6-16-2016

A Multi-Objective Approach to Tactical Maneuvering Within Real Time Strategy Games

Christopher D. Ball

A MULTI-OBJECTIVE APPROACH TO TACTICAL MANUVERING WITHIN REAL TIME STRATEGY GAMES

THESIS

Christopher D. Ball, Capt, USAF

AFIT-ENG-MS-16-J-004

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-16-J-004

A MULTI-OBJECTIVE APPROACH TO TACTICAL MANEUVERING WITHIN REAL TIME STRATEGY GAMES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Christopher D. Ball, B.S.C.E.

Capt, USAF

June 2016

AFIT-ENG-MS-16-J-004

A MULTI-OBJECTIVE APPROACH TO TACTICAL MANEUVERING WITHIN REAL TIME STRATEGY GAMES

THESIS

Christopher D. Ball, B.S.C.E.
Capt, USAF

AFIT-ENG-MS-16-J-004

Abstract

The real time strategy (RTS) environment is a strong platform for simulating complex tactical problems. The overall research goal is to develop artificial intelligence (AI) RTS planning agents for military critical decision making education. These agents should have the ability to perform at an expert level as well as to assess a players critical decision-making ability or skill-level. The nature of the time sensitivity within the RTS environment creates very complex situations. Each situation must be analyzed and orders must be given to each tactical unit before the scenario on the battlefield changes and makes the decisions no longer relevant. This particular research effort of RTS AI development focuses on constructing a unique approach for tactical unit positioning within an RTS environment. By utilizing multiobjective evolutionary algorithms (MOEAs) for finding an "optimal" positioning solution, an AI agent can quickly determine an effective unit positioning solution with a fast, rapid response.

The development of such an RTS AI agent goes through three distinctive phases. The first of which is mathematically describing the problem space of the tactical positioning of units within a combat scenario. Such a definition allows for the development of a generic MOEA search algorithm that is applicable to nearly every scenario. The next major phase requires the development and integration of this algorithm into the Air Force Institute of Technology RTS AI agent. Finally, the last phase involves experimenting with the positioning agent in order to determine the effectiveness and efficiency when placed against various other tactical options. Experimental results validate that controlling the position of the units within a tactical situation is an effective alternative for an RTS AI agent to win a battle.

Acknowledgements

I would like to thank Dr. Gary Lamont for guiding me his vast experience in the field of Artificial Intelligence, being of great assistance in this research effort. Gratitude also extends towards Jason Blackford and Donald Gruber for their prior efforts and inspiration for my project.

Christopher D. Ball

Table of Contents

Page

List of Figures

List of Tables

A MULTI-OBJECTIVE APPROACH TO

TACTICAL MANEUVERING WITHIN

REAL TIME STRATEGY GAMES

I. Introduction

This thesis documents efforts to improve the AFIT Real Time Strategy (RTS) Artificial Intelligence (AI) agent with a tactical positioning algorithm. In particular, this project utilizes Multi-Objective Evolutionary Algorithms (MOEAs) to determine an effective spacing position without relying on an exhaustive search method of the solution space [10]. RTS games provide a versatile platform for testing new AI techniques in a tactical environment as there is an infinite number of real-time scenarios that can be created for simulation [11].

1.1 Military Tactical Decision Making

When in a battle, effective maneuvering for each asset in the fight is critical. Each person, vehicle, weapon or tool needs to be used in the best possible manner to achieve the most desirable outcome: victory. On the job training isn't the best option for new leaders, even realistic training takes resources that may not be available at all times because of maintenance issues, lack of availability or risk of damage to the resource. Because of this, virtual training can be immensely useful with the ability for repetition of scenarios as often as needed. The AFIT RTS AI agent, that this research improves, seeks to assist with that training by providing a more realistic performing opponent to practice against.

1.2 Real Time Strategy Games

Real Time strategy games are an excellent domain to explore various AI techniques for both tactical and strategic decision making. Being a popular video game genre, it frequently employs many war time scenarios, both real world and fictional [11]. Such a training environment allows for an exploration of effective strategic plans and tactical maneuvers. The AFIT RTS AI agent, in particular, provides an opponent to practice these strategic and tactical techniques against that adapts to the trainee's skill level.

Tactical execution of units within a battle involves the usage of formations and the movement of such throughout the course of a battle is an important facet of winning an engagement. This particular project provides an expansion of the AFIT RTS AI agent that adapts to the user's skill and strategy by implementing a new tactical technique involving formation management.

1.3 Research Goal

The goal of this project is to develop and test an extension to Gruber's tactical AI portion of the AFIT RTS AI agent for the Balanced Annihilation mod for the Spring RTS Engine [6]. This extension, building on previous work by other AFIT students, performs formation based management of the units in a real-time environment to preserve the units on the field and lead the opposition into a disadvantageous situation. The goal of this research is *to take the invested resources of units and maximize their value by controlling their positioning.*

1.4 Research Objectives

Based on the goal, the research is designed to achieve the following objectives:

1. Design a mathematical problem representation and algorithmic solution to execute the tactical position process in an RTS environment. This allows various scenarios to be evaluated and potential algorithmic solutions to be weighed against one another.

2. Evaluate the offline performance of various MOEA algorithms within the positioning problem domain. A properly employed algorithm shall provide an effective answer based on the mathematical representation within the quickest time possible, allowing for an effective dynamic solution.

3. Evaluate the online performance of the full tactical positioning algorithm against other scripted agents found within the AFIT AI RTS agent. This demonstrates the effectiveness of the positioning algorithm by how many units are alive at the end of battle.

1.5 Research Approach

This research is building upon the previous work of Blackford [5] and Gruber [6], which showed that MOEA techniques can be adapted to both strategic (build order) and tactical (targeting) decision making. The implementation of their MOEA techniques improved the performance of the AFIT RTS AI agent in relation to other strategies and tactics, and demonstrated that they were viable options. In particular, Gruber's tactical agent optimized the targeting choices of the units under the agent's control. This research expands on the tactical realm, demonstrating techniques that can be used for unit control to optimize their positions in relation to the enemy's formation. This provides alternate tactical capabilities that can be chosen to best combat the enemy.

This research effort begins by defining the positioning problem, understanding the search space and building effective evaluation techniques. In this process, an offline simulation is developed to ensure that the algorithm is functioning properly. An offline simulation is useful as it can debug the algorithm design for various static scenarios without the stress of an evolving, complex real-time situation.

Once the positioning problem is defined and an algorithm is designed, it can be integrated into the AFIT RTS AI agent. An algorithm test is then performed to provide the desired functionality and adjustment for a real-time environment. This includes any and all adjustments required for transitioning from a static, offline simulation to a real-time, online simulation.

After being fully integrated into the AFIT RTS AI agent, the algorithm can then be tested with various MOEA alternatives. Simulations are ran with a wide variety of MOEA techniques to find the most effective for a given situation.

Once an MOEA is selected, the agent is then pitted against various other scripted agents as well as Gruber's previously developed targeting agent [6]. This testing is designed to determine the effectiveness of the positioning algorithm when placed into a real-time combat situation.

1.6 Thesis Organization

The remainder of this thesis is a discussion of the research, from the development through testing and analysis. Chapter II provides a background into Real Time Strategy games and developing agents within that environment as well as an overview of RTS Tactics and Multi Objective optimization. Chapter III covers the development of a positioning algorithm, the decisions that are made during the design process and the observed challenges found with integrating into the RTS environment. Chapter IV contains the design of experiments, giving an explicit description on how the experiments are performed. Chapter V provides an analysis of the results of the experiments. Chapter VI contains the overall conclusion of the project as well as observation for future improvements in this realm as well as other areas within the AFIT RTS AI agent.

II. Background

This chapter provides a base level of information about some of the major themes for this thesis research, such as decision making techniques, RTS games and platforms, and an overview of strategic options like build orders and tactical techniques. The chapter also provides an overview of previous AFIT research projects that investigate new improvements with RTS AI agents, as well as a discussion of current RTS AI research being conducted elsewhere. Finally, this chapter concludes with an explanation of this research and how it provides a progression of RTS AI tactical decision making techniques.

