Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games

Thomas Willer Sandberg
twsa@itu.dk
220584-xxxx

Supervisor
Julian Togelius

Master of Science
Media Technology and Games
IT University of Copenhagen

February 2011

Abstract

This master thesis presents an approach for how to implement an Evolutionary Multi-Agent Potential Field (EMAPF) based AI in small scale combat (SSC) scenarios in real-time strategy (RTS) computer games. The thesis will show how evolutionary algorithms can be used to auto tune and optimize potential field parameters for making unit agents able to navigate and hunt down enemy squads/troops in an efficient way in an unknown environment.

The very popular top-selling RTS game StarCraft: Brood War has been chosen as test platform, mainly because the game is extremely well balanced and has a complexity in the game world, which makes the development of AI's very challenging. Since StarCraft is not open-source, the free open-source framework Brood War Application Programming Interface (BWAPI) has been used to communicate with the game.

In order to make the AI able to control unit agents moving around in the game world, they will be effected by different types of potential fields placed in both static and dynamic tactical places around the game world. One of the hurtles when designing the potential fields has earlier been the tuning of them, which can be time consuming, if done manually. This thesis therefore presents an approach for how auto tuning with the use of evolutionary algorithms can be implemented into the Multi-Agent Potential Fields methodology.

The work that has been done in this thesis provides a good basis for future work for designing EMAPF based AI for full scale RTS games. The potential fields found in this thesis could be adapted by an AI playing full RTS games, StarCraft in particular. The AI will be able to turn the potential fields on and off, depending on what tactics are found to be best suited to different states.

Acknowledgements

First, I would like to thank my supervisor, Julian Togelius, for his feedback and his help on keeping focus on the important and thereby guiding my work in the right direction. Secondly I would like to thank my secondary supervisor Georgios Yannakakis for giving me an overview of the various game AI projects available and for introducing me to Julian Togelius.

Also I would like to thank the people behind The Brood War Application Programming Interface (BWAPI) for providing a free and open source framework for creating AI modules for StarCraft. I am also thankful to the people behind bwapi-mono-bridge for making it possible to use C#.NET to write a StarCraft AI and thanks to Johan Hagelbäck for helping me get started with Multi-Agent Potential Fields.

Last but not least I would like to thank my wife Birgitte and my son Elias for always being there for me, even while I was so busy with this project.

Contents

1 Introduction **1**
- 1.1 History of the RTS genre . . . 1
- 1.2 AI in RTS games . . . 2
- 1.3 Motivation . . . 3
- 1.4 Contribution . . . 4
- 1.5 Limitations . . . 4
- 1.6 Problem Statement . . . 5
- 1.7 Thesis Structure . . . 5

2 Related Work **6**

3 Methodology **8**
- 3.1 Artificial Potential Fields . . . 8
- 3.2 Multi-Agent Potential Fields . . . 9
- 3.3 Evolutionary Algorithms . . . 10

4 Test Platform Setup **13**
- 4.1 StarCraft . . . 13
  - 4.1.1 Physical Properties . . . 14
  - 4.1.2 Unit control methods . . . 14
- 4.2 BWAPI challenges . . . 15
- 4.3 System Requirements . . . 17

5 EMAPF for SSC RTS Scenarios **19**
- 5.1 Identifying objects . . . 19
- 5.2 Identifying fields . . . 20
- 5.3 Assigning charges to the objects . . . 22
- 5.4 Tuning the charge parameters . . . 26
  - 5.4.1 EA Problem definition . . . 27
  - 5.4.2 Parameters . . . 27
  - 5.4.3 Evaluation function (Fitness function) . . . 27
- 5.5 Assigning granularity of time and space in the environment . . . 28
  - 5.5.1 Training map units . . . 29
- 5.6 The agents of the AI system . . . 31
- 5.7 The MAS architecture . . . 32

6 Experiments and Results **33**
6.1 Initial experiments 33
6.2 Main experiments and results 35
6.2.1 Discussion 37
6.3 Competition and observations 40

7 Conclusion **44**
7.1 Competition 44
7.2 EMAPF 45
7.3 EA 45
7.4 Future Work 46

Appendix: Extra experiments **50**

Chapter 1

Introduction

1.1 History of the RTS genre

The real-time strategy (RTS) game genre has been very popular since the first RTS game came out. There have been some discussion about which RTS game was the first. Some people claim that Utopia from 1981 (by Intellivision) was the first, others say that Dune II from 1992 (by Westwood Studios) was the first real RTS Game(Barron, 2003).

Even if Utopia from 1981 actually was the first RTS game, Dune II was the first modern RTS game to use the mouse to control units, and it laid the foundation for all modern RTS games. Dune II from 1992 and later Command & Conquer (1995), Red Alert (1996) from Westwood Studios, Warcraft (1994) and later StarCraft (1998) from Blizzard Entertainment are the main reason for the initial popularity of the genre, and thus define what people see as a modern RTS game today (Geryk, 2010).

Since then, the Age of Empires (AoE) series produced by Ensemble Studios (owned by Microsoft since 2001), has helped keep the popularity high until the last AoE was released in 2007 and the company closed down, at the same time as their last game, Halo Wars, was published, in the start of 2009. Luckily this is not the end of the RTS genre as the popularity is still high. When Blizzard released StarCraft II in the end of July 2010, 1.5 million copies were sold after just 48 hours and over three million copies were sold worldwide in the first month after its release (Release, 2010). At the same time, Electronic Arts is still releasing new Command and Conquer and Red Alert games regularly and some of the old employees from the closed company Ensemble Studios have started new companies, for example Robot Entertainment who is working on releasing an Age Of Empires Online[1] later in 2011.

The precise definition of an RTS game has for many years been debated in the game industry. Blizzard's Rob Pardo has, for instance, defined an RTS game as: "A strategic game in which the primary mode of play is in a real-time setting" (Adams, 2006). In my opinion this is too broad a definition, by which a game like SimCity would be defined as an RTS game, which I would not consider as such, but rather as a simulation game. I would define an RTS game as a military strategic game where the player controls resource gathering, base / unit building (macromanagement) and unit control (micromanagement) in a real time setting, in which the primary goal is

[1] `http://www.ageofempiresonline.com/`

to defeat the enemy by strategic thinking. This definition will be the one I use for the rest of this thesis, when I refer to an RTS game.

The most popular RTS game of all is still undoubtedly StarCraft by Blizzard Entertainment. According to (JB, 1999) StarCraft was the best-selling computer game of 1998 with more than 1.5 million copies sold worldwide. The 2009 Guinnes World Records (Release, 2009) announced StarCraft as being the best-selling strategy game for PC ever, with 9.5 million copies sold worldwide. Actually, Blizzard COO Paul Sams claims that the original Starcraft has sold over 11 million copies worldwide(Graft, 2009), which makes StarCraft the fourth best sold PC game ever, only surpassed by The Sims, The Sims 2 (developed by Maxis and published by Electronic Arts) and World of Warcraft (Blizzard Entertainment) (List-All, 2008; Wikipedia, 2010).

1.2 AI in RTS games

One of the biggest issues with almost all RTS games today, including StarCraft, is that the build-in computer player, often denoted as AI, is not able to give hard enough resistance to defeat skilled human players without cheating. In many RTS games, the most difficult AI's are in fact cheating, using complete map information (no Fog of War), having more start-up resources, accelerated build time and/or faster income. According to (Nareyek, 2004), cheating "'is very annoying for the player if discovered"'. It would therefore be very interesting to see a non-cheating hard AI being able to defeat a skilled human player with the same conditions as the human player has.

For many years conventional planning based solutions like e.g. the basic A* algorithm and different types of state machines has been state of the art for path-finding and navigation in most RTS games. The A* algorithm is very good at handling long range movement with static objects (Liu et al., 2010; Pinter, 2001), but when it comes to dynamic obstacle avoidance (e.g. other agents) in a real time environment in RTS games like StarCraft, the A* algorithm falls short. This is probably one reason for the growing interest in alternative solutions during the last decade. Other reasons could be the increasing CPU power in the newest PC's and consoles or new demands from the end users for more believable and challenging AI's. Because of the fact that no one has yet been able to make an AI capable of defeating the highest skilled human players, there is still a lot of research to be done in the field of RTS games.

Thus, it might be time for new types of algorithms that are more efficient. Johan Hagelbäck from Blekinge Institute of Technology has done some research in the use of Multi-agent Potential Fields (MAPF) for RTS games and has been able to build an AI, using MAPF, capable of winning more than 99 percent of the games against four of the top five teams at the Open Real Time Strategy (ORTS) tournament 2007, and at the same time it was able to easily win the 2008 tournament too. He believes that MAPF will gain ground as the game AI field matures (Hagelbäck and Johansson, 2008c; Hagelbäck and Johansson, 2008b). All the other competition entries were based on more traditional approaches with path-finding and higher level planning (Hagelbäck and Johansson, 2008d). Therefore it is interesting to see that

the MAPF has better performance than the more traditional solutions. One of the reasons MAPF has not been used much by the gaming industry has been that the tuning of the MAPF can be very difficult and time consuming(Hagelbäck, 2009a). This thesis will therefore explore the possibilities of using optimization techniques like evolutionary algorithms to automate the tuning of the MAPF and thereby avoid the need of spending too much effort on MAPF tuning.

1.3 Motivation

Besides from being one of the best-selling games ever, it is incredible that a more than ten year old game can still get so much attention from fans. In Korea, there is a professional league of StarCraft players (Bellos, 2007) and StarCraft is also the only game that has been competed in for all ten years the international e-sports event The World Cyber Games (WCG) has been held[2].

The great interest in StarCraft has also influenced the AI community. In 2010 a huge StarCraft AI competition was held at the 2010 conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE) with 29 AI competitors at The Expressive Intelligence Studio in UC Santa Cruz[3]

But why is this game so popular? I think some of the reasons are that the story and universe are so interesting, and that the three different races are so different and at the same time so balanced and competitive, that no one has been able to find a complete strategy plan that can beat everyone, like the IBM's Deep Blue AI in Chess.

It has always fascinated me how the AI were able to control so many units at once in RTS games. In many other game genres such as most platform and first-person shooter (FPS) games, each player need only control one character/unit at a time. In RTS games like StarCraft, a whole army of up to 200 men would need to be able to navigate a dynamic game world, while at the same time avoid colliding with others, find and destroy the enemy/enemies using the right tactics, build up a base and an army which is able to win against the enemy, defend own bases, scout and plan ahead who/where and when to attack. At the same time RTS games operate in real-time, making it a very challenging task to implement an RTS AI which is fast enough to avoid situations where the units are standing idle, because the bot is thinking about the next move.

Initially, I started out exploring the possibilities of using Monte Carlo Tree Search algorithm as an AI solution for an RTS game like StarCraft, but it seemed impossible to make a StarCraft simulator fast enough to run the game smoothly. Instead I started to explore the use of Artificial Potential Fields (APF) in games to see if someone had had success with implementing that for RTS games. As mentioned earlier, Johan Hagelbäck had explored the use of MAPF(Hagelbäck, 2009a), which e.g. extends the knowledge of APF (see Chapter 2 and 3 for more information), in AI's for RTS games, and it seemed to be efficient enough to outperform more traditional AI solutions, if the MAPF is implemented correctly.

[2] `http://en.wikipedia.org/wiki/World_Cyber_Games`
[3] `http://eis.ucsc.edu/StarCraftAICompetition`

While it is very difficult and time consuming to tune MAPF, it is interesting to see if it would be possible to automate the tuning of the different MAPF found for a specific problem; in this case tuning of MAPF for the RTS game StarCraft.

1.4 Contribution

This thesis introduces an approach for how to auto-tune and optimise multi-agent potential field parameters for AI's in RTS computer games, StarCraft in particular.

The findings in this thesis show that it is possible to find an optimal or near optimal combination of potential field parameters with the use of evolutionary algorithms.

1.5 Limitations

Small-scale combat (SSC) tactic skills is very important to have for a StarCraft player. A StarCraft match can easily be determined by how well a player controls his units in battle (micromanagement). Of course it is also very important for a StarCraft player to have a good build Order for specific maps and situations, where the player is able to manage economy and unit building, and control the overall strategy plan (macromanagement). But players who can build large armies fast but are weak in micromanaging units will not win against players who master their micro management skills, because those players will often be able to win SSCs against large armies with only few units and ultimately beat the opponent player.

Of course, having a mixture of both strong micro- and macromanagement will be essential for a good StarCraft player. But in this thesis the focus will only be on micro management and this will be reflected in the use of only SSC maps, where both teams only have units and no buildings or minerals.

The reason for this limitation is primarily a time issue. Six months is simply too little time for one person, with no prior experience with RTS Game AI, multi-agent potential fields and the BWAPI framework, to make a strong full blown RTS Game AI and at the same time document everything in details.

It would naturally take a lot longer to run a full StarCraft game than it does running a SSC game. When optimizing MAPF we will need some kind of evolutionary algorithm (EA) and here it would be necessary to run thousands of games and it would simply take too much time to run so many full games, compared to running SSC games. It would of course be possible to tune and optimize MAPF for macromanagement, like where to build, mining etc., but more processor power and time would be needed.

The trained evolutionary (E)MAPF-based AI solutions found and optimized in this thesis, will of course be able to be used for a full blown StarCraft AI. The different potential fields found in the thesis could just be turned on and off, depending on higher tactical decisions.

To reduce the noise influence in the optimisation process I will avoid using advanced maps with lots of big static obstacles, because the focus of the tuning is on optimizing battle tactics with as few killed agents and as many hit points and

shields left as possible.

I will only optimize and tune MAPF for a small number of different unit types for one race, because my goal is to make a proof of concept and thereby show that it is possible to auto tune and optimize MAPF for an RTS game, which will not require optimising the potential fields for all the unit agents in StarCraft. If a full blown AI is made later, it would of course be a very good idea to optimise the potential field parameters for all the units in StarCraft.

For simplicity, the fog of war is not active in this master thesis, i.e complete map information is activated, mainly because it would take too much extra unnecessary time for the EA, if the units needed to explore the map to find the enemy. If the explored map areas were also to be handled, a map handler and other high level tactical planners would need to be implemented and would have removed focus from the concrete purpose of this thesis (see 1.6).

