and D. Aha. Automatically generating game tactics via evolutionary learning. In *AI Magazine*, volume 27, pages 75 – 84, 2006.

[80] M. Ponsen and P. Spronck. Improving adaptive game AI with evolutionary learning. In *Computer Games: Artificial Intelligence, Design and Education*, pages 389 – 396, 2004.

[81] M. Ponsen, P. Spronck, H. Muoz-Avila, and D. Aha. Knowledge acquisition for adaptive game AI. In *Science of Computer Programming*, volume 67, pages 59 – 75, 2007.

[82] D. L. Poole, A. Mackworth, and R. G. Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, New York, January 1998.

[83] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the International Conference on Evolutionary Computation*, pages 249 – 257, 1994.

[84] N. Puppala, S. Sen, and M. Gordin. Shared memory based cooperative coevolution. In *Proceedings of the 1998 IEEE Conference on Evolutionary Computation (ICEC'98)*, pages 570 – 574. IEEE, 1998.

[85] P. S. Rosenbloom. A world-championship-level Othello program. In *Artificial Intelligence*, volume 19, pages 279 – 320, 1982.

[86] C. D. Rosin and R. K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 373 – 380, San Francisco, CA, 1995. Morgan Kaufmann.

[87] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. In *Evolutionary Computation*, volume 5, pages 1 – 29, 1997.

[88] F. Schadd, S. Bakkes, and P. Sprock. Opponent modeling in real-time strategy games. In *8th International Conference on Intelligence Games and Simulation*, volume GAMEON-NA,2007, pages 61 – 68, 2007.

[89] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship calibre checkers program. In *Artificial Intelligence*, volume 53, pages 273 – 290, 1992.

[90] M. Schoenauer and Z. Michalewicz. Evolutionary computation: An introduction. In *Control and Cybernetics*, volume 26, pages 307 – 338, 1997.

[91] C. R. Schwenk. *The essence of strategic decision making*. D.C. Heath and Company, 1988.

[92] C. R. Schwenk. Strategic decision making. In *Journal of Management*, volume 21, pages 471 – 493, 1995.

[93] B. Scott. AI game programming wisdom. In S. Rabin, editor, *The Illusion of Intelligence*, pages 16 – 20. Charles River Media, 2002.

[94] W. E. Spangler. The role of artificial intelligence in understanding the strategic decision-making process. In *IEEE Transactions of Knowledge and Data Engineering*, volume 3, pages 149 – 159, 1991.

[95] P. Spronck. *Adaptive Game AI*. PhD thesis, Maastricht University, 2005.

[96] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game AI with dynamic scripting. In *Machine Learning*, volume 63, pages 217 – 248, 2006.

[97] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Online adaptation of computer game opponent AI. In *Proceedings of the 15th Belgium-Netherlands Conference on Artificial Intelligence*, volume 2003, pages 291 – 298, 2003.

[98] D. Sudharshan. *Marketing Strategy. Relationships, Offerings, Timing & Resource Allocation*. Prentice Hall, 1995.

[99] R. S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, volume 3, pages 9 – 44, 1988.

[100] D. J. Trump and T. Schwartz. *Trump, The Art of the Deal*. Random House, New York, 1987.

[101] A. Turing. Computing machinery and intelligence. In *MIND*, volume LIX, pages 433 – 460, 1950.

[102] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[103] L. X. Wang. *Adaptive Fuzzy Systems and Control – Design and Stability Analysis*. Prentice Hall, 1994.

[104] H. Xie, M. Zhang, and P. Andreae. An analysis of constructive crossover and selection pressure in genetic programming. In *Genetic and Evolutionary Computation Conference (GECCO'07)*, volume 2, pages 1739 – 1746, 2007.

[105] L. Zadeh. Fuzzy sets. In *Information and Control*, volume 8, pages 338 – 353, 1965.

To whom it may concern,

The following provides information regarding the changes made to address the examiner's comments in accordance with my supervisor and head of school.

Regards,

Phillipa Avery

To address the concerns of Kenneth De Jong, the following changes were made.

The title was changed to represent the specific game of TEMPO.

The justification of parameter choices made has been addressed in two ways. Firstly, a theoretical description of the domain was added to the coevolutionary background section. Changes made have been highlighted with italics.

As with the EA, there are a number of variations that exist on the basic algorithm. There can be numerous populations/species, which can change the way individuals are evaluated. Coevolution can also be used to find solutions through cooperation instead of competition, where individuals are evaluated on how well they form a complete solution. For example, De Jong and Potter [40, 83] describe a technique that coevolves a complex structure by decomposing the structure into substructures, and assigning a species for each substructure. The species are then combined through cooperative coevolution to find a complete structure.

As the purpose of coevolution is to compete against the opposition species, the evaluation stage of the algorithm becomes an important factor. With EAs the correct design of the evaluation function is crucial to provide a useful outcome, and it is no different with coevolution. In coevolution on a general level, the fitness of an individual should be determined by how well they do at the task at hand against the opposition population. One of the key factors here is the measurement against the opposition species. The manner of selecting opposition individuals can affect the performance of the evaluation. Rosin and Belew recognized this restraint in [87], and suggested techniques that select from the opposition population using random selection, best of generation selection and competitive fitness sharing selection. Bull [23] also evaluates the use of a roulette wheel style of selection for evaluation. Theoretically, any number of different methods of selection could be used depending on the desired outcome and problem at hand.

The quality of the evaluation function is also affected
opposition species. To truly measure the effectiveness of an individual, it should theoretically be played against all opposition individuals to assign a fitness. However, often this is not a realistic task due to processing requirements, and restricted selection from the opposition must be performed. If the selection is too restricted however, the evolution could suffer an insufficient view of the opposition solution space, and adaptation to the opposition species

could be hampered. Ideally, a compromise between processing time and evaluation performance should be found.