2.1 Decision Making

When creating a decision making agent, like the one used as the AFIT agent, it is important to understand the decision making process. While there are many methods to use that help make decisions, they all share the same facet of analyzing observed input data. An input is composed of relevant observed data, where the quantity of the data is just as important as a quality. Enough data should be gathered to make an informed decision without gathering so much data that the whole process is overwhelmed. Additionally, the data being gathered should be relevant to the decision being made as there is no value in gathering information about muzzle velocities of a weapon when purchasing a family sedan. This technique of finding the right amount of information required is called "thin slicing" [12]. Armed with the input information, it needs to be placed into a decision process. One method of decision making is called the OODA Loop, which is named after its four distinct phases: Observe, Orient, Decide, and Act [1].

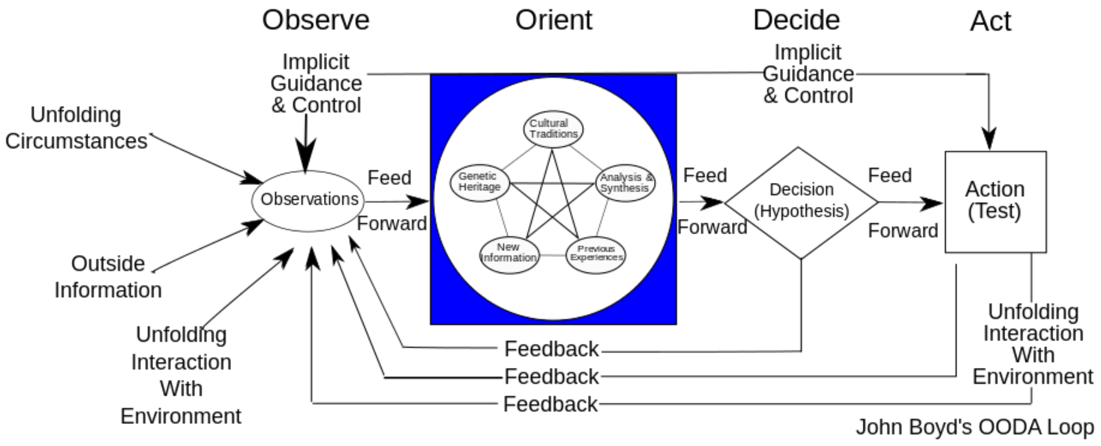


Figure 1. Boyd's OODA Loop [1]

OODA Loop.

The OODA loop is a simple decision making model that follows the steps of Observe, Orient, Decide, and Act. This process is intended to be highly universal in design and able to be applied to a wide range of situations. Figure 1 demonstrates this process.

- **Observe:** The observe phase of the OODA loop is the gathering of information about the current situation. In an RTS agent, this is the collection of the current state of the agent's units and the enemy's units such as positions, numbers and other related information. There is no processing done at this point.
- **Orient:** Orienting is analyzing the information gathered in the previous phase. An agent would begin the analysis of the gathered data. Strategic info would result in determining of the enemy's strategy. Tactically, the agent would analyze items such as attack power, hit points, and positions. The results of this phase are used in the next phase.
- **Decide:** The decision phase is making a decision based on the orientation of the observed data. An AI agent would be making decisions such as a strategic

build order to follow or tactical targets to fire at. Each member of a solution population represents a unique decision.

- **Act:** Acting is executing the decision made in the previous phase. In MOEAs, this means executing the proper balance of the various objectives found in the previous phases.

Incompleteness of OODA Loop Execution.

Sometimes the OODA loop can be forced into a premature restarting of the process[1]. This is because either the findings in the previous phases change the analysis of the situation or the situation has changed so rapidly that a reanalysis is required. For the former, where it is best to just choose an decision and proceed onward as the loop is likely unending. For the later, decisions need to either be made faster in order to keep up with the ever-changing scenario or the overall strategy for the scenario needs to be change to accommodate the situation.

Thin Slicing.

Thin slicing is a concept introduced in Gladwell's book, *Blink*[12]. Wherein, Gladwell describes the concept of people making snap decisions based off of small amounts of information. The idea is that the decisions made this way can be just as accurate as going into an in-depth analysis of all the data available. With more experience, a person tends to make more accurate decisions with thin slicing as that person can understand the problem space more intuitively than someone new to it. When applied to an AI agent, a proper learning algorithm can replicate a player's experience to make thin-slicing decisions much more effectively while providing an adequate solution for the problem. If implemented properly, this can cut down greatly on the agent's analysis time.

Current State Analysis.

A popular method of finding a solution to a problem is to look at every possible option at every decision point. While this decision process eventually yields an optimal solution, complex scenarios can take a large amount of time to find such a solution. This is undesirable because an real time problem space never stays still long enough to find a solution. When it performs a dynamic shift, the optimal solution also shifts along with it and would require restarting the calculation of options and repeating the whole process all over again, possibly getting the search stuck in a loop where it is required to restart before it ever finds an acceptable solution. In the RTS environment, this would mean that the opposition would have free reign over doing anything they want while the agent accomplishes no productive actions.

Instead of performing a thorough search to find an acceptable solution, a version of thin slicing should be implemented in order to reduce potential search time. This removes the concept of finding the perfect solution from the search all together, as the perfect solution is an unnecessary component of the search. In an RTS environment, one only needs to be better than the opponent. This does not require playing perfectly, as being better than your opponent is typically good enough to win the game.

When working within the AFIT agent, this means taking a snapshot of the scenario when the agent is called and making a decision based on the information provided right at that moment. The time for the calculation must be minimized in order to avoid the orders being outdated before they are even issued. Ultimately, an agent should be proactive in making efforts towards a victory condition.

2.2 Real Time Strategy (RTS) Games

Real Time Strategy, or RTS, are a genre of video games that focus on wargame simulation [11]. This often consists of the joint optimization of various goals, leading the player to gather the resources to build an army and lead that army to victory against the opposition. This level of balance between the optimizations can provide a large diversity in strategies, such as a quickly built strike force to surprise the enemy or a more durable start to overwhelm the enemy with a late-game superiority.

An RTS game typically starts with two or more players on different locations on a map. From there, they must build their base of operations to acquire new units. In order to facilitate the building of new units, the players can scout nearby their position through terrain hidden by the *fog-of-war*[13] to find resources that can be gathered and spent to construct more advanced buildings in their base and thus build more powerful units. At the same time, the players should utilize tactical options such as scouting the map for the enemy's location through fog of war. Once the enemy's position has been identified, the player can then proceed to win the game, typically through the destruction of the opponent's base.

Prior to victory, some strategic elements must be performed. Generally, a player must take steps to identify the strategy their opponent is attempting to execute. Adapting their build order to counter an enemy's strategy is needed to provide key supplies to defeat the enemy.

Each RTS game has their own unique traits, Some have different resource management techniques, while others focus on novel unit ideas. A brief history of the RTS genre is presented to highlight important advancements and how the games evolved to capture a more effective simulation feel[11].

Dune II.