1.6 Problem Statement

This master thesis will explore the possibilities of using an Evolutionary Algorithm to tune and optimize a multi-agent potential field based AI, for a small-scale combat scenario in an RTS game. To research this, an open-source RTS game will be used as test bed platform for testing the implemented algorithms. The very popular RTS game StarCraft: Brood War is chosen for this purpose, as I have judged it to be one of the best candidates, mainly because the game is well balanced and can be sped up many times compared to normal game speed, which is necessary when using Evolutionary Algorithms. StarCraft is in itself not open-source, but a free open-source framework called Brood War Application Programming Interface (BWAPI) will be used to grant control of the StarCraft platform.

1.7 Thesis Structure

This master thesis is structured into several chapters each of which presents different kinds of contributions for the project. The different chapters contain:

- Chapter 2: the related work that has been done in the field of this thesis.
- Chapter 3: the methodology of the different algorithms and approaches used to achieve the goal of this thesis.
- Chapter 4: the setup and creation the test bed platform.
- Chapter 5: the creation of the EMAPF-based AI in a SSC RTS Scenario.
- Chapter 6: the experiments and their results.
- Chapter 7: the conclusion and some thoughts on possible future work in the field.

The source code for the developed EMAPF-based AI for this master thesis can be found on the enclosed CD.

Chapter 2

Related Work

This chapter will present related work that has been done in the field of this thesis.

This master thesis is a continuation of research on the possibilities of using Multi-Agent Potential Fields (MAPF), that Johan Hagelbäck introduced in (Hagelbäck, 2009a), as an alternative AI solution for RTS games. In this thesis the focus will be on the possibilities of auto tuning MAPF and instead of using the Open Real-Time Strategy (ORTS) game engine as a test platform, this thesis will instead be using the very popular RTS game StarCraft and BWAPI, which will be described further in chapter 4.

The theory behind MAPF extends the knowledge of Multi-Agent Systems (MAS) and Artificial Potential Fields (APF), which is a concept originating from robotics. APF was first introduced in 1986 by Oussama Khatib(Khatib, 1986). Subsequent research by (Arkin, 1987) has resulted in new updated knowledge on APF by introducing a new concept called *Motor schema*, which serves as a means for reactive/reflexive navigation of a mobile robot and utilizes the super-position of spatial vector fields as a way to generate behaviour.

As (Hagelbäck and Johansson, 2008a) describes, many other studies concerning APF are related to navigation and obstacle avoidance, see e.g. (Borenstein and Koren, 1991) and (Massari et al., 2004). Together, these studies have given us a technique that is able to avoid numerous simple obstacles. Combined with a real-time autonomous obstacle avoidance approach, the result is improved and this makes it possible for robots at high speed to get through cluttered environments with highly complicated obstacles, according to (Borenstein and Koren, 1989).

Within the last decade, new interesting uses of APF in multi-agent environments have been presented. In 2002 (Howard et al., 2002) has demonstrated that an APF approach can be used to deploy mobile sensor networks in an unknown environment. Same year (Johansson and Saffiotti, 2002) and in 2004 (Röfer et al., 2004) showed that APF can be used in multi-agent systems like robot soccer. In 2004 (Thurau and Bauckhage, 2004) presented a bot for the FPS game Quake II, using APF to help with agent guidance to create human-like situation dependent movement patterns.

The theory behind MAS has been researched for many years and was earlier also known as distributed AI. The research in MAS started over 30 years ago at the University of Massachusetts Amherst (Lesser and Corkill, 2010). MAS have been used in many different AI problem-solving tasks e.g. in board games. In 1995 Kraus(Kraus and Lehmann, 1995) introduced the ideas about role-oriented MAS

and used it in the board game Diplomacy. Many articles have been written on MAS and in 2006 Johansson presented a proposal for a general MAS architecture for board games(Johansson, 2006).

In 2008 (Hagelbäck and Johansson, 2008d) show that role-oriented MAS also work combined with APF in RTS games and hereby they introduced the new concept multi-agent potential fields, and were the first to present a methodology on how to generally design MAPF-based solutions for RTS games (see section 3.2 for more information).

Interesting work has also been done on SSC in RTS games. (Laursen and Nielsen, 2005) use rule-induced timestamped game trees with machine learning methods for rating game states in the tree.

At the IEEE Conference on Computational Intelligence and Games 2010 (CIG) at ITU in Copenhagen, the first StarCraft AI competitions were held[1]. Unfortunately, only very few participated, but interesting enough, Ben Weber presented a StarCraft AI (called EISBot), which was able to play full StarCraft games, using reactive planning to manage and coordinate simultaneous low-level tactical actions and high-level strategic reasoning (Weber et al., 2010). Further, Johan Hagelbäck presented a MAPF-based StarCraft AI battling against EISbot. Later same year Ben Weber organised another StarCraft AI competition, namely at the AIIDE 2010[2]. This competition had impressive attendance with no less than 29 participants competing in 4 different tournaments from SSC combat scenarios to full blown StarCraft games. It will be interesting to see how different combinations of old and new AI techniques can help push the limits of the possibilities of AI research for RTS games in the upcoming AIIDE 2011, which will be hosted by the University of Alberta.

The MAPF-based StarCraft AI (called BTHAI) did not get to the top of the best performing AI's in the AI competition[3] in 2010, but maybe it would have performed better with use of optimization techniques for tuning the MAPF. The winner (Overmind) [4] however, actually used artificial potential fields and reinforcement learning to optimize their AI for specific situations, but only a demonstration and a relatively superficial article(H., 2011) about overall tactics has been produced.

To my knowledge, no one has yet described a detailed approach for developing a solution for auto tuning and optimisation of MAPF's for RTS games in academia before, which this thesis will do.

To be able to auto tune and optimize MAPF's for RTS games, some kind of optimization algorithm like e.g. evolutionary computing (EC) is needed(see e.g. 3.3). A lot of research has been done in the field of EC. Alan Turing was the first to propose the idea of evolutionary search back in 1948, but it took more than ten years before the first implementations of this idea was introduced, more precisely in the 1960's, and the implementations were called evolutionary programming (EP), genetic algorithm (GA) and evolutionary strategies (ES). In the early 1990s the newest evolutionary approach was introduced as genetic programming (GP). These four algorithms are today termed **evolutionary algorithms** (EA)(Eiben and Smith, 2003).

[1] `http://game.itu.dk/cig2010/?page_id=19`
[2] `http://eis.ucsc.edu/StarCraftAICompetition`
[3] http://eis.ucsc.edu/Tournament4Results
[4] http://overmind.cs.berkeley.edu/

Chapter 3

Methodology

This chapter describes the techniques and algorithms used to achieve the objective in this thesis, including Multi-Agent Potential Fields, Evolutionary Algorithms etc.

3.1 Artificial Potential Fields

As mentioned earlier, the underlying theory of the MAPF approach is Artificial Potential Fields (APF). APF is designed for controlling a robot/agent to navigate through environments with both static and dynamic obstacles, without colliding with them.

Basically APF works like a charged particle moving through a magnetic field. The environment of APF is split up in a grid of tiles where force fields are placed on specific key positions. Each force field either has a positive (attractive) or a negative electric charge (repulsive) and may be linear or tangential depending on the type of objects in the agent's environment.

The agent will need to have an overall goal position, which will have a positive electric charge and thereby attract the agent. APF is a reactive approach, where the agent will constantly have to remain extremely vigilant to new changes in the environment (e.g. other agents moving around) and therefore the entire route from start to the goal is not planned ahead.

In APF the agent will only look one step ahead to check which of the 8 surrounding tiles that returns the highest attractive value and then move to it. If the highest returned value is the tile the agent is currently on, the agent will stand idle. This could either mean that the agent has reached its target position, which is great because it is the purpose of the APF approach, or that the agent unfortunately has been stuck on a local optimum, for instance if the way to the goal is blocked by a large static object, e.g. a mountain. This issue with getting stuck on a local optimum seems to be very difficult to solve using solely APF, but (Thurau et al., 2004) has described a solution which lets the agent generate a negative trail on a specified number of previous visited tiles, similar to a pheromone trail in the ant colony optimization algorithm. The trail will have a repulsive force on the agent and thereby push the agent away from the local optimum.

According to (Hagelbäck, 2009b) most local optima can be solved with this approach, but APF still have problems in very complex terrains. As mentioned

earlier, the strength of APF lies in large dynamic environments with large open areas and less complicated terrains. Therefore, having complicated terrains where there is a risk that an agent can get stuck, is out of the scope of this thesis, while the focus will remain on auto tuning and optimisation of MAPF in SSC scenarios on less complicated terrains with large open areas.

3.2 Multi-Agent Potential Fields

When working with MAPF in a specific domain, there will always be some issues that the designer will need to handle. To structure this, Johan Hagelbäck has, in 2008, introduced a methodology where he describes how to design a MAPF-based solution for an RTS game (Hagelbäck and Johansson, 2008d).

MAPF for RTS games were first introduced by (Hagelbäck and Johansson, 2008d), where a methodology for how to design a MAPF-based solution for an RTS game is described. The methodology describes six phases for designing a MAPF-based solution, but the third phase about assigning charges has a weakness that could be improved. The problem is that the approach uses manually trial and error to try to find the best parameters for the potential field charges, which can be very time consuming and does not necessarily give an indication of whether an optimal or near optimal solution has been found. Therefore this thesis introduces a new phase called "'The process of tuning and optimizing the parameters of potential field charges"' which is added as phase 4 after the charges has been assigned to the objects. All the original phases and the new phase will be described below and will be followed in this thesis.

1. The identification of objects. The purpose of this phase is to identify all the objects that will effect or have some kind of impact on the agents. These objects can be either static (which means they will always remain unchanged), or dynamic, meaning that they can move around, change shape or change their type of impact (repulsive to attractive or vice versa) on the agents.

2. The identification of the driving forces (fields) of the game. The purpose of this phase is to identify the different force fields, which will be placed on tactical positions in the game world, to push the agents towards their overall goal. These fields can be repulsive forces generated by e.g. static terrain objects, attractive forces generated by e.g. the goal object (e.g. nearest enemy spotted unit or base). Force fields can also have a more strategic impact on the agents, like telling where the next goal is or more tactical impact, like coordinating how close the agents of a squad should be before attacking the enemy.

3. The process of assigning charges to the objects. The purpose of this phase is to specify the formulas for calculating each specific potential field value. All these formulas will be summed up to form a total potential field value, which will be used by the agents when choosing their next actions. High potential field values will attract the agents, while low potential field values will repulse the agents. The design of the formulas will depend on what type of game it is implemented in.

4. The process of tuning and optimizing the parameters of potential field charges. The purpose of this phase is to add an evolutionary approach to the already found formulas in the prior phase. An evolutionary algorithm can be used here to auto tune and optimise the parameters of the already found formula charges for the potential fields. This phase is my contribution to the original MAPF methodology and a more precise example of use can be found in section 5.4, and in chapter 6 the experiments and results from the approach will be presented.

5. The granularity of time and space in the environment. The purpose of this phase is to decide on the resolution of time and space. Resolution of space is a measure for the tile size in the grid of potential fields. Small tile size requires more memory, but will provide a high level of detail for the agents. Big tile size will require less memory and make the update of the potential fields faster, but will give a low level of detail for the agents and can lead to inaccurate informations, which can then lead to bad actions. The resolution of time is a measure for how far the agents can get in a game frame, depending on how much time the AI spends on thinking, and is consistent with the choice of tile size and the AI implementation in general.

6. The agents of the system. The purpose of this phase is to identify the objects that should become agents controlled by the AI, and decide which actions these agents should perform in different states, based on the total potential field values in the surrounding tiles.

7. The architecture of the MAS. The purpose of the last phase is to decide what the high level strategies should be. The different types of agents should have different roles. The agents could e.g. have roles for build order management (making decisions on which buildings/units to build when), management of where to build on the map, resource management, global map management (saving the locations of previously spotted enemy units, buildings and minerals etc.), exploration management (which units should be used for exploration, where to explore the map and when), squad management (overall tactics for controlling when to attack and optimize the attacks at enemy units) etc.

   It would make sense to use some kind of a reactive planner, like the EISBot StarCraft AI(Weber et al., 2010), for managing and coordinating all these agents' low and high-level decisions simultaneously. But this last phase is out of the scope of this thesis.

3.3 Evolutionary Algorithms

Evolutionary Algorithms (EA) are inspired by biology and is a fairly simple, though effective, simulation of natural evolution (survival of the fittest) in a computer, often used for making automated problem solvers. As mentioned in chapter 2, there are several different types of methods for EA, but all of them follow the same general outline and differ only in technical details, which will be explained later in this section. The general outline of EA is to simulate evolution by generating a population

of individuals (chromosomes) with as high fitness as possible. All the chromosomes contain some values (also called genes), which can solve a specific problem. The goal for EA is to ensure that only the fittest chromosome candidates, who solve the environmental challenges best, will survive and be selected for the next generation, while the worst performing candidates will be removed from the population, mutated or recombined with other chromosomes (cross-over). This optimization technique is an iterative process, where the EA tries to increase the quality of the chromosome values for each iteration, until a feasible or optimal solution is found or a predefined maximum limit of generations is reached (Negnevitsky, 2005; Eiben and Smith, 2003).

The different EA dialects differs from each other in the use of mutation and cross-over. In GA, the primary variation operator is cross-over and mutation is secondary, while for ES the opposite is the case, and sometimes mutation is the only variation operator for ES. For EP, mutation is the only variation operator, while for GP, cross-over is the primary operator, and sometimes the only operator(Eiben and Smith, 2003). In this thesis a combination of both types of variation operators will be tried, and using mutation solely will also be tried, to find out which combination gives the best results, when tuning and optimizing potential field parameters for an EMAPF-based AI in StarCraft.

Below, a walkthrough of the overall selection strategy for the EA used in this thesis will be presented.

1. A number of chromosomes equal to the population size are created.
2. Each chromosome is either initialized with random values between 0-1 or loaded from an XML file, containing earlier trained data.
3. All the chromosomes are saved into the population.
4. All the values from the chromosomes in the population will be copied and converted by multiplying them by a predefined value, so that each of them represents a potential field parameter.
5. All the potential field parameters are then added to the same amount of StarCraft AI solutions as the population size, which then adds them to each separate unit agent available in this specific scenario.
6. All these StarCraft AI solutions will now in turn play the same StarCraft map a fixed number of times, and the EA will then rank their performance, based on how well they play (fitness function).
7. The entire population is then sorted, so that the ones which performed best will be in top and the worst in the bottom of the population.
8. Now a survivor selection (repopulation) will be run including the following states:
   (a) The ten best performing chromosomes will be saved for next generation, to make sure not to delete any potentially optimal or near optimal chromosomes.