Another factor to affect coevolutionary algorithm is that of *cycling* or *forgetting*. As mentioned before, by incorporating more than one species in the evolutionary process, the different species affect the fitness landscapes of their opponents. This works as follows: suppose one population develops a strategy that works against an opposition, that strategy should then spread throughout the population. The only way the other species can compete is to focus on beating that strategy. Thus, the focus of coevolution is to beat the other species' current winning solution, not find an overall optimal solution (if one even exists). In problems where an evaluation function for an optimal solution can be difficult to define, this can be a distinct advantage. However, this process can also lead to stagnation in the adaptive process as the species begin to cycle solutions.

Coevolutionary learning happens at a species level, and the individuals that are not performing competitively are phased out. As a result, individuals that were at one point competitive, but have been countered by an opposing species, are subsequently left behind. At some future stage in the coevolution these individuals may once again become good strategies, and must be 're-invented' by the coevolutionary process. This cycling behaviour in conjunction with the way the fitness landscape of one species chases the other species' form the Red Queen effect [33]. The Red Queen is a character from Lewis Carroll's *Through The Looking Glass*, who was constantly running and never getting anywhere, as the surrounding landscape was keeping up with her.

As mentioned, coevolution is ideal for problems where the optimal can be hard to define. This concept is a perfect application for the creation of self learned computer players for simultaneous games. Defining an evaluation function to find 'the' optimal player for this is near impossible, as there is no real measurement to use. By utilizing coevolution however, the players are evaluated on how well they learn play, and players with good strategies are allowed to emerge. They are encouraged to find winning strategies by simply playing against another population of individuals that are doing the same thing.

To address the justification of used parameters, the following was added/changed in the approach chapter:

After the initialization, the individuals are evaluated against the other population. The basic system applies the following evaluation technique (with modifications introduced later). We iterate over each population, with each individual played against r randomly chosen individuals from the opposition population. The random selection mechanism was carried over from the research by Johnson et. al., and it is possible that a different be more effective. For the purposes of this research however it is sufficient, and further experiments on the matter are outside the scope of our work. For most of our experiments we used a sample size of r=20. Through experimentation we found that this size gave us enough sample scope for the population size of 100 that was used in the experiments.

and then in the memory chapter:

Our initial experiments with this system involved implementing the memory structure as described and randomly selecting individuals to play against. To evaluate the fitness, a number of games were played against the opposition (r_1=20), re-adjusting the evaluation after each game. To include this strategy of evolving against the memory, we then provided a mechanism for playing an additional number of games (r_2=20) against the memory, and adjusted the fitness of the current individuals in the same manner as that for the opposition games. The additional sample size of r_2=20 was chosen to match the opposition sample size, and minimize the impact of the memory on the performance.

for the use of ten runs in the experiments:

All experiments performed in this section have the same experimental settings, except when stated otherwise. Due to the stochastic nature of the experiments, we conducted each experiment over ten separate runs, all with the same environmental and evolutionary configuration. Due to the time and hardware restraints ten runs was consistently used for all experiments. However, we did experimentally run more than ten times, and there was not a distinctly noticeable difference. Less than ten runs did however have a greater impact.

Finally, we included additional graphs representing the standard deviation on either side of the average results. We used a Bezier curves function to smooth the average so the trends were more visible. We then included the standard deviation + and - on the same graph. This hence provides a much better way to read the results, with the inclusion of a confidence interval.

The use of memory with multi-objective optimization was used as reference when deciding what direction to take with the research. However, it was not directly connected with the research presented in the thesis, and was not included in the literature review/background chapter for this reason. We stated in the background chapter that we were restricting the description to the use of memory in coevolution, and focused on the techniques used, not the domains they were applied to. If we were to include a review of all the research into the addition of memory for the different domains and areas outside coevolution, it would greatly increase the length of the review without much gain to the reader. From our review of the current research, there has not been any unique methods using memory and coevolution with multi-objective research. Most used memory structure variations from the research described in the literature review on the subject. However, to clarify this point, the following paragraph in the literature review was changed accordingly:

This section discusses some of the memory solutions that have been developed specifically for coevolution. There has also been work on the addition of memory to EAs for use in many

domains such as game playing and multi-objective optimization. In such domains, it can be important to retain an external memory structure to store past information. Extensions to the EA exist for the initialization and supplementing of individuals [72], and using the memory to guide the search. There has been a lot of work on this subject, part of which is reviewed by Branke in [22]. Branke defines two forms of memory. The first is implicit memory, where the individuals themselves have some form of redundant information forming a long term memory. The second category is explicit memory, where there is a storage mechanism used to reintroduce previously learnt information at a later stage of evolution (e.g. replacing weaker individuals with previously good ones, or initializing an EA with individuals from a similar previous EA run). The remainder of this section however, focuses on the specific research of memory in coevolution, as the purpose and outcomes of its use differ

---

Subsequently, we believe the additions made for these concerns also addresses the concerns put forth by Marc Schoenauer. The use of the Bezier curves representation throughout the results addresses his third point. Additionally the smoothed results makes the changes in results more apparent addressing his second point.

The suggestion of using the evolved players fighting against each other as a performance measure is confusing, as this is the precise way that coevolution works. Plotting the graph of this measure would provide no measurable improvement, as it changes with the opposition population each generation. It is for this exact reason that Johnson et. al. (whose research this work was based upon) chose to use a 'expert' as a measurement technique. The use of a static mechanism as a measurement metric was also addressed by other research which has been mentioned in the literature review.