Dune II: The Building of a Dynasty, released in 1992, stands as a marker of the first "Traditional RTS" that combined the concepts of its predecessors into a solid package. Themed after Frank Herbert's *Dune* science fiction novel series, it was the

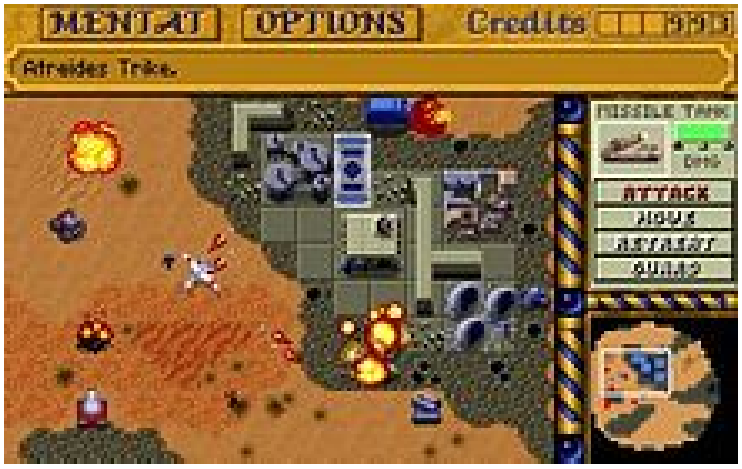


Figure 2. A screenshot of Dune II [2]

first game that provided the resource gathering concept for building new units as well as the base construction philosophy. *Dune II* also provides the concept of the technology tree, a series of dependencies the player is required to satisfy to access more powerful units and buildings, with a trade-off of requiring more investment to reach those items.

Additionally, *Dune II* also has one of the first AI agents utilized in a real time strategy environment. While there existed flaws in AI execution, it can be recognized as a starting point for future RTS AI agents [2].

WarCraft and StarCraft.

WarCraft and *StarCraft*, two historic gems of the RTS genre by Blizzard Entertainment, are strong areas of research for those seeking to learn more about RTS. *WarCraft* was the first in the series, released in 1994, and provided a high-fantasy conflict between Humans and Orcs. Sequels included additional forces to play as, but the core of the game held true to the RTS tenants. The most unique resource introduced by the game is the *supply* concept, where a player has to build an infrastructure to support their armies. In *WarCraft* that is represented by farms, and the player can not build units beyond what their farms can support, adding a ceiling to the army size [14].

StarCraft originally was intended to be "*WarCraft* in space", but evolved into a life of its own with the high amount of popularity it received [15]. *StarCraft* balances three factions against one another: the Terrans, which are humanity in space, the Protoss, which are an ancient, powerful alien race that's fewer in numbers, and the Zerg, which are a biological swarm led by a hivemind. While retaining the resource and supply concepts from *WarCraft*, *StarCraft* separated itself with the uniqueness of the armies and their functions. Terrans follow most traditional human war concepts, but had many base structures that could be moved after construction. The Protoss are constrained in having to build their buildings within a certain radius of their Pylons, their supply buildings but also had much more durable units. The Zerg being biological in nature, had units that would sacrifice themselves to construct new buildings, as well as each building providing a biological foundation on the ground nearby to construct additional buildings. All units are also constructed from hatcheries, with larve allowed to be developed into any unit unlocked in the technology tree. *StarCraft* is still a pillar of RTS even today, with its sequel *StarCraft II*, frequently seen at professional level video game tournaments [16].

Total Annihilation.

Total Annihilation provides a large-scale variation of the RTS concept. What makes it unique is that instead of gathering resources with workers, (Spice in *Dune II*[2], Crystals and Gas in StarCraft[15]) a captured resource location provides a steady stream of that particular resource into the player's stockpile. Construction of units require a certain number of resources that are taken at a steady rate from the resource storage, which counterbalances the steady income of resources. With the

Figure 3. A screenshot of Total Annihilation [3]

vast number of units being built, players build large scale armies rather quickly and pit them against each other in a large, strategic fashion in order to capture resources and eventually defeat their opponent [3].

Company of Heroes.

Company of Heroes is a game that handles unit and resource management in a unique way. This World War II themed RTS divides the map into regions that can be controlled. Controlling a region provides resources and a common win objective by gaining victory points from controlled regions. Unit management within the game of *Company of Heroes* emphasizes squads. Individual infantry units are not built. Instead, they are handled as squads. What this emphasizes is the control of groups of units as opposed to the individuals. The reason why *Company of Heroes* is an interesting game is that it provides new possibilities for AI agents such as units being managed as squads as well as contesting key locations on the map as opposed to a simple seek and destroy concept [17].

Spring RTS Engine.

The *Spring* RTS Engine provides a strong open source RTS environment for research. There are many options that *Spring* can provide, the most useful of which are visualization, unit customization and the open-source interface [18].

The visualization component allows for a 3-dimensional representation of the simulation as it is happening. Paired with the customization, many historical or real-world scenarios can be created from the units involved to the terrain featured. By representing this visually, animations can be created that demonstrate the battle as it happens, as opposed to numeric representations. This allows for easier visual feedback of the scenario as it is happening for both the user and the AI agent developer.

Unit customization is a very powerful component provided by the *Spring* engine. This allows for an open customization of the units that are utilized in a particular game within the *Spring* engine, providing a wide amount of flexibility and allowing for representation of real-world units within the environment.

Spring is an open source engine. This allows for application of modifications and AI agents with relative ease compared to other RTS environments. The open source concept allows for a direct communication between AI agents and the *Spring* environment.

Balanced Annihilation is a modification to the Spring engine that replicates the game play of Total Annihilation style games [19]. This particular mod is used within the Spring engine for this research effort as it is a fully designed game with an AI option. Other games within the Spring Engine community tend to avoid AI implementation in favor of focusing on Player vs Player combat[**?**].

Tactical Airpower Visualizaton.

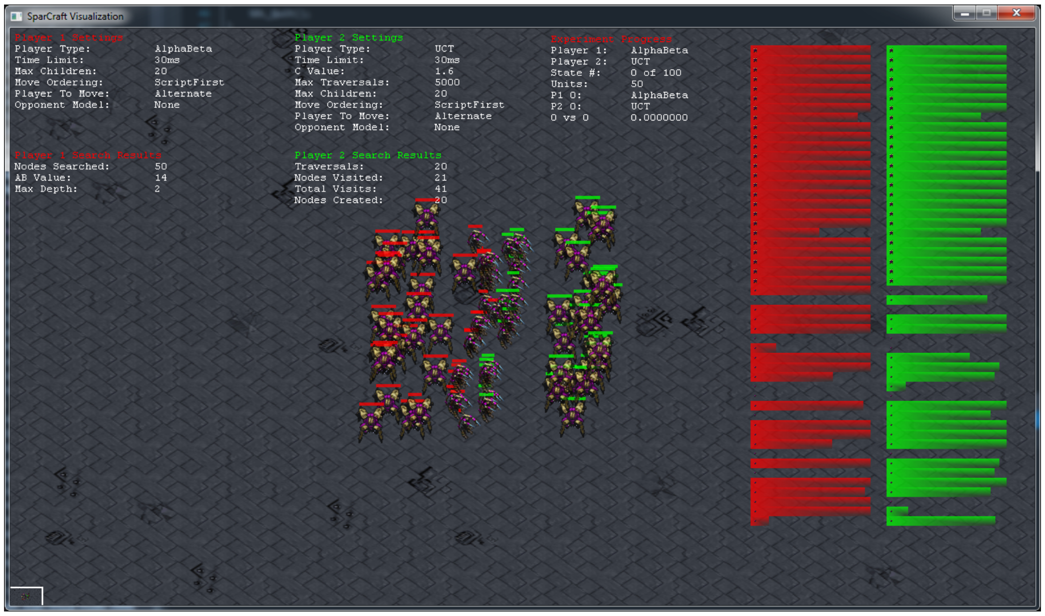
Tactical Airpower Visualization (TAV) is one of the recent iterations of the Air Force's approach to model an air campaign through an RTS environment [20]. Used in many officer training courses available at Maxwell Air Force Base, the game provides a top-level perspective to the coordination of an air campaign. Each training course provides a different scenario that simulates a conflict somewhere on the globe with a vast majority of potential situations that may occur in a real-world situation.

TAV is unique because it is a team-based affair. Each player on a team controls a different group of units with a different goal in mind. For example, one player could control Air-to-Air Superiority while the another player may be focused on Air-to-Ground operations and a third could be supplies and transportation. All these players are lead by a central command, one or two players who do not have direct control over any units but will provide direction to their forces as a whole. Teams are scored with a point total based off of their performance in the scenario.