(b) Now two things can happen, either

i. A copy of all the chromosomes, except for the last 10, will be mutated, while the ten worst performing solutions will be deleted.

ii. A one-point crossover will be run on a copy of a specific number of the best performing solutions. The one-point crossover operation randomly chooses a crossover point where two parent chromosomes are split up. The first or second part of the first chromosome are randomly chosen to match the other chromosome's opposite part. This will then create one new offspring, where the other two parts left are thrown away. A number of the best performing chromosomes, equivalent to remaining rooms left in the next generation, will be copied and mutated.

(c) All these chromosomes will then form the new population for the next generation.

9. The newly generated population will then be used and the algorithm will continue from step 4 until a predefined number of iterations has been reached.

Chapter 4

Test Platform Setup

As mentioned earlier, the RTS game StarCraft: Brood War (version 1.16.1)[1] will be used as test platform for this thesis. There are many good reasons for choosing StarCraft as the test platform for this thesis, and one of them is that the game is extremely well balanced, but also because the complexity the game world offers, which makes the development of AI's very challenging. The AI is required to make tactical decisions for up to a very large number of units, which each have to navigate, defend and attack at the right moment in real-time.

4.1 StarCraft

StarCraft has three distinct and well balanced races, namely Zerg, Protoss and Terran. All the units are unique for each race.

Zerg units and buildings are all organic, can be produced very fast, are cheap compared to the other two races, and can automatically regenerate their own life. The weaknesses of the Zerg race is that they can only build on creep, their worker units (called Drones) die when they produce and mutate into a building, their units are weaker than the other races, and therefore the Zerg race rely on a great number of units to overwhelm the enemy.

The Protoss race has the most powerful and strongest units, which all have shields that automatically regenerate. Protoss worker units (called Probes) can build multiple buildings simultaneously and their Photon Cannons can attack both air and ground, as well as being detectors. The weaknesses of the Protoss race is that their units are more expensive and have a longer build time. Protoss buildings are useless if they have been cut-off from their energy supply (called Pylons).

The Terran (human) race has only military units with range attack and can provide significant benefits towards close combat units (with no range attack), if they are micromanaged correctly. Terran also has other benefits, e.g. all units can either be healed by medics or be repaired by the worker units (called SCV's). Buildings can also be repaired. Some Terran buildings can lift off and slowly fly away from the command center, if the Comsat Station is added, the Terran race can temporarily scan a small area, which reveals all enemy units and buildings in that area (both visible and invisible enemies) and the Terran buildings can be build everywhere.

[1] http://eu.blizzard.com/en-gb/games/sc/

The weaknesses of the Terran race is that their buildings burn down after a certain damage threshold has been reached, it can be harder to defend new Terran expansions compared to the two other races and it requires better micromanagement skills to control and make use of the benefits from the different units' special abilities.

I have chosen to use the units from the Terran race for testing the effect of using evolutionary algorithms for optimizing and tune the MAPF. The reason for this choice is mainly that the before mentioned range benefit and the demand for better micromanagement for controlling Terran units, which make the Terran race more interesting when the focus of this thesis lies on micromanagement and not on macromanagement.

But in order to be able to write an AI for StarCraft, some kind of framework is needed to communicate with the game through code. Since StarCraft is not open-source, a free open-source framework called Brood War Application Programming Interface (BWAPI) will be used to communicate with the game(group, 2011). BWAPI is written in C++, but several different wrappers for other programming languages exist, including C#.NET, Java, Python, PHP, Haskel, Ruby etc.(Weber, 2010b). Since my C# and Java skills are much better than my unmanaged C++ skills, I had to choose between using the C#.NET wrapper (bwapi-mono-bridge)(bwapi-mono-bridge group, 2010), or the java proxy version (bwapi-proxy-java group, 2011; Weber, 2010a). After testing both wrappers, I choose the bwapi-mono-bridge, because it looked more promising when as to features. The bwapi-mono-bridge included all the same methods as BWAPI, because it uses SWIG (http://www.swig.org/) to automatically generate the code wrapping files needed for connecting with BWAPI, while the java proxy version seemed to be hand-written and not all functionality from BWAPI was accessible. This choice later proved to be crucial for this thesis, because the java proxy was never updated to work with newer versions than BWAPI 2.6.1 (released in January 2010), while the C# wrapper was regularly updated. A lot of bug fixes and new features are added regularly for BWAPI, so I can highly recommend using BWAPI or a wrapper, which is updated regularly.

4.1.1 Physical Properties

A StarCraft map consists of a two-dimensional, tile-based map. The size of a StarCraft map can vary between 64×64 up to 256×256 build tiles. There are three different types of coordinates for a StarCraft map:

- Positions, which are measured in 1×1 pixels and have the highest resolution.
- Walk Tiles, which are measured in 8×8 pixels and can be used for units when walking around on a map (walkability data is available at this resolution).
- Build Tiles, which are measured in 32×32 pixels and are useful for build management (buildability data is available at this resolution).(group, 2011)

4.1.2 Unit control methods

All units in StarCraft can move in 8 different directions, and in order to be able to control and give the individual units orders, BWAPI includes the following applicable

methods:

rightClick

Takes a target *Position* as input, which refers to where a specific unit is ordered to move to. The *rightClick* method works like the right click in the GUI. After calling this method the unit will be in a move state: The *isMoving* method can tell if the unit in the move state. The *move* method is somewhat equivalent to rightClick method, but seems slower, so the rightClick method will be used in this thesis.

There is also an overloaded *rightClick* version which takes a target *Unit* as input. If the specified unit is a worker and targets a mineral patch (inherits from Unit), it can be ordered to gather resources from it. The methods *isCarryingGas* and *isCarryingMinerals* can be used to check if the unit is in a gathering state. Otherwise if the specified unit is a military unit and targets an enemy unit it will start attacking it. The method *isAttacking* can be used to check if the unit is in an attacking state.

attackMove

Takes a target *Position* as input, which refers to where the specific unit is ordered to move to, while attacking enemy units nearby (within max shooting distance). Using this method, the unit will move slower than using the *rightClick* method.

attackUnit

Takes a target enemy *Unit* as input and moves towards it until it is within max shooting distance.

stop

Orders the unit to stop. The method *isBraking* can be used to check if the unit is in the state of slowing down.

4.2 BWAPI challenges

The BWAPI and bwapi-clr-client is still a beta version with risk of errors and random crashes that has not yet been solved. To prevent too many crashes during the evolutionary runs, I have been forced to update to a newer version of BWAPI and bwapi-mono-bridge several times during the development of the MAPF-based StarCraft AI, because I found new errors in BWAPI that crashed the game. I reported the errors to the BWAPI page(group, 2011), and the BWAPI team then fixed them in a later updated version of BWAPI. There is very good support from the BWAPI team, but the problem with using two different platforms developed by different people (BWAPI and bwapi-clr-client), is that it takes a lot of time from reporting an error at the BWAPI team has fixed the error, and the bwapi-clr-client team has integrated it into theirs. In the meantime, I had to spend time on making some temporary solutions until the errors were fixed in BWAPI, and later integrated with bwapi-clr-client.

At first, I started developing my evolutionary MAPF (EMAPF) based AI in BWAPI 2.6.1, using AI Module DLL[2], which was the only one available in that version. But BWAPI seemed to be very unstable at that stage with too many crashes and it seemed impossible to write an AI that could be trained using evolutionary algorithms, because it is very essential that the game can be restarted at least 5.000 times without crashing, if the EA is to make good progress. Another issue was that it seemed very difficult to save data between games, because the AI Module was destroyed after each restart, i.e. there was no way in the code to save values like fitness through all the necessary restarts, without saving into a file and reloading it for the next generation, which would take unnecessary extra time.

Luckily, from BWAPI version 2.7.2 it was possible to use client communication instead of the dll module, which made it possible to save data between restarts. The problem was that the bwapi-mono-bridge needed to be updated and made compliant with the specific BWAPI version, before it would work. This did not happen before BWAPI version 3.2, so in the meantime I could only work on MAPF and prepare the code for evolutionary algorithms and hope for the best, namely that they would update the bwapi-mono-bridge.

When the bwapi-mono-bridge was finally up to date with BWAPI 3.2, I restructured and rewrote all my code to be able to work with the newest version, using a client communication process running in commando prompt, instead of a preloaded dll module. The time was well spend, because the evolutionary process could now run a lot faster, as it did not need to save data to a file after each restart.

But unfortunately, after short time I observed a new issue. When using EA, it is crucial that the game can be restarted after a match end (in BWAPI the method is called onEnd event), but it still crashed after 5-20 restarts.

I found out, that when calling the restart game method from the onEnd event, the game sometimes restarted twice very fast, and the second time the game would crash. This I reported to the BWAPI team.

But before this issue was fixed by the BWAPI team, I tried making a workaround in the onFrame event (called in each game frame), where a method was called every 60^{th} game frame, which checked if one of the teams had zero units left and then called *restart game*. The problem was that it was necessary to change the map settings for all test maps, so the game would never automatically end, this needed to be handled by code instead. This workaround did not fix the crashing completely; the game still crashed once in a while between 5-20 games, which of course is not enough.

Another solution I tried was setting the *auto_restart* = ON in the *bwapi.ini* file, which automatically restarts a game after a match once it is completed. This solution solved the problem with crashing, but was a lot slower than the restart game method in code, and sometimes it stopped at the final score board at the end of the game, waiting for user input.

In BWAPI 3.3 the issue with the *restartGame* method that was not working correctly during *onEnd* event and sometimes called twice, was fixed. After this fix, the evolutionary trainer could restart the game between 500-10,000 times, without any game crash. But occasionally the game still crashes with a memory leak error.

[2]The AI will be loaded into shared memory when StarCraft starts

In BWAPI version 3.4, there have been some attempts at fixing the memory leaks, but the same game crashing problem still exists.

The lead developers on the BWAPI team have the following comments to the problem: "'Figured it out. THIS IS A STARCRAFT ISSUE, so if you didn't use bwapi and just sat in single player and went to Restart Game a few thousand times, you would encounter this same error. The reason why this hasn't surfaced is because nobody has played a few thousand games without exiting the game once (and reported it)[3]."' I.e. normal use of the game would be to play the game for some time, maybe restart a couple of times and then exit the game again after use. The game has never been exposed to such extreme situations as an EA does, where it is indeed necessary to restart the game several thousand times without exiting the game once. The game is simply not designed for being restarted so many times, without exiting the game once in a while.

The BWAPI team is, in the time of writing, working on a possible patching hack to the problem, but until then, to prevent losing any crucial data on game crashes, I made it possible through a constructor in the evolutionary *Trainer* class, to save data after each generation, load an XML file with already trained data and start training from the generation the trainer had reached before the crash.

This thesis uses BWAPI 3.4 Beta and bwapi-mono-bridge 3.4a, and because of the mentioned bugs in StarCraft, there is still a risk for crashes, when using the *Trainer* class for many generations. The main method in the *Trainer* class will also need to be called when running the AI for StarCraft game play without training.

4.3 System Requirements

Below, all the programs and utilities required for running the trained Evolutionary (E)MAPF-based AI are listed:

- A full version of StarCraft Brood War updated to patch 1.16.1.
- Chaoslauncher Version 0.5.2.1 for Starcraft 1.16.1 or newer[4].
- BWAPI version 3.4 beta[5]. Follow the first 4 steps in the instructions of the read me file in the zip file.
- Microsoft .Net 3.5 Framework SP1 and/or Mono 2.8 or later [6].
- Microsoft Windows 7
  - Older versions of Windows might work with the AI, but have not been tested.
  - BWAPI uses DLL injection and is therefore unlikely to work in Wine, a virtual windows environment or Mac[7].

[3]urlhttp://code.google.com/p/bwapi/issues/detail?id=322
[4]`http://winner.cspsx.de/Starcraft/`
[5]`http://bwapi.googlecode.com/files/BWAPI_Beta_3.4.zip`
[6]`http://www.mono-project.com/Main_Page`
[7]`http://eis.ucsc.edu/StarCraftBWAPI`

- Microsoft Visual Studio 2010 is only needed if you want to see or change code in the solution (Not needed for running the AI). If you want to be able to compile the bwapi-native module inside the solution, Microsoft Visual Studio 2008 with SP 1 is also needed.

To start with copy the release folder from the DVD to your hard drive. To start the AI: run the Mono- or the Windows commando prompt (CMD) as administrator, then locate and run the StarcraftBot.exe (admin mode is very important, or Chaoslauncher will not be able to see the client AI). Be sure that the xml file "'!StarCraftUnitAgent.xml"' is located in the same folder, as the .exe file. Then run the Chaoslauncher as administrator. Be sure that the BWAPI Injector (RELEASE), Chaosplugin and optionally W-Mode (if StarCraft is to run in window mode) is added to the list of plugins inside of Chaoslauncher. Then just click start, and the game should start. Choose a single or multi player game, and start choosing one of the test StarCraft maps used in the thesis and set the type of game to "use map settings", otherwise both teams start with a command center, instead of only units.

For easier menu handling, the bwapi.ini file from the bwapi-data folder can be set to automatically start the game, using a predefined map.

Chapter 5

EMAPF for SSC RTS Scenarios

For this thesis I have implemented an Evolutionary (E)MAPF-based AI for playing SSC's in the RTS game StarCraft: Broodwar, using the proposed six phases from the methodology (Hagelbäck and Johansson, 2008d) plus the extra phase where the parameters of the potential field charges are tuned and optimized. As explained previously in section 3.2, this phase is added after the charges have been assigned to the objects.

This chapter will therefore describe the design and implementation of my MAPF-based solution using the 6 phases from the original methodology together with the new phase.

5.1 Identifying objects

The objects in the StarCraft world can be identified and split up in two categories, namely **static objects** like *map edges, walls, mountains, mineral and vespene geyser fields* and **dynamic objects** like *units (Workers, Marines, Goliath, Tanks, Hydralisks, Zerglings, Zealots etc.), buildings (Supply depots, Barracks, Factory, Star Port, Armory etc.).*

One might wonder why buildings are not marked as static and the reason is that many Terran buildings, e.g. the command center, can fly and move around in the world. Another reason is that buildings can go from not existing, to being build on a specific position on the map, to being destroyed, compared to e.g. mountains or other environmental objects, which will remain unchanged through the entire game. It is discussable if the mineral fields are static or dynamic, because they can never be moved, but they can be exhausted, when workers has gathered all the minerals from the field, and will then no longer collide with any units. At the same time, the mineral fields will remain unchanged if they are untouched or not exhausted.