A problem with this system is that the scenarios are heavily scripted. Enemies take actions at predictable intervals or doesn't perform any aggressive actions until the player does so first. This means that once the team identifies the weaknesses of the scripts, they can be exploited with no resistance from the opposition. This inspires a desire in this project to create a more "human" agent that performs both effectively and somewhat unpredictably in order to provide a more effective training simulation[20].

2.3 RTS Development Platforms

RTS games provide a great player interface into a real time simulation environment, but AI agent development within the game isn't always an easy affair, with many games not intending to have an AI agent being developed for it. In order to



```
Player 1 Settings                      Player 2 Settings                     Experiment Progress
Player Type:        AlphaBeta          Player Type:        UCT               Player 1:   AlphaBeta
Time Limit:         30ms               Time Limit:         30ms              Player 2:   UCT
Max Children:       20                 C Value:            1.6               State #:    0 of 100
Move Ordering:      ScriptFirst        Max Traversals:     5000              Units:      50
Player To Move:     Alternate          Max Children:       20                P1 O:       AlphaBeta
Opponent Model:     None               Move Ordering:      ScriptFirst       P2 O:       UCT
                                       Player To Move:     Alternate         0 vs 0      0.0000000
                                       Opponent Model:     None

Player 1 Search Results                Player 2 Search Results
Nodes Searched:     50                 Traversals:         20
AB Value:           14                 Nodes Visited:      21
Max Depth:          2                  Total Visits:       41
                                       Nodes Created:      20
```

Figure 4. SparCraft simulation of two squads of Protoss Dragoons fighting against one another [4]

make the development process much smoother, simulation softwares for various RTS games have been developed.

Wargus.

Wargus is an open-source recreation of *WarCraft II* for a full suite of environment manipulation, many of which are unavailable in the actual *WarCraft II* engine. *Wargus* behaves as an extention to the *WarCraft II* environment, and requires many features from a *WarCraft II* installation [21].

SparCraft.

SparCraft is an open source environment to simulate the *StarCraft* game engine, typically used in performance analysis of AI agents built for *StarCraft*. *SparCraft* can replicate damage, armor, hitpoints and research, but cannot account for collisions and area effect damage[22].

2.4 Strategic Decision Making

The concept of a strategy can be distilled down the high level objectives that need to be met in order to reach a goal. Traditionally, the decision making process for a strategy is left with the leadership of the group that requires the decision. The decisions that are made by the leadership affect large numbers of resources such as people and materials. In warfare, strategy is used to accomplish the goals of the military leader or a country. There have been many military leaders throughout history who have varying viewpoints on effective strategies. For example, Clausewitz describes how a strategy can either be a quick, decisive attack or a series of long, drawn out battles of attrition with the ultimate goal of reducing the enemy's effectiveness [23]. The United States Army Field Manual of Military Operations provides a series of 9 essential concepts [24]:

1. **Define an objective** Direct every operation towards a clearly defined and attainable goal

2. **Seize the initiative** Don't wait for the enemy to act first. Become proactive, not reactive

3. **Mass of Force** Concentrate combat power at the appropriate place and time for a decisive victory

4. **Economy of Force** Allocate what resources the operation requires to be successful. Too few and the mission could fail, while too many risks items unnecessarily that could be used elsewhere.

5. **Outmaneuver the enemy** Placing the enemy in a poor relative position forces the opponent to only have poor decisions to choose from

6. **Unity of Command** Ensure that everyone is united under one commander in order to maintain focus on the objective

7. **Security** Prevent the enemy from attaining an advantage

8. **Surprise** Hit the enemy where they are least prepared for maximum effectiveness

9. **Simplicity** Keep the plan simple and clear to ensure thorough understanding for all parties involved

A properly designed military strategy contains the objectives of preventing the enemy from fighting through the destruction of their forces or the removal of their resources through the capturing of their territory.

Strategy in RTS Games.

Strategic planning in RTS games typically requires the emphasis on the efficiency of the constructive actions, known as a build-order [5]. The build-order is responsible for advancing a player through technological improvements that allow the player access to more effective combat units, buildings or research. The typical RTS game starts the player off at the lowest technological level, where the player produces infantry and resource gathering units. As the player invests resources into new technologies, usually through the construction of buildings, the player gains access to better equipped infantry and light vehicles and eventually powerful items such as tanks and artillery. When developing a good strategic plan, having a knowledge of other players meta-level strategies is needed, as being able to adapt to the opponent's strategy is key to gaining the upper hand in a battle.

Figure 5 describes the possible methods to develop a strategy for an AI agent in an RTS game. The first major decision is to distinguish if the agent is to be

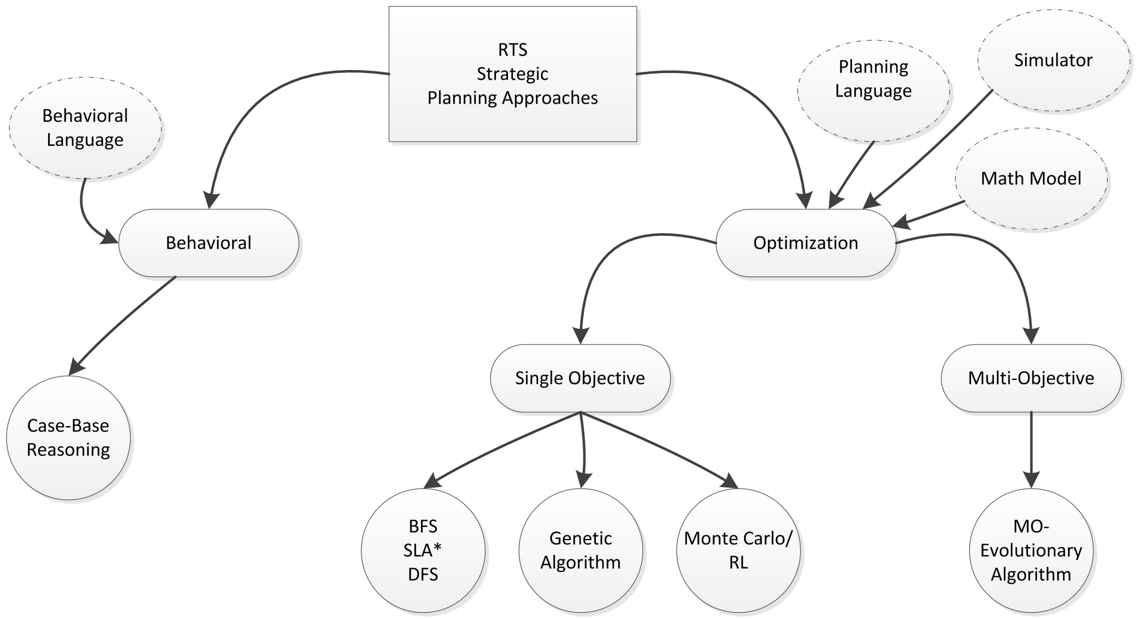


Figure 5. RTS Strategic Planning Tree [5]

behavioral or optimized. Behavioral focuses identifying the opposition's strategy and countering it. A behavioral agent gathers data on the opponent and attempts to match it with a known strategy and then implement a counter-strategy based on the identification made. The optimization branch of the tree ignores what the opponent is doing strategically. Instead, the agent focuses on optimizing a certain aspect of a build order[5].

2.5 Tactical Decision Making

Tactics are the detailed steps required in executing a strategy. Unlike defining a strategy, the details of what encompasses the tactics vary greatly between situations. They can take into account the current status of resources such as personnel health, ammunition stores, and enemy positions. They also utilize the knowledge of the environment, such as terrain features like hills and choke points.

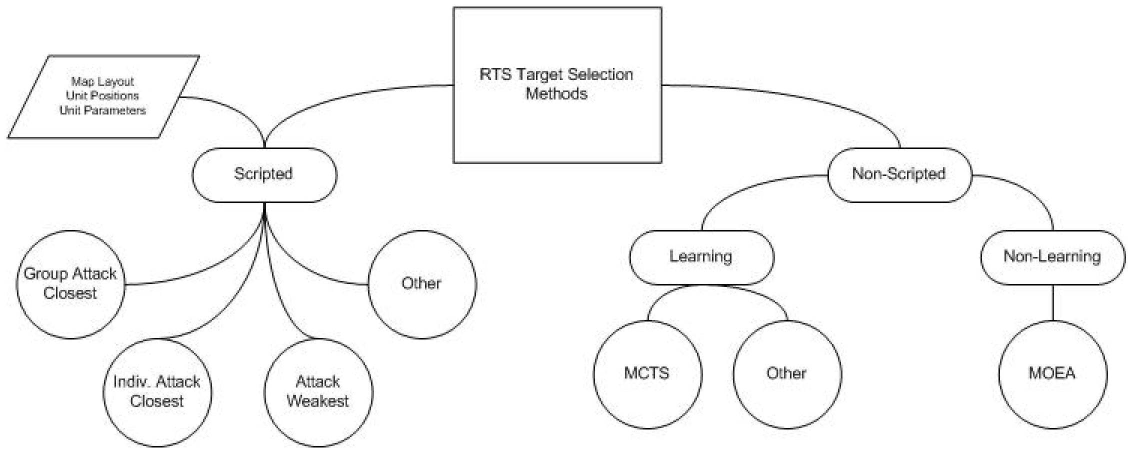


Figure 6. RTS Tactical Planning Tree for Targeting [6]

Tactics in RTS Games.

Within an RTS game, tactics involves the low-level management of individual units or groups of units. When playing an RTS game competitively, tactical micromanagement is an important key to success. It includes the movement decisions of where to send units, what targets that should be attacked and how to maneuver within a combat situation. Important decisions such as keeping all the units as a large force or splitting all units into smaller groups, possibly down to individuals, make all of the difference in the goals that are being attempted. When a fight breaks out, what enemies to prioritize to target is also important as taking out a high-damaging unit or group of units can severely cripple the enemy's ability to fight.

Figure 6 shows the decision tree for making tactical decisions related to target selection. On the left, there are various, meta-level scripted methods that encompass simple approaches. Tactics such as the entire force attacking the closest enemy, or each individual unit attacking their closest enemy. There is also the ability to attack the weakest unit or a unit that has been identified before the game by the script writer as an important target. These approaches are simplified, however, and can be exploited given knowledge of the script being executed.

On the other side of the decision tree, there exists various tactical options to compensate for the simplicity of scripted methods. Learning methods can be implemented through the usage techniques such as the Monte Carlo Tree Search (MCTS) [25] to analyze the state of the battle and make an appropriate decision. Alternatively, a set of expert data could be provided to an agent such that a similar situation could be identified and an effective tactical approach could be used for that situation. As the last remaining option on the tree, MOEAs provide the ability to create a tactical decision based off of the currently available information to attempt to outperform a human player [6].

Maneuvering Tactics in Combat Scenarios.

This project focuses on movement based tactics. Army Field Manual 3-21.8 (FM 7-8) has a good description of infantry based tactics, where it describes the procedure for an infantry squad to move through hostile territory [26]. This research effort in particular focuses on tank platoon tactics, which the Army covers within FM 3-20.15 [27]. This document in particular covers many important things related to Tank combat, such as Offensive, Defensive, Patrol, Escort and Reconnaissance. When discussing formations for maneuver, FM 3-20 specifically states "Formations are not intended to be rigid, with vehicles remaining a specific distance apart at every moment." This is important because too many assets clustered together can be hit by the same enemy munitions (air strikes, mortars, mines, and other explosive devices) and block each other when incapacitated. This also allows for the various units within the group to attack weaker portions of the enemy formation as they adjust rapidly to the changing situation of combat.

Taking this knowledge to use, countering a strategy employing tanks with area-of-effect weapons, as seen in Blackford's work [5], would require a spreading of units on the agent's side. Additionally, the spread needs to keep as much firepower on the enemy as possible, meaning the agent's forces cannot be spread out too much.

2.6 Developing a RTS AI Agent

When creating an AI to tackle an environment, it is easy to overlook all the things that humans take for granted. The challenge comes from implementing these in a very clear form that is either mathmatical or algorithmic. The resulting implementation must be both accurate in its calculation as well as being fast enough to be calcuated within a reasonable amount of time. When dealing in RTS games, there are 6 defined challenges that agents must overcome [28]:

- **Resource Management:** This challenge deals with balancing the acquisition of resources in an RTS game with the investment of those resources into buildings and units to further the agent's strategy. Players typically refer to this as a *build order*
- **Decision Making Under Uncertainty:** Unknown information makes it rather challenging to develop a perfect strategy. *Fog of War* and other methods of concealment found in RTS games mean that there are only partial pieces of information that will be available. An effective agent must be able to devise an effective strategy based on this knowledge.
- **Spatial and Temoporal Reasoning:** Navigating a single unit amongst a terrain is quite a formidable task, let alone navigating an entire army. When enemy units are added to the mix, then the ability to navigate effectively becomes incredibly difficult. Additionally, with the environment being real-time, the situation changes as time progresses. A solution that is optimal at one time period can be completely irrelevant during the next.

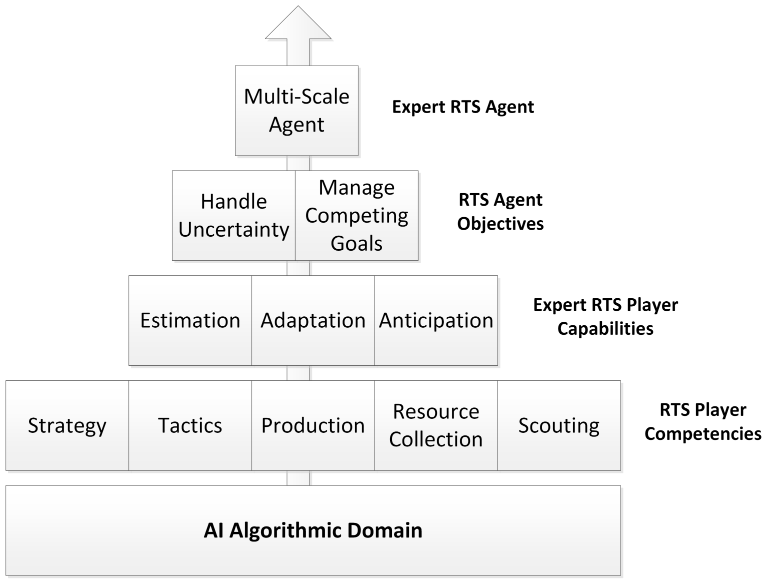


Figure 7. RTS Agent Pyramid [5]

- **Collaberation Between Agents:** Sometimes teamwork is required for an agent as a whole. Various sub-agents that are group together as a whole require communication between the various sub components to balance out their function. Additionally, should two agents be placed in the same alliance in an RTS scenario, communication to ensure their goals do not conflict is important.

- **Opponent Modeling and Learning:** When engaged with an enemy, understanding what that opponent is doing is essential for devising an optimal strategy. An effective opponent modeling strategy allows for an agent to recognize what the opponent is trying to accomplish with their strategy. After an agent is able to identify the strategy, choosing the proper course of action to best counter it is a vital step to determining the optimal path. Typically this involves remembering the outcomes of past encounters with the identified strategy and chooses a sequence that performed well.

RTS Agent Subfunction Breakdown.

Figure 7 demonstrates how an agent can be broken down into sub components that help cover each of the areas described earlier in this section. Building from the bottom up, each block refers to a skillset that an agent requires in order to successfully build upon others. The lower skillsets are more fundamental in nature, such as gathering resources or executing a build order. Each subsequent level is a more narrow and complex concept that is more difficult to execute effectively within an RTS agent.