The units are split up in own, enemy and neutral units, which will all have different kinds of fields, see section 5.2. Further, the AI's own units are split up in workers, army and squads. Most workers are not supposed to be added to a squad, unless mechanical units are included in the sqaud, then SCV's would be able to repair the mechanical units. In this thesis all workers are just workers, and will not be added to the squad. The army is a collection of squads. A squad is then a collection of military units, and can have a goal, e.g. find and attack nearest enemy.

There are only one squad in this thesis, and this contains all the AI's own military units in the map.

5.2 Identifying fields

In this thesis I have identified 8 different potential fields for the used MAPF-based AI, and these will be described below:

1. Maximum Shooting Distance (MSD) attraction. MSD refers to the maximum fire range for a specific type of the AI's own units, from where this specific unit will be able to start shooting and hitting an enemy unit, e.g. the Terran Goliath has a MSD of 5 build tiles against both ground and air units, if no upgrades attached. The main tactical goal field for each of the AI's own unit will be to move towards the closest enemy unit at each of the AI's own unit's MSD, which will have an attractive force, so that all the AI's own units will move towards this field, when their weapons are ready to fire. With this field the AI's own units will be able to surround the enemy unit/units at an appropriate distance, where the enemy can't harm the AI's own units, unless they have a better or the same MSD.

2. Weapon Cool Down (WCD) repulsion. Refers to the minimum length of time a unit needs to wait after a weapon has been fired to the weapon is ready to fire again, also called reload time. In many games (StarCraft in particular) there is often a very good balance between speed and power. A high WCD period (slow) often results in great weapon strength, while a low WCD period (fast) often results in weaker weapon strength. For instance, a Terran marine uses gauss rifles, or also called a coilgun, which has a very low WCD period and the weapon is much weaker compared to the Terran Siege Tank cannon.

   When the AI's own unit's weapons has been fired and is in the state of WCD (weapons needs to be reloaded), all enemy units will have a repulsion factor on the AI's own units, forcing them to retreat.

   The efficiency of the AI will depend very much on the coherence between these first two mentioned fields, but the rest of the fields are of course also important, or they would not be here. These two fields however, forms the main tactic.

3. Centroid Of Squad (COS) attraction, which refers to the centroid of all the units positions in a squad. See e.g. figure 5.1, where the squares are the units in the squad and the circle in the middle is the centroid. This field is used to attract the squad units to stay within a specific radius from the centroid of the squad , to minimize the risk of letting the AI's own units getting caught alone by a huge enemy force. At the same time, this field can help the AI's own units follow each other and thereby maximize the fire power on few enemy units at a time.

   Sometimes this field can give problems, when the unit at the top of the group is closest to an enemy unit, who is closer to the top of the map, while another of the AI's own units is in the bottom of the squad, is closest to an enemy,

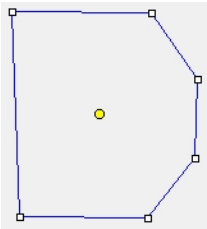


Figure 5.1: Centroid Of Squad attraction field

who is in the bottom of the map. This can make a conflict, if the centroid of the squad field has the same strength as the MSD attraction. The centroid of the squad will then attract the units to move there, while the two enemy units also will attract them. Then the AI's own units will end up standing idle, until the enemy takes an action

4. Center Of The Map (COM) attraction. This potential field will be used to attract the AI's own units to the center position of the current map. This will help securing that the AI's own units prefer the middle rather than the map edges or corners, as they could then easily end up being trapped by an enemy force.

5. Map Edge (ME) repulsion. This potential field will be used to repulse the AI's own units if they get too close to map edges or map corners and thus supports the COM field.

6. Own Unit (OU) repulsion. This potential field will have a repulsion effect on other of the AI's own units, if they get too close and will help make sure that the AI's own units do not get in the way of each other. If this potential field is tuned wrongly, the AI's own units would have a risk of not being able to retreat when in a WCD, because another of the AI's own unit's was standing in the way.

7. Enemy Unit (EU) repulsion. This potential field will have a repulsive effect on the AI's own units when they come within a certain radius from the enemy units' position.

8. Neutral Unit (NU) repulsion. NU's can be split up in two categories, namely critters and resources. Critters are small creatures which are on no team, cannot attack anyone and are randomly moving or flying around on the map. The main purpose of the critters in StarCraft is to give the game world a more natural feeling, but in this thesis they are added on some training maps to make the pathfinding and collision detection more difficult. The other type of NU is resources, which can be minerals or vespene geyser fields, which in normal full StarCraft games are used to buy buildings and units for. Figure 5.2 below shows all the neutral units in StarCraft. NU will have a repulsion effect on the AI's own units when they come within a certain radius from an neutral unit's position.

This potential field will not be updated using EA though, because the fitness function is focused on how well the squad performs according to killing the enemy and loosing as few units as possible (see more in section 5.4.3). The strength of this field will be assigned manually.

Figure 5.2: Neutral Units in StarCraft: Brood War

The strength of all these potential fields, except the NU potential field, will be tuned and optimized by the evolutionary algorithm (see chapter 5.4).

If these 8 potential fields are optimized and tuned well, it should be enough for the EMAPF-based AI to control the units to navigate around in a simple environment, and avoid colliding with static obstacles like minerals and vespene geyser. The units should also be able to adapt well to changes in the game scenario, like when dynamic obstacles be they own, enemy or neutral units like critters are standing in the way. Further, all units should be able to move towards their own maximum shooting distance of the enemy, attack when their weapons are ready to fire and retreat while their weapons are reloading.

5.3 Assigning charges to the objects

The specific potential field charge formulas presented in this section was found using trial and error. Microsoft Excel 2010 has been used to test that the formulas would give the wanted gradual repulsive or attractive effect. This was also helpful for the next phase, because it made it easier to find the range for parameters that would make sense for the different repelling and attractive fields. Some parameters should maybe have low values between 0-10, while for others it would make sense to use numbers from 0-1000. The specific charge parameter values will then be found by the EA (see section 5.4 and chapter 6 for more information).

Repelling fields can either gradually decline from zero to a large negative number or gradually decline from a large positive number to zero, the closer one of the AI's own unit's gets to a repelling field. Attractive fields can either gradually incline from a large negative number to zero or gradually incline from zero to a large positive number, the closer one of the AI's own unit's gets to an attractive field.

Both repelling and attracting fields can either effect units everywhere on the map or only affect units in a defined area around a specific field. Therefore some fields

can have a minimum and/or maximum range where it has an effect on the AI's own units within the range and when out of the range the units will not be effected by the field at all.

All dynamic units will generate a repelling field used for obstacle avoidance. The main driving field will be the MSD, which is the main goal position for all the AI's own units, when their weapons are ready to fire.

The returned potential field values generated from all the formulas are summed up, and the unit agents moves to the best surrounding potential field, i.e. the one with the highest returned value. This process will continue throughout the game.

Below, a detailed description of the specific field formulas will be presented.

Formula acronyms

The following formula acronyms will be used by many of the formulas, but all formulas have their own variables in code, so that they can be tuned and optimized separately. It is only for simplicity that they are named equally in this thesis, i.e. force is named ForceEnemyUnitsRepulsion, ForceMSD, ForceSquadAttraction etc. for the separate variables in code, but will just be named f for force in this thesis.

d	= distance from possible next position to object[1].
r	= range
rPerc	= range percentage of the maximum distance in a map.
f	= force
fStep	= force Step
eMSD	= enemy unit MSD
oMSD	= own unit MSD
MSDDiff	= oMSD - eMSD (difference between oMSD and the eMSD)
USize	= the unit type size.

The last letter of all the potential field formula abbreviations indicates if the force of the potential field is attracting (a) or repelling (r).

Maximum Shooting Distance (attraction)

The *Maximum Shooting Distance* field attracts the AI's own units, when their weapons are ready to fire (not in WCD state). The optimal position for the AI's own units will be in it's MSD to the nearest enemy unit. If the enemy unit has a shorter MSD than the AI's own unit and the distance to the enemy unit is less than the AI's own unit's MSD, the unit will get the force value minus the repulsion value returned from the *enemy unit repulsion*, see equation 5.3.5.

The *Maximum Shooting Distance* field is calculated as:

$$MSDa = \begin{cases} f - fStep \times (d - ownMSD) & \text{if } d > ownMSD, \\ f + d & \text{if } 0 < MSDDiff, \\ f & \text{if } d <= ownMSD. \end{cases} \quad (5.3.1)$$

[1]The object in d, can refer to one of the AI's own, enemy, neutral unit or Centroid Of Squad etc., it depends on the equation.

Centroid Of Squad (attraction)

The *Centroid Of Squad* field is attracting the AI's own units to the centroid of the squad and is useful for grouping spread units together. When all the units in a squad are grouped within a predetermined radius, they will not be effected by this field any more. If the AI's own units are outside of this radius, the units will get a lower potential value, forcing them to stay together, unless other charges have a stronger attraction or repulsion like if an enemy unit, with a repelling force, is moving in between and forcing the AI's own units to back away to keep a good distance to the enemy.

The *Centroid Of Squad* field is calculated as:

$$COSa = \begin{cases} f & \text{if } d < rPerc \\ f - fStep \times d & \text{else} \end{cases} \tag{5.3.2}$$

Center Of The Map (attraction)

The *Center Of The Map* field attracts the AI's own units to be within a predetermined radius from the center of the map. The reason for using this field is that it could be a good tactic to be near the map center and to gain control of the map instead of being caught in a corner somewhere. I.e. this field supports the map edge repulsion (see equation 5.3.8), to make the AI's own units to stay away from the edges.

For this field d refers to the distance from one of the AI's own units to the center of the map.

The *Center Of The Map* field is calculated as:

$$COMa = \begin{cases} f & \text{if } d < rPerc \\ f - fStep \times d & \text{else} \end{cases} \tag{5.3.3}$$

Weapon Cool Down (repulsion)

The *Weapon Cool Down* field is working very closely together with MSDa (see equation 5.3.1). This field is only activated when one of the AI's own unit's weapon is in it's WCD state, otherwise the MSDa will be activated. In this way, all units will move forward attacking the enemy while their weapon is loaded and retreat out of enemy range while reloading, which is a very effective approach (also often refereed to Hit and Run tactics, see chapter 6 for more information).

The equation WCDr works by checking if an enemy unit can attack one of the AI's own units at all. Some units can only attack ground, while others only can attack air. Therefore there will need to be a check for whether the enemy can reach the AI's own unit, otherwise this field will just return zero and have no influence. Afterwards it checks if the AI's own unit is in the enemy unit's MSD + an extra range, found by EA. If both statements are true, the same calculation as used for repulsion of the AI's own, enemy and neutral units will be used.

The *Weapon Cool Down* field is calculated as:

$$WCDr = \begin{cases} -(f - fStep \times d) & \text{if } CBA \ \&\& \ d <= eMSD + r, \\ 0 & \text{else} \end{cases} \tag{5.3.4}$$

CBA = own unitType can be attacked by enemy unitType.

Enemy Unit (repulsion)

The *Enemy Unit* field is repels the AI's own units from moving into an enemy unit's MSD, when the AI's own units have a higher MSD than the enemy. If the AI's own units has the same or shorter MSD than the enemy, the AI's own units will only be repelled from colliding with the enemy unit. This field combined with the attraction from the MSDa (see equation 5.3.1) then, gives the highest potential for the AI's own units in a radius around the closest enemy unit. Unless their weapons are reloading (see equation 5.3.4)

The *Enemy Unit* field is calculated as:

$$EUr = \begin{cases} -(f - fStep \times d) & \text{if } MSDDiff > 0 \ \&\& \ d <= eMSD, \\ -f & \text{if } d <= USize \end{cases} \tag{5.3.5}$$

Own Unit (repulsion)

The *Own Unit* field repels other of the AI's own units from a predefined range + the AI's own units unit type size and is used for obstacle avoidance.

The *Own Unit* field is calculated as:

$$OUr = \begin{cases} -(f - fStep \times d) & \text{if } d <= USize + range \\ 0 & \text{else} \end{cases} \tag{5.3.6}$$

Neutral Unit (repulsion)

The *Neutral Unit* field repels the AI's own units from a predefined range + the the AI's own units unit type size and is used for obstacle avoidance. This field is the only one that will not be tuned by EA. The reason for this is that the EA does not look at how fast the enemy is killed and does not directly compare how well the unit agents navigate thus, the EA is not able to optimize this potential field.

The following parameter values were found by trial and error and seems to work well for avoiding collision with neutral units.

f = 200
fStep = 1.2
range = 8

The *Neutral Unit* field is calculated as:

$$NUr = \begin{cases} -(f - fStep \times d) & \text{if } d <= USize + range \\ 0 & \text{else} \end{cases} \tag{5.3.7}$$

Map Edge (repulsion)

The *Map Edge* field repels the AI's own units when moving into a predefined range from a map edge. A StarCraft map has position (0.0) in the upper left corner, meaning that the y coordinate in a unit's position will be larger when moving downwards, and lower when moving upwards.

For this field there are some extra variables, not used by any of the other fields.

x	= own unit x position coordinate.
y	= own unit y position coordinate.
dLeftE	= distance to left map edge from current position.
dTopE	= distance to top map edge from current position.
dRightE	= distance to right map edge from current position.
dbottomE	= distance to bottom map edge from current position.
w	= map width.
h	= map height.

The *Map Edge* field is calculated as:

$$MEr = \begin{cases} -(f - fStep \times dLeftE) & \text{if } x <= r \\ -(f - fStep \times dTopE) & \text{if } y <= r \\ -(f - fStep \times dRightE) & \text{if } x >= w - r \\ -(f - fStep \times dBottomE) & \text{if } y >= h - r \end{cases} \tag{5.3.8}$$

5.4 Tuning the charge parameters

In this thesis, an evolutionary approach is added as an extra step to the original MAPF methodology. This section will describe how evolutionary algorithms can help optimize and tune the parameters. Later in chapter 6 the experiments and results will be presented.