2.7 Previous AFIT Agent Developments

The agent being modified for this research topic has been in development for five years, with Jason Blackford's work in 2013 and Donald Gruber's work in 2015 as the most recent improvements. The objective for each step of the development of the agent is to build on and improve a customizable RTS AI agent that can be used as a means to train military members in strategic and tactical decision making.

Adaptive Response - Weissgerber's Agent.

Weissgerber's agent, developed in 2010, is capable of reacting to the current situation in an RTS game by analyzing and acting on a subset of "features" which are cpaable of encompassing the current state of the game. The result was an agent that was able to outperform scripted agents by analyzing their previous performance and developing an active counter strategy. The resulting counter strategy was 100% effective against the scripted agent it was developed to beat by chosing decision paths that lead to a winning condition through optimization of the analyzed features [29].

Strategy Optimization - Di Trapani's Agent.

Di Trapani's agent identifies an incoming wave of enemies and develops a strategy to counter the situation [9]. This work is a continuation of Weissgerber[29], where Di Trapani determines the advantages and disadvantages of 8 different strategies tested against each other strategy. These strategies were identified as Infantry rush, Blitz, Bomber, Expansion, Tank Rush, Defend Artillery, Anti-Air and Turtle. The testing of each item was performed with a goal to identify counter strategies for each of the tested methods. With the results of counter strategies, Di Trapani utilizes various classifiers to predict the enemy's strategy and choose a counter strategy to employ [9].

Build Order Optimization - Blackford's Agent.

Blackford's project continues the work from Di Trapani by creating a method to optimize the strategic decision making done in the early stages of a game. In particular, the agent focuses on Build Order Optimization (”BOO” for short) through a Multi Objective Evolutionary Algorithm (MOEA) to optimize the build order for a given faction in the game. The MOEA selected balanced three objective functions. First, The agent focuses on minimizing the steps to transition from a current strategy to a new strategy. Secondly, it represents the consumable resources required to transition between the strategies. Thirdly, the time required to switch strategies, or makespan, is weighed for each option. The result is that this MOEA is capable of out-manufacturing each other AI agent that it was tested against.

Tactics Optimization - Gruber's Agent.

Gruber's agent, the most recent addition, aims to improve the combat effectiveness the units that were built by Blackford's agent improvements. By utilizing an

MOEA technique, Gruber optimized the tactical decisions of units when targeting units controlled by the opposing force. A pitched battle was established setting two teams of twenty-five “stumpy” tanks against one another. Stumpy tanks are unique because they fire an area-of-effect ordinance. Gruber’s addition to the agent demonstrated to be highly effective against other scripted options such as “attack closest” and “attack weakest” [6].

Unit Management - AFIT RTS AI Agent Continuing Work.

A plethora of work has been completed for the AFIT RTS AI agent, having the software being passed off between multiple developers. This does leave the question of “What is there left to accomplish?” It can be said that there is much work left to be done. As an example, building placement could be improved. The agent now, while building in an efficient order, can put buildings in some unusual locations. An additional example, the agent at times constructs a building that gathers resources on a resource node, but does not take into account with a short time investment a more valuable resource node could be utilized. Another weakness of the current agent is the lack of scouting capability. It is heavily reliant on a crutch of perfect vision in order to operate. Finally, the agent has a simplified tactical approach to moving combat units around the map. A lead unit is chosen and all other units follow in a “big chaotic ball”. There is a possibility of improving the movement on more of an individual level, giving the option for formations or positional control of units mid battle. Simply put, there are many “fine tuning” approaches to improving this agent that are available.

2.8 Current Research in RTS AI Tactics Optimization

Tactical AI development isn't exclusive to just AFIT. Others have conducted similar research into tactical unit management, making it prudent to take note of the work that has been done before. Each of the following research efforts provide their own unique contributions to tactical management in the way of target selection, formation management and battle decisions.

David Churchill's Research.

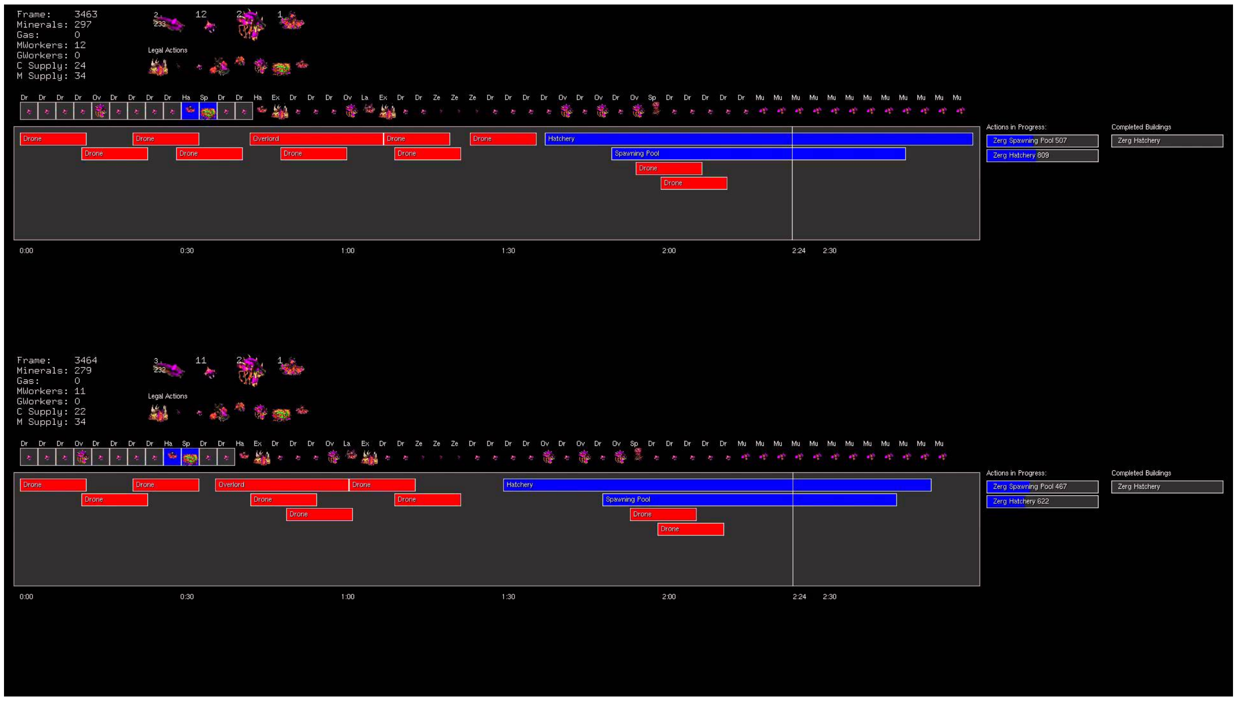
David Churchill, a strong figure in the realm of RTS AI research, is frequently developing new techniques for RTS AI. He keeps a public database of his work in his University of Alberta webpage [30]. This AI is the result of Chruchill's PhD dissertation, where this RTS AI agent has competed within several StarCraft AI competitions[31]. The following subsections highlight his work in designing the University of Alberta Bot.

Portfolio Greedy Search.

Churchill's Portfolio Greedy Search performs a variant of a hill-climbing search by analyzing potential future moves to make a decision associated with targeting enemy units. In his 2013 paper *Portfolio Greedy Search and Simulation for Large-Scale Combat in StarCraft*, Churchill describes an algorithm that out-performs both Alpha-Beta and UCT searches [32].

Build Order Optimization.

Churchill's UAlbertaBot, an AI that plays StarCraft, implements a build order optimization technique is that completely heuristic based. Utilizing an in-house developed software called Build Order Search System, or BOSS, it determines the most


```
Frame:    3463
Minerals: 297
Gas:      0
MWorkers: 12
GWorkers: 0
C Supply: 24
M Supply: 34
```

```
Frame:    3464
Minerals: 279
Gas:      0
MWorkers: 11
GWorkers: 0
C Supply: 22
M Supply: 34
```

Figure 8. StarCraft Build Order Search System visual example [7]

efficient path of building based on the desired resources being acquired. It determines the best times when to build what items, solving a very complex scheduling problem in real time [33].

Formation Management.

One aspect of having individual unit control is allowing them to maintain formations. Advantages to formations can include safe spacing of units or keeping more valuable units in safe locations or positions that maximize their output. A 2008 paper entitled *Dynamic Formations in Real-Time Strategy Games* analyzes employing such formations. The paper describes an employment of algorithms that create a dynamic formation in relation to enemy forces within a game. This dynamic formation manager performed well against a variety of existing agents, demonstrating the potential for improvement through formation management of units in an RTS game [34].

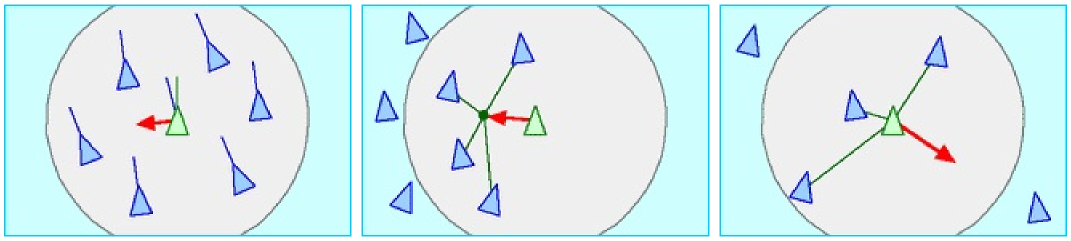


Figure 9. A visual representation of the three Boids concepts. From left to right: Alignment, Cohesion and Separation [8]

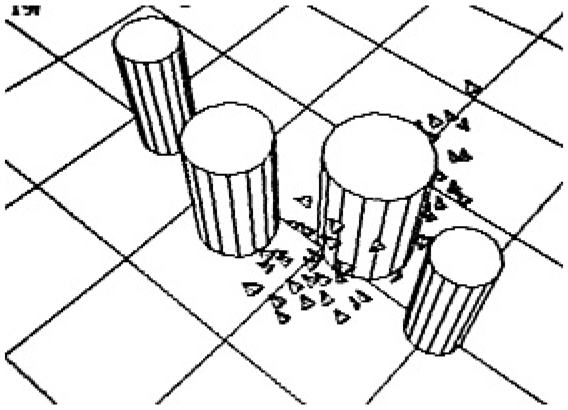


Figure 10. Demonstration of a flock of boids navigating around an obstacle [8]

Boids.

A concept by Craig Reynolds intends to algorithmically represent the flight and function of a flock of birds as they are flying together [8]. They use the concepts of Separation to avoid crowding, Alignment to keep heading with the heard and Cohesion to keep towards the average position of local flockmates to maintain a formation with one another while traveling. Each individual boid makes a decision on it's next movement based on the distance to other boids in the flock as well as the angle the other boids in the local neighborhood have in relation to itself (Figure 9).

By applying these concepts, Reynolds was able to demonstrate a flock of boids able to navigate about terrain to a destination. This means that each individual boid is capable of viewing the path ahead and plotting a route that avoids the obstacle while maintaining coheesion with the flock. Sometimes, the flock splits in two to

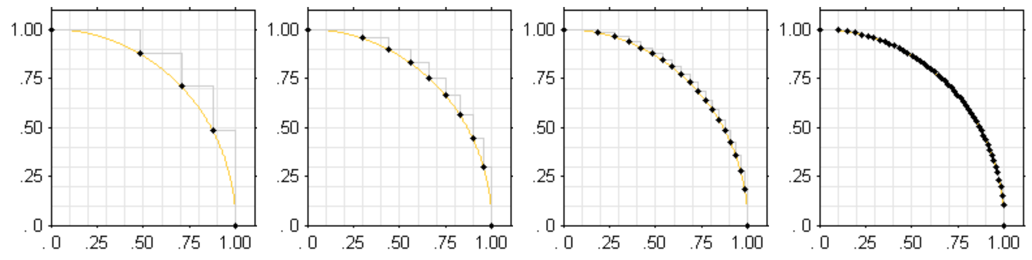


Figure 11. Example Pareto Front with Different Population Sizes (5, 10, 20, 50)

navigate around an obstacle (Figure 10), after the obstacle has been overcome, the flock then merges back together.

This is a similar problem to the research question posed in this paper. The concepts presented in Boids are benificial to generating an algorithm for the RTS Positioning Problem.

RTS AI Genetic Algorithm Implementations.

Opening strategic plans are a popular method for optimization, as each action should be executed as efficiently as possible in order to maintain optimality in a strategy. Gmeiner, Donnert and Kostler, from the University of Erlangen-Nuremberg, developed a multiple objective genetic algorithm that accomplishes optimality of a strategy in the game *StarCraft II* through the usage of the NSGA-II algorithm [35].

2.9 Multi-Objective Evolutionaly Algorithms

Multi-Objective Evolutionary Algorithms (MOEAs) are methods of solving problems by analyzing the potential results with regard to a variety of objective metrics [36]. Where a single objective focuses on maximizing a single facet of strategic importance, through either an equation or metric, an MOEA weighs multiple such facets in order to make a decision, thus providing a wider range of outcomes. Each of these weighted functions being evaluated together forms what is known as a Pareto front. The Pareto front consists of a set of optimal solutions that maximize the solution based on the ranking of which functions are more important. Since each solution cannot be completely maximized in a multi-objective problem, there are tradeoffs between attempting to maximize each objective function. Every axis in the Pareto front is an individual function, with each point within the space as an optimal relationship between those axes. This front provides the user with a visual method of showing how the functional weighting affects the overall outcome of the algorithm [10]. An example of a Pareto front can be seen in Figure 11.

MOEA Software Packages.

MOEAs are complex algorithms. Implementing such algorithms from scratch can be quite a daunting task and ensuring that they are functioning properly can be an even more difficult proposition. Employing an already existing software solution takes the difficulty of implementing an MOEA away. There are several MOEA software libraries existing that can accomplish this, some of the more capable libraries are:

PaGMO/PyGMO.

Parallel Global Multiobjective Optimizer, or PaGMO, is a C++ based algorithm platform developed by the European Space Agency that emphasizes parallel processing of common MOEA softwares through the utilization of an island model. PyGMO is a variant of PaGMO that implements a Python interface into the PaGMO system[37].

MOEA Framework.

MOEA Framework is a Java based open source library for a wide variety of MOEAs. It provides a easy to implement algorithms as well as a suite of various analytical tools built into the software [38].

ParadisEO.

ParadisEO is a C++ based MOEA software package that builds its design around modules for different focuses in MOEA structures [39]. The EO variant focuses on Evolving Objects. It allows for various "components" to be utilized based on the problem requirements [40].

jMetal.

jMetal, or Metaheuristic Algorithms in Java, is an object-oriented, Java-based framework for MOEAs. It contains many MOEAs along with several useful metrics for analysis[41].

Description of MOEA Algorithms.

There are a wide variety of MOEA algorithms that exist. Choosing the proper MOEA for a problem matters because various factors such as calculation speed, population size and rate of convergence are all facets that impact the results. The MOEA algorithms chosen for evaluation are NSGA-II, SPEA2 and NSPSO, each of which is described in the following subsections. These algorithms were chosen because they are very prominent and popular MOEA options that are executed effectively across various software libraries.

NSGA-II.