J. Hagelbäck and S. Johansson have used manual trial and error to try finding the best parameters for the potential field charges in an ORTS game, but has also "'tried to use traditional AI methods such as genetic algorithms to tune the parameters of the bot, but without success"'(Hagelbäck and Johansson, 2008a).

When working with few potential field charges it is relatively easy to use trial and error on the parameters, but when having several potential field charges it is a lot more complicated to get an overview of how the different charges will effect each other and thereby make it almost impossible to tune the parameters manually to an optimal or near optimal solution. With the use of EA this operation will be simplified a lot. It is though important to make sure that the implementation of the EMAPF solution is working well, before trying to use EA to optimise anything, or there is a big risk of never getting any useful results from the optimisation.

This section shows how EA can be implemented into the MAPF methodology for SSC scenarios in RTS games, StarCraft in particular.

5.4.1 EA Problem definition

First it is important to find out what problem the EA is supposed to solve. The overall problem the EA has to solve for the EMAPF-based AI is to find the optimal or near optimal strategy for destroying all the enemy units on a SSC map in StarCraft: Brood War, surviving with as many units as possible and with as many hit points and shields left as possible. In many situations it is not likely that the units will survive battles without casualties, but finding a strategy that will make the best use of the units abilities, using the available potential fields, will be the main goal of the EA.

5.4.2 Parameters

In order to be able to solve this problem, the EA has 21 parameters available, which can be tuned and optimized, spread over the 7 potential fields: MSDa, COSa, COMa, WCDr, EUr, OUr and the MEr. The NUr is, as mentioned in section 5.3, not included in the EA.

The overall formula for all the potential field charges used for the EMAPF-based AI in this thesis is nothing but the equation of a straight line. But it is still important to understand how it works for the potential field charges. So an a explanation of how it will be used for potential fields will be presented here.

The parameters used can be categorised into force, forceStep and range. The force parameter determines what the highest or lowest possible value for that particular potential field can be, depending on whether it is a attractive or a repelling force field.

The force step determines the slope, i.e. how strong the field should be; in other words how much will the field value increase or decrease for every time a unit gets closer to the target goal.

The range can have a min and / or max value, where max determines how far away a unit can move before it is positively or negatively affected by the field, and the min range will determine how close a unit can get to a specific field before the unit is effected by the field. The min range is for instance used for all dynamic unit repelling fields, while the max range e.g. is used in COSa and COMa.

Some fields do not have a range, which indicates that these fields will effect the AI's own units everywhere on the map.

The distance variable can of course not be optimized, because it just indicates how far away a possible next neighbouring position of an own unit is to a specific potential field position.

5.4.3 Evaluation function (Fitness function)

For being able to compare different possible solutions for solving the problem, an evaluation function (fitness function) is needed.

To get a comparison as detailed as possible between the performances of all the chromosomes in the population, I have chosen to use a more detailed game score than the build-in score from StarCraft. My game score does not take into consideration which team is winning or loosing, but looks at the types of units both teams has

from the start, how many units that have been killed on the opponent team versus how many of the AI's own units that have died (the points depends on the unit types on both teams) and lastly total hit points and shields left in the AI's own squad.

Time is not directly used in the fitness score, but if a specific solution is spending too much time before one of the teams has destroyed their opponent, they will be punished by getting no points for survived units. The reason for that is to avoid fast units moving around on the map, fleeing from the enemy units forever, which would make the EA go into an infinite loop. The fitness function should always prefer solutions that try to eliminate the opponent, rather than solutions that allow units to retreat infinitely, because no reinforcements will ever arrive in any of the SSC training maps. In full StarCraft games, retreating would of course be a good tactic in some situations, where the opponent has the strongest army and the AI would then return to the base for reinforcements, but then a functionality for analysing the strength of the opponent's army would be needed, and this is out of the scope of this thesis.

The precise fitness score is calculated as follows:

$$\begin{aligned}\text{Fitness score} = &(Sum(TotalUnitBuildCount \times UnitTypeBuildScore) \\ &+ Sum(KilledOppUnitCount \times UnitTypeDestroyScore)) \\ &- Sum(TotalOwnUnitLossCount \times UnitTypeDestroyScore)\end{aligned}$$

If the StarCraft game doos not run out of the predefined time frame, the following calculation will be added to the total fitness score

$$\text{Fitness score} += TotalOwnHitpointsShieldsLeft \times 10 \tag{5.4.1}$$

The *buildScore* and *destroyScore*, which is different for each unit type, is taken directly from the StarCraft game.

5.5 Assigning granularity of time and space in the environment

I have used two main training maps for evolving the Terran units Goliaths and Vultures respectively (see figure 5.3). Both types of units were trained against the Zerg unit type called Hydralisks. The map size was 64×64 build tiles large, which is the smallest map size in StarCraft, and will support SSC's best, because then the units cannot run too far away from each other, and are forced to attack each other as fast as possible. Larger maps would simply give the units too much space to walk on.

Each unit will be able to move either horizontally, vertically or diagonally, in a total of 8 different directions or stand still for each action. To decide in what direction the unit should move to for best solve their goal, they use the 8 different potential fields. As input, all the necessary parameters chosen by the EA will be

passed to each separate potential field equation. The distance parameter will also be passed to each equation. When working with distances the highest resolution is used, namely 1×1 pixels, to get the highest level of detail. The distance is used to measure how far away a specific object is to a next possible position. To reduce the use of memory and computational time, only the nearest 8 fields around each of the AI's own unit's will be calculated and will only be called after each 5^{th} game frame.

None of the potential fields check for collision with environmental obstacles, because before the potential field equations are called, there will always first be a check on whether the next possible position would contain some environmental obstacle, which would collide with the AI's own units, or if the next possible position is outside the map. If one of those checks returns true, this field will not be passed to the equations as a possible position to move to. Both checks use walk tile size, which has a resolution on 8×8 pixels.

5.5.1 Training map units

In this section all the units used in the training maps will be described and there will be a discussion about why those units had been chosen. All the training maps is created with the StarCraft Campaing Editor, which is included when installing StarCraft. The training maps are also included on the enclosed CD. Figure 5.3 shows the units used in the training maps.

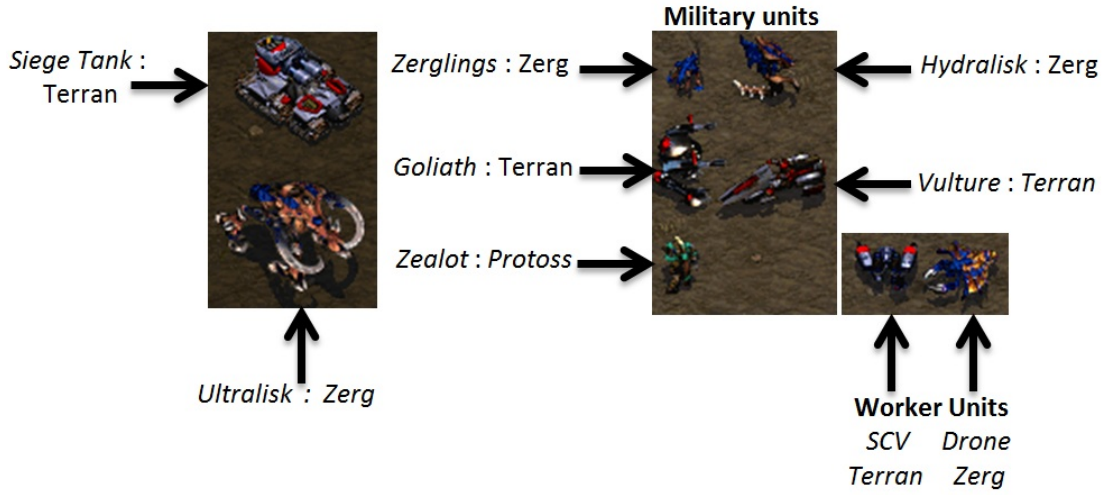


Figure 5.3: Military and worker units used in the training maps

Table 5.1 shows some information about the properties of the units used in the training maps. The MSD is an abbreviation for the maximum shooting distance (attack range) and the WCD is an abbreviation for weapon cool down (see 5.2 for more information). The attack property is the amount of damage that a unit's weapon deals per attack. Some units can both attack ground (G) and air (A), while other units only can attack either ground or air. Like the Goliath, some units deals more damage to air units than ground units and vice versa. Hit points is the amount of health a unit has. Large units has often much more hit points than smaller units, but are also often slower than smaller units. Shields (S) is only used by Protoss

units, which all has this extra amount of shield besides their hit points. The shield will need to be destroyed on the Protoss units, before they start loosing their hit points.

	Hit points	Armor	Attack	MSD	WCD	Sight
Terran Units						
SCV	60	0	5(G)	1(G)	15	7
Vulture	80	0	20(G)	5(G)	30	8
Goliath	125	1	12(G)/20(A)	5(G)/5(A)	22	8
Siege Tank	150	1	30(G)/70 Siege	7(G)/12	37/75	10
Zerg Units						
Drone	40	0	5(G)	1(G)	22	7
Zergling	35	0	5(G)	1(G)	8	5
Hydralisk	80	0	10(G)/10(A)	4(G)	15	6
Ultralisk	400	1/3	20	1(G)	15	7
Protoss Units						
Zealot	100	1 / 60(S)	16(G)	1(G)	22	7

Table 5.1: Overview of unit's properties.

Some of the abilities may differ between different StarCraft patches and also if units' have got upgrades. In this thesis, the units will not have any upgrades in the training maps though.

Unit's WCD is not completely the same all the time, because Starcraft randomizes the base WCD and adds a value between (-1) and (+2)[2].

In the first training map there are 9 Goliaths vs. 21 Hydralisks. In the other training map there are 9 Vultures versus 9 Hydralisks. On both maps there is also a worker unit for each team.

The two workers are added to the training maps to make it possible for the fitness function to differentiate the strength of the different solutions (chromosomes) better and to give some balance between aggressive and defensive behaviours. If the AI's own military units are too frightened and choose to flee too often, they will easily loose their worker, and spending too much time on fleeing will, as mentioned earlier, remove the extra points for the remaining hit points, and they will not be able to get points for the enemy military units or the enemy worker. On the other hand if the AI's own military units are too aggressive they may loose too many of their own military units.

Finding the right balance by adjusting the strength and range of the different potential fields is a main goal for the EA.

I also made some other small training maps for testing the generality of the AI, to see how well the AI will handle other types of units, which it has not been trained against. The AI will also be running on both sides, to see how well the AI performs using for instance Zerg instead of Terran (as it has been trained for), but also to make sure that the teams are not unfair for the build in AI playing the Zerg race in the training maps.

[2] `http://code.google.com/p/bwapi/wiki/StarcraftGuide`

The number of units on both sides in the battles have been chosen considering that the units owned by EMAPF-based AI playing as Terran should have less or equal total number of hit points than the build-in StarCraft AI. For the first training map the total hit points of the 9 Goliaths is 1125, while the 21 Hydralisks has 1680 hit points, at the same time all Zerg units regenerate hit points continuously and shoots faster, because of the low WCD. So the advantage is certainly on the Zerg team, but in return the Goliath unit (as seen in table 5.1) has a slightly longer MSD and attack strength, but they are also outnumbered.

The only way the Terran team should be able to escape alive from this battle, would be for the AI to employ very good micromanagement and it is therefore a requirement that the EMAPF is designed well for the AI and afterwards tuned and optimized to solve the task, which is the purpose of the EA.

For the second training map, with the 9 Vultures against the 9 Hydralisks, the total hit points of both teams is 720. The strength of the Vulture unit is definitely their speed, which e.g. can be used to scout the map, for example for checking where the enemy base is or if the enemy base has added new expansions. Their speed also make them useful for harassment. For instance if they move in full speed into an enemy base behind their resources and kill some workers, before the enemy will have a change to hunt them down, then retreat again (hit and run tactic). Vultures give most damage to small units and is therefore most effective against small melee units like Marines, Zealots, Zerglings and workers as mentioned before. The weaknesses of the Vultures is specially that they can only attack ground units, so all air units constitute a great danger for Vultures. The speed of the vultures, can also be a weakness when they are controlled by humans, because it makes them more difficult to control, and just a few mistakes and they can be dead, while they do not have much hit points and no armor.

I have chosen Vultures against Hydralisks, because Vultures is normally not very good against ranged attack units, but it could be interesting to see how well it could end up performing against the Hydralisks, if the EMAPF is designed well and tuned by an EA. The Vultures have slightly better attack than the Hydralisks, but only give 50 % damage to medium sized units, like the Hydralisks. The Vultures have half the attack speed (higher WCD) compared to the Hydralisks. The most interesting thing about them, though, is that they have a slightly longer MSD than the Hydralisk, and combined with their speed, the EA has a possibility to tune the potential fields, so that the Vultures can be microed well and maybe end up winning.

5.6 The agents of the AI system

The AI in this thesis contains two overall types of agents. The first one is a unit agent and all the AI's own units are attached to each separate Unit Agent. Each Unit Agent has an overall target goal which is decided by the other type of agent available for the AI, namely the Tactical Assault Agent. The Tactical Assault Agent contains all the military units in a specific squad, and gives the overall target goal for each Unit Agent. The overall target goal in a SSC map will always be the position of the nearest military enemy unit or enemy worker unit who is repairing other units. If there are no military units or repairing workers, then the target goal

will be set to the position of the nearest enemy worker unit, or the game will end and the EMAPF-based AI has won.

All the Unit Agents will be ordered to find the shortest route to the nearest enemy unit, by using it's 8 potential fields. This also means that the Unit Agents, if their contained unit type has a higher MSD than the enemy unit it has been ordered to attack, will try to be near it's own MSD, instead of near the target goal position. The target position is only used as an indicator for where the enemy unit, which is to be attacked, is positioned.

The Unit Agent has an extended version of each unit type, so that each type of unit, e.g. Tanks, Vultures etc., could use their special abilities like Siege mode for tanks and mines for Vultures. Because of lack of time potential fields for their special abilities have not been implemented. The Vultures will therefore only use their attack skill in the training maps. If the AI should be extended to work for full scale StarCraft games, it would of course make sense to have som potential fields for e.g. the Vultures, so that they were attracted to put their mines into choke points, where it is highly likely that the enemy would walk across to get to the AI's own base. For the tanks, it would of course make sense to have high potential fields for Siedge Mode, when they are in the MSD of the nearest buildings, and heavy support from for instance Golliaths would of course be needed for the Tanks, since they cannot attack air and are very slow.

5.7 The MAS architecture

The overall high level strategy used for the AI in this thesis is to find and attack the nearest opponent units, and not stop until everyone is destroyed. There is no choice of retreating if the enemy has too many units. This strategy would of course not be enough for an AI for a full StarCraft game, but this thesis focuses on SSC where you do not have to think about base building, when to attack the opponent player, or how many and which units to build yo be able to win against an enemy squad or base.

Chapter 6

Experiments and Results

This chapter follows up on section 5.4 and will describe and evaluate several experiments with different EA configurations for auto tuning and optimisation of EMAPF for the AI in SSC scenarios in StarCraft. For all the experiments the EA will be used to optimize the 21 parameters for the 7 potential fields presented in section 5.3. In correlation with the EA methodology all the parameters are initially loaded with random numbers into each chromosome in a population and later measured by the fitness function, that determines how well the AI performs against the build-in AI, with the assigned specific parameters for the potential fields. The potential field parameters used for the 10 best performing AI's will survive, while the rest will be deleted or run through some variation operators like mutation and/or one-point cross over, to create new combinations of parameters that will hopefully make the AI perform better against the build-in AI. In all experiments this optimisation process will go on for 25 generations.

All experiments have been run on a 2.4 GHz Intel quad-core CPU with 4 GB of RAM on Windows 7 x64. With this computer configuration the evolutionary experiments took between a couple of hours for the fastest runs to around 24 hours for the longest runs. For each experiment I will run StarCraft in max speed to be able run as many games as fast as possible.

6.1 Initial experiments

In order to be able to measure different types of EA configurations for the MAPF-based AI, I will start out making two initial tests.

First I will make a scatter plot by running 100 different AI solutions against the build-in AI with totally random generated potential field parameters and plot their fitness scores to determine the *variability* in their performances. The scatter plot showing the *fitness variability* is plotted in figure 6.1.

Figure 6.1 shows that there is a big difference between the best and the worst performing random generated potential fields parameters, but most of the samples is between -2000 and 4000, which seems to be a fine fitness variation.

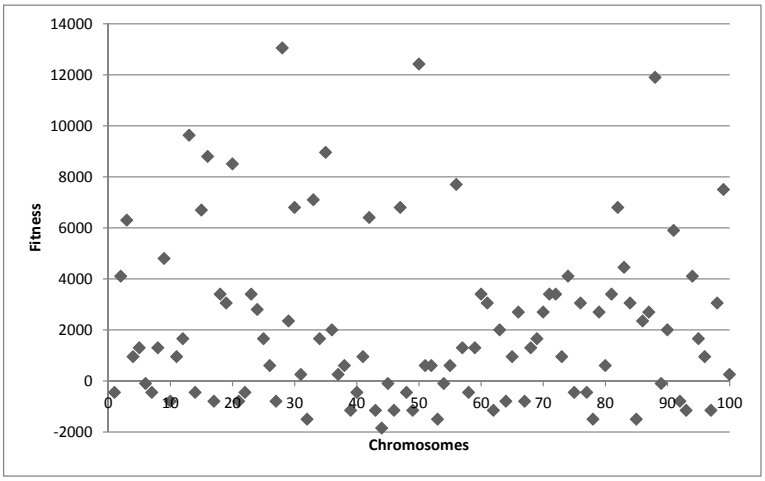


Figure 6.1: A scatter plot showing the fitness score for 100 different AI solutions

Section 5.5.1 mentions the base values for some different units' attack strength, but these values can vary with a small random value for each attack. The same thing is true for the WCD, which also is randomized slightly after each attack, which makes StarCraft non-deterministic. Therefore, in the second initial test I will explore how much *fitness noise* the randomness in StarCraft will produce if I run 10 different AI solutions 10 times against the build-in AI with totally random generated potential fields parameters. Table 6.1 shows the standard deviation, T-Test, average, maximum and minimum fitness score calculated based on the 10×10 runs, which can help examine how much fitness noise there is and to see if there is a risk that the noise could damage the reliability of the fitness function, which will be used in all the experiments.

Round	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
1	250	-1150	6700	11670	2100	950	7500	-450	13110	-1850
2	-1500	-450	8960	12960	7100	250	6000	3750	11820	-100
3	-1150	950	2700	4800	2000	1300	8750	600	7600	-1150
4	-100	600	7500	12870	-1500	600	14520	1300	7200	-1500
5	-800	2350	3050	12060	7850	950	10920	950	9230	-1150
6	950	-1500	2350	8400	8030	6700	7500	950	2700	-1850
7	-100	-1500	6700	5150	2000	-450	7850	1650	950	-1150
8	250	-1150	2350	2700	950	250	12620	2000	11310	-450
9	2000	-1150	1650	12030	-450	1300	5150	3050	7600	-1850
10	-100	2350	6700	12050	-1150	1650	3050	250	13620	-800

Sample Number	10	10	10	10	10	10	10	10	10	10
Average	-30	-65	4866	9469	2693	1350	8386	1405	8514	-1185
Max.	2000	2350	8960	12960	8030	6700	14520	3750	13620	-100
Min.	-1500	-1500	1650	2700	-1500	-450	3050	-450	950	-1850
Standard Deviation	1014,40	1525,17	2682,91	3886,55	3667,85	1979,34	3473,73	1267,86	4227,80	605,09
T-test		(c1,c2)	(c1,c3)	(c1,c4)	(c1,c5)	(c1,c6)	(c1,c7)	(c1,c8)	(c1,c9)	(c1,c10)
		0,95258	0,00019	0,00002	0,04626	0,07083	0,00002	0,01236	0,00010	0,00759
			(c2,c3)	(c2,c4)	(c2,c5)	(c2,c6)	(c2,c7)	(c2,c8)	(c2,c8)	(c2,c10)
			0,00017	0,00001	0,04848	0,09126	0,00001	0,03118	0,00008	0,05229

Table 6.1: Fitness noise for 10 chromosomes in 10 rounds

The t-test in Table 6.1 shows that with 10 trials, most of the random generated potential field parameter samples are stochastic independent, except for the first two chromosomes. The two first chromosomes seem to follow almost the same distribution: the t-test value is very close to 1, the average is almost the same and the minimum fitness values are equal. All the other samples seem to be stochastic independent. I will be using a population size of 40, so there should be enough randomness in the parameters to give some useful results. Half of the samples have a standard deviation of more than 2500, which in my opinion is relatively high, i.e. even if the EA finds one good combination of potential field parameters that seems to win with many hit points left in one match, the AI might lose in the next two matches, or win narrowly. This means that running the EA with just 1 trial per chromosome will not necessarily find the optimal or near optimal combination of potential field parameters, but can still give an indication of which combinations of parameters are good and which are bad.

6.2 Main experiments and results

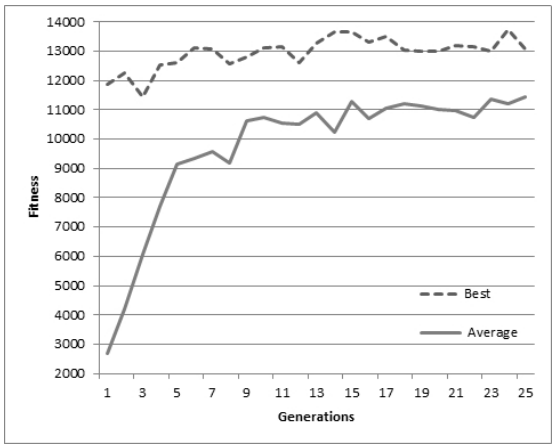
In order to measure which EA configuration that provides the most reliable and best results I will present 5 main experiments and 2 secondary experiments, all with different EA configurations, using the same fitness function to evaluate the results from the training. Table 6.2 shows the 7 different experiments.

Experiment	Unit type	Trials	Generations	Chromosomes	Variation operators
1	Goliath	10	25	40	Mutation and one-point crossover
2	Goliath	10	25	40	Mutation
3	Vultures	10	25	40	Mutation
4	Goliath	2	25	40	Mutation
5	Goliath	1	25	40	Mutation
Older experiments run before update to BWAPI 3.4					
6	Goliath	1	100	80	Mutation
7	Goliath	2	100	40	Mutation

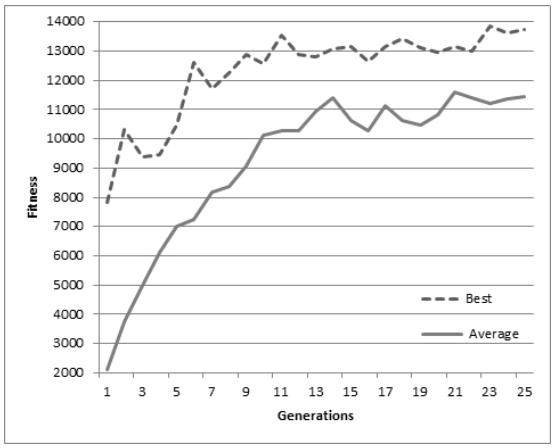
Table 6.2: Experiments

The unit type specifies which units the EMAPF-based AI has. The map called **GoliathsHydralisks3.scm** is used in all the experiments where the unit type is set to Goliath and have 9 Goliaths against 21 Hydralisks. Experiment 3 uses map **VultureVsHydralisks.scm** with 9 Vultures against 9 Hydralisks (see section 5.5.1 for more information on the units). The results from the training in the first five experiments can be seen in figure 6.2.

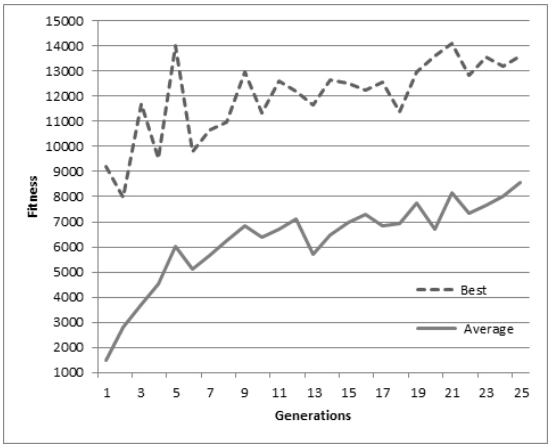
After these experiments I will take the best performing AI solution from the first generation (with totally random parameters and no evolution) against the best performing AI solutions from the last generation, from one of the previous experiments and run them 30 times each against the build-in AI in the **GoliathsHydralisks3.scm** map. This experiment will show if the trained AI solution has actually significantly improved or not.



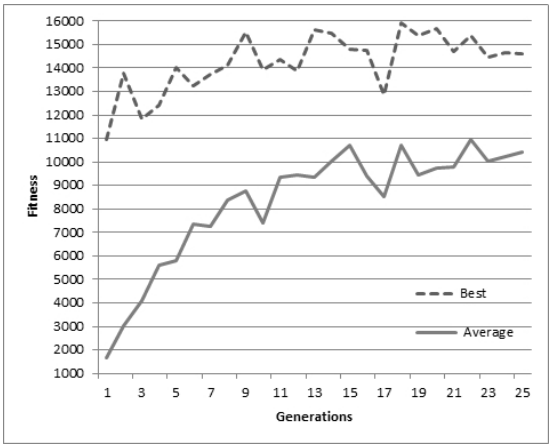
(a) 10 trials for 25 generations with 40 chromosomes using mutation solely (Goliaths).



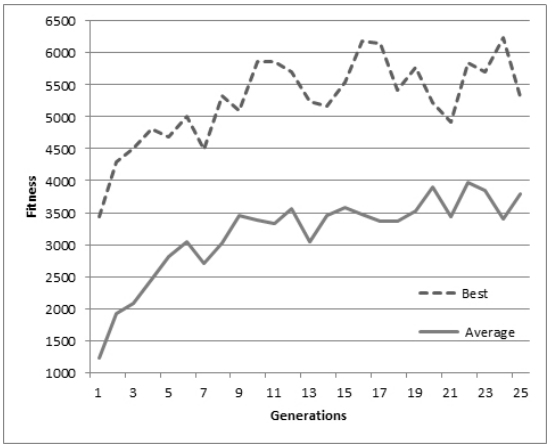
(b) 10 trials for 25 generations with 40 chromosomes using mutation and one-point crossover (Goliaths).



(c) 2 trials for 25 generations with 40 chromosomes using mutation solely (Goliaths).



(d) 1 trial for 25 generations with 40 chromosomes using mutation solely (Goliaths).



(e) 10 trials for 25 generations with 40 chromosomes using mutation solely (Vultures).

Figure 6.2: Normalized performance graphs for training of potential field parameters

Figure 6.2 shows five different configurations of EA. All the graphs show two curves, one for the average performance of the entire population of chromosomes, and the second curve shows the best performing chromosome. Each chromosome represents an AI solution with each of their own unique combinations of potential field parameters.

The chromosomes have 21 genes, one for each parameter. Each gene starts with a random value between zero and one and each gene has a multiplier assigned, which is used to convert the gene values to real potential field parameters (in theory also called mapping from genotype to phenotype) and back again. The reason for having both the gene values and the real potential field parameters, is mainly that it is a lot faster and easier to apply variation operators like mutation to a list of floats, which is the gene's data representation in this thesis, than it is applying variation operators to an object where all the values are saved in specific named properties. Another reason is that the different gene values will need to have different multipliers for some of the potential field parameters, which would make the process of applying mutation almost impossible, if all gene values were not within the same scope.

In all the experiments the *force* values have a multiplier set to 1000, i.e. if there is a gene with value 0.68 then the corresponding potential field parameter will be 680. The *forceStep* values all have a multiplier set to 10, while the *range* values each has a multiplier set to either 512 or 100 for the *ranges* with percentage.

In my implementation, the mutation operator mutates a random number of the gene values in each chromosome, with a random number drawn from a Gaussian distribution, with a mean set to 0 and the standard deviation set to 0.1.

The other variation operator, one-point crossover, is only used in the experiment shown in figure 6.2 graph (b) and is implemented so that a copy of a specific number, which in this particular experiment is 40, of the best performing chromosomes will run through the one-point crossover operator. The operator then takes two of the chromosomes at a time, splits them at a random position, combines the two parts randomly, and returns one of the new combined chromosomes. This operation is in this particular experiment applied to a copy of the complete population, and in return we get 20 new combined chromosomes for the next generation.

The 10 best performing chromosomes will always survive to the next generation in all the experiments, and for graph (b) only the last 10 chromosomes will be mutated, while for all the other experiments, mutation is applied to the last 30 chromosomes in the population.

6.2.1 Discussion

If we take a look at the average performance curves in all the experiments, they seem to have the same tendency: the curve rices rapidly in approximately the first 10 generations and then it rises more slowly, corresponding with the curve for best performing chromosomes that seems to be approaching an optimal or near optimal solution between 10 to 25 generations. The average curve can of course never rise higher than the best curve and will unlikely never converge with it; even if all the chromosomes were totally equal, there would be no guarantee that they all would get the same fitness score, because StarCraft is non-deterministic and as we saw in the fitness noise table 6.1 there can therefore be big differences in the performance

for the same configured AI solution from match to match.

If we compare the results from figure 6.2 with the two graphs in the appendix, it looks like there is very little or no improvements after around 25 generations. The results from the appendix, was run before updating to BWAPI 3.4 and after some smaller bug fixes in my AI though, but it is unlikely that the update and the small bug fixes would have made a difference to the tendency with almost no improvements after the first 25 generations, which is why I have only used 25 generations for all the newest experiments seen in figure 6.2.

At first sight, it looks like figure 6.2 graph (d) where only one trial for each chromosome in a generation gives the best results, when evolving a MAPF-based AI for StarCraft. The reason for this observation is that it is a lot faster running only one trail, than running several trials for each chromosome in a generation and at the same time it seems to find the highest fitness scores at a maximum of 16000. If compared to the normalised results in graph (a), (b) and (c) using respectively 10 and 2 trials per chromosome, the results gives a maximum around 14000 in fitness score. The problem is though, that with only 1 trial you cannot be sure that the result you get is valid, and that you actually has found an optimal or near optimal solution, and thereby will get the same result if the AI with the best optimized potential fields is run again. Even if the AI tries to do exactly the same in several consecutive games, you cannot be sure to get precisely the same results every time, because, as mentioned earlier, StarCraft is non-deterministic. StarCraft has, as mentioned in section 5.5.1, some base values for how hard different units attack, but this value can vary with a small random value for each attack. The same thing is true for the WCD, which will also be randomised slightly after each attack, while the MSD value for instance will remain the same all the time.

So from the 4 graphs I would recommend using at least 2 trials per chromosome, but 10 would of course be better, but if time is taking into account, 2 trials is much faster, and can also give okay results, but you could still not be sure that the one with the highest fitness in two games will also have the highest score in the next two games.

Another approach could be to start out in a couple of generations with only 1 trial per chromosome, and later set the number of trials to 2, 5 or 10 per chromosome, because the 1 trial approach could be helpful to pick out a sample of the possible best chromosomes, but with 1 trial we cannot be sure which of the maybe 10 best chromosomes is really the best. Here we need some more trials for each of the chromosomes, to be more sure. This approach could also be combined with training on several different maps with different challenges in them, e.g. different kinds of enemy units, but this was not possible with the version of BWAPI used in this thesis.

Figure 6.2(e) proves that it is not just luck that the EA has a positive effect on the behaviour of the Goliath unit type, but that the EA can be used to tune almost all types of units, for instance also Vultures, which were used in this experiment.

It can be a lot more difficult to evolve units which are much slower than the enemy units with the current 8 potential fields used for this MAPF-based AI though, because the two potential fields, MSDa and WCDr, will prompt the units to try to retreat when they are reloading (weapon cool down state). But it is not possible to retreat, if the enemy is much faster than the AI's own units.

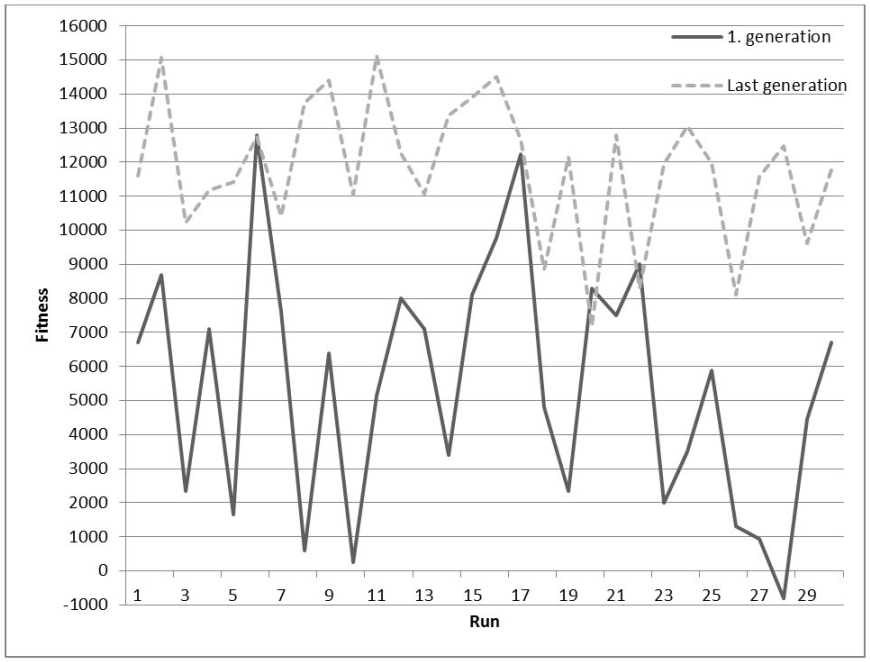
To solve this possible issue, a slow unit would probably need to be trained to-

gether with other types of units, so that they can try to always stay behind the other of the AI's own units. It is also very rare for anyone to attack the enemy with slow units alone. The Protoss *Reaver*, which is the slowest unit in the game, could for instance be trained together with the Protoss transport unit, called *Shuttle*, so that they together could harass the enemy by letting the Shuttle drop the Reaver, let the Reaver shoot and then pick it up again with the Shuttle right after the shot and then use WCDr to retreat, making it very difficult for the enemy to get any hits on the Reaver or the Shuttle.

With the current 8 potential fields it would be possible to train the Terran *Siege Tank* to use the same behaviours with the hit and run tactic like the Goliaths and Vultures use in this thesis, but to take full advantage of the *Siege Tank* the AI would need some more potential fields for being able to use its Siege Mode in tactically clever ways. Siege Tanks are very vulnerable to air units, because they can only attack ground units and are very expensive too, so to attack with them alone would often prove to be kind of a suicide mission. But if they are heavily guarded and the nearest enemy buildings e.g. are within range (MSD state), they can cause excessive damage to enemy buildings, when in Siege Mode, but they are very dependent on the support from especially anti air military units like Goliaths, because it takes a very long time for them to return to non Siege Mode, where they can move and retreat again. The Siege Tanks should also have another potential field when defending a base, that would force them to move to a near tactical hiding position in the base. This position could for instance be behind bunkers and anti air *Missile Turrets*, where it would then go into Siege Mode, because Siege Tanks in Siege Mode are also very effective against large groups of units trying to attack the base, since they cause splash damage to all units in a specific area around the unit that has been hit.

6.3 Competition and observations

To compare the actual difference in performance between the first generation and the last generation, 30 matches between the fittest AI from the first generation against the build-in AI and 30 matches between the fittest AI from the last generation against the build-in AI has been run, to see if there would be significant differences in their performances. The results can be seen in figure 6.3. The map used in this competition is called "'Goliath9VsHydralisks21.scm"' and a video showing the trained AI against the build-in AI can be seen on YouTube[1].



Sample Number	30	30
Average	5462	11817
Max.	12800	15120
Min.	-800	7190
Standard Deviation	3543,72	2019,84
T-test		(c1,c2)
		4,85121E-11

Figure 6.3: Fitness scores for best chromosomes from the first and last generation.

Overall, it is easy to see that the AI with trained potential field parameters has significantly improved in performance and is much better than the AI with totally random parameters, but it happens that the AI with untrained potential field parameters gets higher fitness score, than the worst trials from the trained AI. If the trained AI is to perform better, the Goliath units would need to survive with more hit points. This would require some SCV's with their own potential fields making them able to follow and repair the mechanical units during the attack.

[1] `tinyurl.com/9goliathsvs21hydralisks`

Table 6.3 presents 11 AI videos showing different tactics, units and scenarios for the trained EMAPF-based AI developed in this thesis. All the videos have a YouTube link.

Video nr.	Map name	Units	Link
1	Goliaths3Zealots6.scm	3 Goliaths (EMAPF) vs. 6 Zealots	http://tinyurl.com/3GoliathsVs6Zealots
2	Goliaths3Zerglings20.scm	20 Zerglings (EMAPF) vs. 3 Goliaths	http://tinyurl.com/ZerglingsVsGoliaths
3	Goliaths3Zerglings20.scm	3 Goliaths (EMAPF) vs. 20 Zerglings	http://tinyurl.com/StayTogetherTactic
4	Goliaths3Zerglings20.scm	3 Goliaths (EMAPF) vs. 20 Zerglings	http://tinyurl.com/IndividualTactic
5	Goliath9VsHydralisks21.scm	9 Goliaths (EMAPF) vs. 21 Hydralisks	http://tinyurl.com/9GoliathsVs21Hydralisks
6	Vultures9VsHydralisks9.scm	9 Vultures (EMAPF) Vs. 9 Hydralisks	http://tinyurl.com/9VulturesVs9Hydralisks
7	Vultures9VsHydralisks9.scm	9 Vultures (EMAPF) Vs. 9 Hydralisks	http://tinyurl.com/VulturesRunningInCircles
8	Vultures9VsHydralisks9.scm	9 Vultures (EMAPF) Vs. 9 Hydralisks	http://tinyurl.com/VulturesWaitForEnemy
9	Goliath9VsHydralisks21.scm	Multiplayer 9 Goliaths (EMAPF) vs. 21 Hydralisks	http://tinyurl.com/TrainedZergVsNotTrainedTerran
10	Goliath9VsHydralisks21.scm	Multiplayer 9 Goliaths (EMAPF) vs. 21 Hydralisks	http://tinyurl.com/TrainedTerranVsNotTrainedZerg
11	TerranMixVsZergMix.scm	3 Goliaths, 2 Tanks & 4 Vultures (EMAPF) Vs. 9 Hydralisks 2 Utralisks	http://tinyurl.com/TerranMixVsZergMix

Table 6.3: Info and links to the EMAPF AI videos

Table 6.3 video (10) shows a match where the AI with trained potential field parameters is playing as Terran (9 Goliaths) against the untrained AI playing as Zerg (Hydralisk). Table 6.3 video (9) is the same match but where they only have switched sides, so that the trained EMAPF-based AI plays as Zerg. The trained AI has only been trained with the Goliaths, but in video (9) you can see that the AI can also play with other types of units and other races. But it seems that the trained AI, when playing as Zerg, is afraid of getting too close to the edges, even when the Goliaths are hiding there, which is a problem because the Hydralisk cannot hit the Goliaths then. So if the AI had had a chance to tune its parameters for Hydrlisks too, it would most likely have performed better.

Overall, the trained AI, when playing as Terran (Goliath), wins a great victory against the untrained AI. Only 1 Goliath is lost, while it kills all the 21 Hydralisks. In video (9) where the trained AI plays Zerg is losing, but manage to kill six Goliaths out of nine. So from these two matches and the results seen in figure 6.3, I conclude that the best performing AI after 25 generations with trained potential field parameters has significantly improved in performance since the best performing AI from the first generation with only random parameters. Video (5) is the same scenario, only here the trained AI plays against the build-in AI, which is playing as Zerg (Hydralisks). The trained AI wins, losing only two units.

Video (1) shows the trained AI with three Goliaths beating six Zealots played by the build-in AI, without losing one single Goliath. One important thing about this match is that the Zealots have 100 hit points and 100 shields, which slowly regenerate over time, whereas the Goliath has 125 hit points, but does not regenerate anything, i.e. in total the Zealots have 600 hit points and 600 shields, whereas the Goliaths only have 375 total hit points. The reFason why they can win is that the trained AI exploit the advantages the unit type Goliath has, which is that they have much longer MSD and thus almost never come very close to the Zealots.

Video (3,4) shows the trained AI with three Goliaths against 20 fast Zerglings played by the build-in AI and video (2) shows the same match only with switched teams. The trained AI wins all 3 matches against the build-in AI. When playing as Zerg, only nine Zerglings were lost. In video (4) the AI uses a tactic where the force in the potential field COSa is very weak, meaning that the units never try to group together in the squad. All the units walk around alone, splitting the Zerglings in smaller groups, and this tactic seems to work, because they kill all 20 Zerglings without losing any units. When I made the COSa potential field, I thought that the trained AI would use this field to group its units together, so that they could focus their total fire power on one enemy unit at a time, so I tried manually to change the parameter values for COSa in a copy of the trained AI, seen in video (4), to see the effect of having a strong COSa. Video (3) shows the result. The three Goliath units started to move together, before they went on in a closer group and engaged with the enemy. With this tactic, the three Goliaths still win, but lose one out of three Goliaths. The main reason, as I see it, is that they have less different positions to go to, while all the Zerglings are following the three Goliaths, so sometimes they cannot move long enough away from the Zerglings, to avoid getting hit. At the same time it is easier for a large group of 20 units to surround three Goliaths in a group, than three Goliaths who are spread over the map. I guess the potential field COSa will be most effective, when the AI's team has the largest army. Then they can focus their fire on a few units, and it will be easier to surround the enemy.

Video (6,7,8) shows the trained AI playing as Terran with nine Vultures against the build-in AI, which is playing as Zerg and has nine Hydralisks. The three videos show three different tactics. In video (6) the AI engages with the enemy from the first second and wins with only one unit loss. The two other videos are earlier AI solutions, which seem to have some bugs, which I have fixed in video (6). The main bug was that the fitness function gave many points for having hit points left after end match, which is ok if the enemy units are killed. If not, I found out that it was better not to give them any extra points for hit points left, if the time had run out, meaning that the enemy was still alive. In video (7) the Vultures' tactic is to run

around in a circle in infinity, because then the Hydralisks will never get to them, because they are a lot slower than Hydralisks. In video (8) the Vultures tactic is to wait until the enemy is within a specific radius and then they attack. This tactic makes them win the match, but lose seven out of ten units (the last unit is a worker).

Video 11 is the big match between different types of units on both the Terran and the Zerg team. The Terran team has three Goliaths, two Siege Tanks and four Vultures against nine Hydralisks and two of the Zerg race's heaviest units with 400 hit points, namely the Utralisks. The trained AI has specifically trained potential field parameters for each of the three Terran unit types. The Terran team has a total of 995 hit points, while the Zerg team has a total of 1520 hit points. The AI with the specifically trained potential field parameters for all three Terran unit types wins with a loss of only four units out of ten, and both Siege Tanks survived, which are the most expensive once on the Terran team.

The fact that the trained EMAPF-based AI wins so convincingly every time, both against the build-in AI with one type of units, but also with several different types of units simultaneously and is able to win convincingly over an untrained AI supports the earlier conclusions that the AI can significantly improve its performances over time by training its potential field parameters using EA. I therefore conclude that using EA to find an optimal or near optimal combination of potential field parameters for micromanagement in SSC in RTS games, is beneficial.

Chapter 7

Conclusion

In this thesis I have demonstrated that it is possible to automate the tuning of multi-agent potential fields (MAPF) for small scale combat (SCC) scenarios in RTS games, or at least for the very popular RTS game StarCraft: Broodwar, using the BWAPI and the C# wrapper bwapi-clr-client to communicate with the game.

I have therefore applied an extra step to the MAPF-based methodology, that is used for designing more alternative AI solutions for RTS games, than the more conventional planning based solutions like e.g. the basic A* algorithm and different types of state machines. The extra step makes it easier to optimize the parameters of potential field charges by auto tuning using evolutionary algorithms (EA). This makes it possible to significantly improve the performance of an AI with untrained potential field parameters (also called EMAPF-based AI) in an unknown dynamic environment.

7.1 Competition

The EMAPF-based AI that I have constructed in this thesis, is able to outperform the build-in StarCraft AI in several different SCC scenarios. I have shown that the EMAPF-based AI both manage to outperform the build-in AI where both teams have only one type of units and when they have several different types of units. The EMAPF-based AI has been trained to use both Goliaths, Vultures and Siege Tanks and is able to control each separate unit type with each of their specific associated potential field parameters at once, and still outperform the build-in AI, which starts with a total hit point amount of 1520, while the EMAPF-based AI only has 995 hit points to start with.

I have also shown that the trained EMAPF-based AI manage to control other types of units, which the AI has not been trained with. The performance would probably have increased if the EMAPF-based AI had been trained with those units as well. Further I have shown that the trained EMAPF-based AI can outperform the best untrained EMAPF-based AI with random potential field parameters, when playing as Terran with the Goliaths units. When playing as Zerg, the trained AI performs better (even when it has not been trained with Zerg) against the untrained AI as Terran, than the untrained AI does while playing as Zerg. It seems, though, that the trained AI is so afraid of the edges and corners, that the enemy Goliaths can

hide there without being hit. This problem was not encountered when the trained AI played as Terran, and i am sure this effect would disappear, if the trained AI also had some trained potential field parameters for that particular Zerg unit type.

7.2 EMAPF

During the design of the EMAPF-based AI, 8 different potential fields was discovered, but only 7 of them were used in the tuning by the EA. The reason for this was that the EA would not have had much opportunity to tune parameters for the neutral units repulsion, simply because they are dynamic objects that randomly move around on the map and are not directly a part of the battle, but would be able to add more randomness to the match outcome. The EA with the current fitness function would not be able to analyse this field.

The 7 potential fields used in the tuning each have a force, a forceStep and a range parameter, i.e. there is 21 parameters that have been tuned in different EA configurations. The 7 potential fields are:

1. attracting the unit agents to each of their maximum shooting distance to the nearest enemy (MSDa).
2. attracting the unit agents to the centroid of their squad (COSa).
3. attracting the unit agents to the center of the map (COMa).
4. repelling the unit agents from the enemies, when the unit agents' weapons are in the weapon cool down state (WCDr).
5. repelling the unit agents from the enemies' maximum shooting distances (EUr).
6. repelling the unit agents from a predefined range from other unit agents, so that they do not collide and are hit by splash damage (OUr).
7. repelling the unit agents from map edges and corners, to avoid being trapped by an enemy force (MEr).

All these potential fields are enough for a trained EMAPF-based AI to control the unit agents to navigate around in simple environments, avoid colliding with static and dynamic units and hunt down the enemy units.

7.3 EA

To provide the most reliable and best results with EA, I recommend using at least two trials per chromosome, but ten trials per chromosome would be better. If time is taken into account, two trials is of course much faster, and can also give acceptable results. A population size of 40 seems to work fine with 25 generations. After the 25 generations it seems that there are very little, if any, improvements in the performance of the AI. The ten best performing solutions survive after each generation, while the rest are run through one or more variation operators. The

end result is almost the same when using both the mutation and 1-point crossover variation operators or when using mutation solely. So for simplicity its easiest to just use mutation solely.

7.4 Future Work

In further studies it would be interesting to see:

- The use of special trained potential field parameters for all the different unit types in an RTS game, StarCraft in particular.
- The use of the SCC trained potential field parameters on more difficult terrains.
- The use of the SCC trained potential fields in a Full RTS game scenario.
- How well these before mentioned improvements will work combined with the use of optimisation techniques for more macro-based decision making issues, like build orders/base building and more overall battle plans, i.e. which units and buildings to build, where and when to build them, when is the squad big enough to be able to beat the enemy units and last where and when to attack the enemy for being able to in the most efficient way to win against the enemy. *Genetic algorithm* could for instance be used to find optimal or near optimal build orders, like (Brandy, 2010) describes the EvolutionChamber made for for StarCraft II[1].
- How well this AI would handle a match against a skilled human player and against other types of AI implementations for StarCraft or other RTS games, without use of any cheating, like complete map information (no fog of war).
- The use of trained EMAPF-based AI's for playing StarCraft or other RTS games, where the focus of the optimisation should be to make the AI as human-like as possible.

[1] `http://code.google.com/p/evolutionchamber/wiki/Overview`

Bibliography

Adams, D. (2006). The state of the rts. http://pc.ign.com/articles/700/700747p1.html.

Arkin, R. (1987). Motor schema-based mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 264–271.

Barron, T. (2003). *Strategy Game Programming with DirectX 9*. Republic of Texas Pr.

Bellos, A. (2007). Rise of the e-sports superstars. http://news.bbc.co.uk/2/hi/programmes/click_online/6252524.stm.

Borenstein, J. and Koren, Y. (1989). Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179.

Borenstein, J. and Koren, Y. (1991). The vector field histogram: Fast obstacle avoidance for mobile robots. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 7(3).

Brandy, L. (2010). Using genetic algorithms to find starcraft 2 build orders. http://tinyurl.com/BuildOrdersSC2. [Online; accessed 17-December-2010].

bwapi-mono-bridge group (2010). bwapi-mono-bridge project page. http://code.google.com/p/bwapi-mono-bridge/. [Online; accessed 17-December-2010].

bwapi-proxy-java group (2011). Bwapi-proxy (java). http://code.google.com/p/bwapi-proxy/downloads/list. [Online; accessed 20-January-2011].

Eiben, A. and Smith, J. (2003). *Introduction to evolutionary computing*. Springer Verlag.

Geryk, B. (2010). A history of real-time strategy games. http://www.gamespot.com/gamespot/features/all/real_time/. [Online; accessed 17-December-2010].

Graft, K. (2009). Blizzard confirms one frontline release for 09. http://www.next-gen.biz/news/blizzard-confirms-one-frontline-release-09.

group, B. (2011). Bwapi project page. http://code.google.com/p/bwapi/. [Online; accessed 20-January-2011].

H., H. (2011). Skynet meets the swarm: how the berkeley overmind won the 2010 starcraft ai competition. `http://tinyurl.com/66zo3w7`. [Online; accessed 20-January-2011].

Hagelbäck, J. (2009a). A multi-agent potential field based approach for real-time strategy game bots. Master's thesis, Blekinge Institute of Technology.

Hagelbäck, J. (2009b). Using potential fields in a real-time strategy game scenario (tutorial). `http://aigamedev.com/open/tutorials/potential-fields/`.

Hagelbäck, J. and Johansson, S. (2008a). Dealing with fog of war in a real time strategy game environment.

Hagelbäck, J. and Johansson, S. (2008b). Demonstration of multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pages 1687–1688. International Foundation for Autonomous Agents and Multiagent Systems.

Hagelbäck, J. and Johansson, S. (2008c). The rise of potential fields in real time strategy bots.

Hagelbäck, J. and Johansson, S. (2008d). Using multi-agent potential fields in real-time strategy games.

Howard, A., Mataric, M., and Sukhatme, G. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. *Distributed autonomous robotic systems*, 5:299–308.

JB (1999). Starcraft named 1 seller in 1998. `http://pc.ign.com/articles/066/066492p1.html`.

Johansson, S. (2006). On using Multi-agent Systems in Playing Board Games.

Johansson, S. and Saffiotti, A. (2002). Using the electric field approach in the robocup domain. *RoboCup 2001: Robot Soccer World Cup V*, pages 117–138.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90.

Kraus, S. and Lehmann, D. (1995). Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171.

Laursen, R. and Nielsen, D. (2005). Investigating small scale combat situations in real time strategy computer games. Master's thesis.

Lesser, V. and Corkill, D. (2010). History and accomplishments of the multi-agent systems lab at the university of massachusetts amherst.

List-All (2008). Best selling pc games. `http://www.listal.com/list/bestselling-pc-games/`.

Liu, W., Zhou, L., Xing, W., and Yuan, B. (2010). Real-time hierarchical movement animation control for autonomous virtual humans. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 1.

Massari, M., Giardini, G., and Bernelli-Zazzera, F. (2004). Autonomous navigation system for planetary exploration rover based on artificial potential fields. In *Proceedings of Dynamics and Control of Systems and Structures in Space (DCSSS) 6th Conference*.

Nareyek, A. (2004). AI in computer games. *Queue*, 1(10):58–65.

Negnevitsky, M. (2005). *Artificial intelligence: A guide to intelligent systems*. Addison-Wesley Longman.

Pinter, M. (2001). Toward more realistic pathfinding. `http://www.gamasutra.com/view/feature/3096/toward_more_realistic_pathfinding.php`.

Release, B. P. (2010). Starcraft ii: Wings of liberty one-month sales break 3 million mark. `http://tinyurl.com/BlizzardPress`.

Release, E. P. (2009). E3 2009: Guinnes world records announces awards at e3. `http://uk.games.ign.com/articles/992/992211p1.html`.

Röfer, T., Brunn, R., Dahm, I., Hebbel, M., Hoffmann, J., Jüngel, M., Laue, T., Lötzsch, M., Nistico, W., and Spranger, M. (2004). GermanTeam 2004. *Team Report RoboCup*.

Thurau, C. and Bauckhage, C. (2004). Learning human-like Movement Behavior for Computer Games.

Thurau, C., Bauckhage, C., and Sagerer, G. (2004). Imitation learning at all levels of game-AI. In *Proceedings of the international conference on computer games, artificial intelligence, design and education*, pages 402–408.

Weber, B. (2010a). Starcraft (java) proxybot. `http://eis.ucsc.edu/StarProxyBot`. [Online; accessed 17-December-2010].

Weber, B. (2010b). Using a remote ai process. `http://eis.ucsc.edu/StarCraftRemote`. [Online; accessed 17-December-2010].

Weber, B., Mawhorter, P., Mateas, M., and Jhala, A. (2010). Reactive Planning Idioms for Multi-Scale Game AI.

Wikipedia (2010). List of best-selling pc video games — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/List_of_best-selling_PC_video_games`. [Online; accessed 17-December-2010].

Appendix: Extra experiments

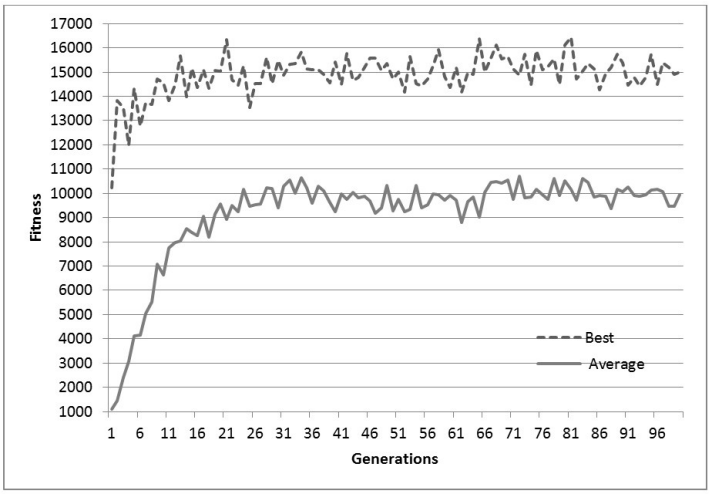


Figure 7.1: Performance graph for training of potential fields of Goliaths with one trial for 100 generations with 80 chromosomes, using mutation solely.

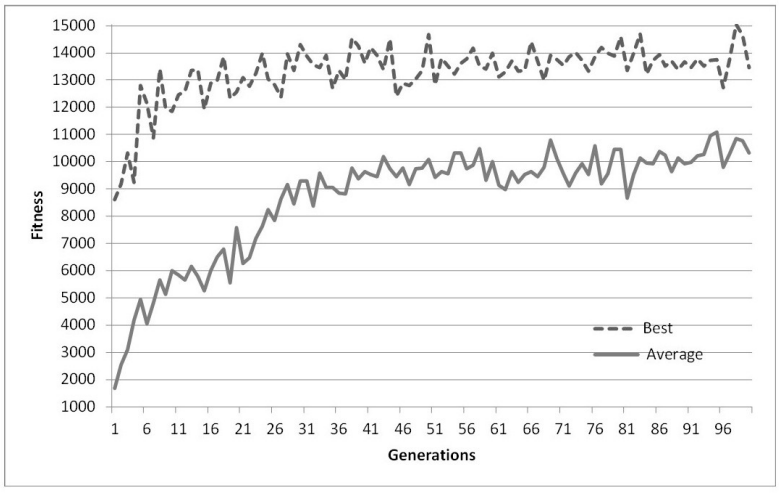


Figure 7.2: Normalized performance graph for training of potential fields of Goliaths with two trials for 100 generations with 40 chromosomes, using mutation solely.