The Nondominated Sorting Genetic Algorithm II, or NSGA-II, is an MOEA developed by Kalyanmoy Deb[42]. This algorithm was developed to be an improvement to the original NSGA algorithm. NSGA an NSGA II both perform their evolution by first performing a nondominated sorting. This sorting identifies the first set of nondominated solutions (the closest ones to the true Pareto front) moving them into a list and repeating the search for the remaining points. The resulting algorithm for NSGA runs in $O(MN^3)$ time. NSGA II, on the other hand, utilizes a faster sorting method that keeps track of what other population members each solution dominates. Each solution then obtains a count for the number of solutions dominating it and when the nondominated solutions are removed, each population member that is previously dominated has their count reduced for each nondominating solution that was separated. This can be accomplished in $O(MN^2)$ time [42].

Both algorithms then perform a function to preserve diversity of solutions. In NSGA II, that is the calculation of the density for each member of the population and using that as part of the ranking factor. Each solution is then sorted by their nondomination rank and then by their crowding distance, preferring those solutions that are in less crowded regions.

NSGA II then performs binary tournament selection, followed by mutation and recombination to create an offspring population. This population is then merged with the original population, which keeps the top number of solutions equal to the original size of the population.

SPEA2.

The Strength Pareto Evolutionary Algorithm (SPEA) 2 is an algorithm designed by Zitzler and Thiele to improve upon the original SPEA [43]. Both SPEA and SPEA2 utilize a population and an external set of solutions called an archive. For each iteration, the nondominated members of the population are copied into the archive. Then, the archive removes all members within that are now dominated or duplicated. If the archive exceeds its size, the algorithm keeps the best evaluating diverse members within.

Fitness values are assigned to each population and archive member. Each member in both sets are assigned a strength value that represents the number of solutions that member dominates. Then each member receives a raw fitness that is the sum of all the strength values of the solutions that dominate that member. When all of the values become nondominated, it becomes difficult to differentiate however as their raw value becomes 0. To compensate, each member is given a density value based on the distance to the nearby population members. This value is then added to the previous raw fitness value to achieve a unique fitness.

After performing binary tournament selection, crossover, and mutation, the algorithm repeats itself by sorting and re-evaluating the population and archive.

NSPSO.

Nondominated Sorting Particle Swarm Optimizer, or NSPSO, is a multi-objective particle swarm algorithm by Ziaodong Li that utilizes the nondominated sorting concept from NSGA II [44].

Partcle swarm optimization (PSO) is a single objective algorithm inspired by behaviors of insects and animals that swarm together [45]. In PSO, a population travels in a swarm-like manner towards a promising area of the search space. NSPSO is a variant of PSO that aims to perform the same effect on multi-objective problems.

The Non-dominated sorting portion of NSPSO adopts the same method utilized by NSGA II [42]. This involves the sorting of population into various non-domination levels. The algorithm selects the least dominated members for comparison and uses them as a reference to give velocities to the population and generating a new population set. It then takes the nondominated members of the new population set and places them into an elite set. This new elite set is merged with the previous nondominated members for the next generation. If there is still space remaining in the population, the best dominated members of the previous population are chosen to fill in the remaining spaces. All of this is performed in $O(mN^3)$ [44].

2.10 Using MOEAs in RTS AI Tactics

This research effort is a novel approach of tactical decision making problem. It utilizes various MOEA styles in order to micromanage unit positions more quickly and efficiently than a human can. Many current research efforts focus on optimizing one single factor of an RTS strategy or tactics through various learning techniques or evolutionary strategies. Instead, this research effort focuses on generating an acceptable solution through available real-time data. Analysis is performed through the testing of various MOEAs with different parameter settings to determine which one provides the best Pareto front. A properly programmed MOEA should find a set of unit positions that is close to optimal to maximize the utility of the resources invested into the individual units, providing a dynamic solution as opposed to a static one.

2.11 Chapter Summary

This chapter provides a summary of many concepts and approaches utilized in this project as well as an overview of the process of decision making in addition to its applications to the RTS decision making problem. Additionally, a comparison between strategic and tactical decision makings is described along with a variety of approaches to the tactical decision making problem. Finally, a variety of RTS platforms are explored and MOEA libraries are summarized in order to provide a frame of reference for the current utilization possibilities of this research. The following chapters provide an explanation of the process of generating an MOEA to create formation positioning in an RTS environment.

III. Methodology of the Positional MOEA

This chapter discusses the construction of the problem from a top down perspective. It begins with defining the problem and mathematically describing various aspects of the problem space. Then, an MOEA based algorithm is developed for that problem definition, using the mathematical metrics to determine the quality of the algorithm's solution. After an algorithm is selected, it is placed into a "sandbox" for testing, leading into an eventual integration with the Spring engine itself.

3.1 Phase 1 - Defining the Problem Space

Before a solution can be placed into the Spring engine and the AFIT agent, it is best to understand what exactly is being solved. The following sections are an explanation of the process behind defining the problem and the algorithmic solution from start to implementation.

Tactical Movement in RTS Games.

In the AFIT RTS AI agent, previous efforts focused on determining strategy effectiveness in optimizing the build orders [5] or tactical effectiveness through combat targeting [6]. While these have been effective improvements to the project, there has been no effort in concerning how units moved. What the AFIT RTS AI agent currently does to handle movement is to choose a lead unit and command every other unit to follow that one unit. Figure 12 demonstrates this operation. While simple, this causes some exploitable situations such as killing the leader unit. It also causes all the units following the leader to cluster together and their path finding techniques build into the game can interfere with one another to cause delays.

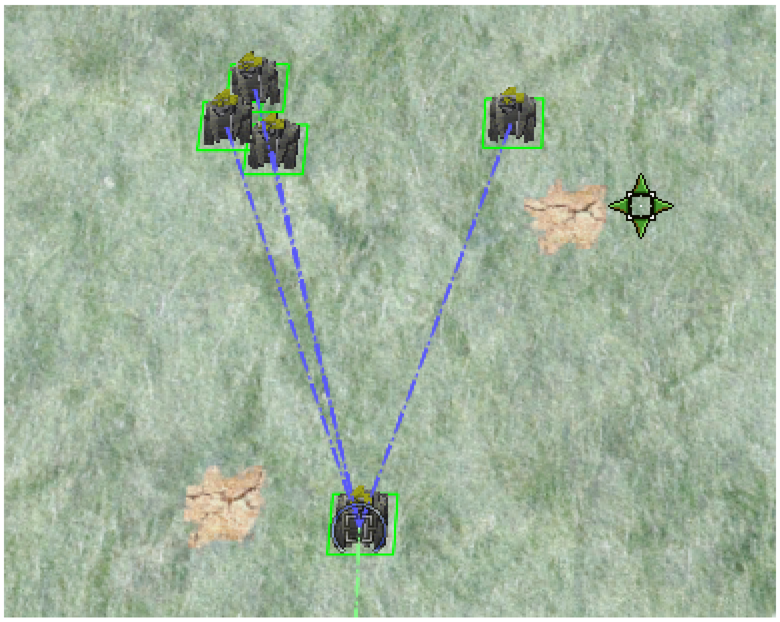


Figure 12. Example of a Leader Unit

This work aims to fix this problem is by spreading out units from one another. Proper spacing allows for several things to occur easier:

- Keeping units from clustering too close together. This alleviates the issue in RTS games of units cutting off another's path, as sometimes units can be dumb and try to drive through one another.
- Countering enemy Area-of-effect weaponry by minimizing the number of allied units hit per attack from the enemy
- Maximizing weapons firing at enemy targets while minimizing the amount of weapons the enemy units can fire on friendly targets.

High Level Design.

The following subsections discuss the high level design of the tactical positioning problem. The problem is defined mathematically in order to derive an algorithmic solution which is used to adequately evaluate a given scenario.

Problem Domain Description.

The Spring RTS engine primarily operates on an 2-dimensional coordinate plane, where each unit controlled by the players can be described by an integer coordinate pair. A third dimension exists within the game, but it is utilized for elevation purposes, which is unused this project. Each unit can move a certain distance within a particular unit in time, which provides a radius of possible locations for the unit to potentially end travel within said time unit. When presented with an enemy force, a player's units must take a combat a tactical stance based on the relationship of friendly forces to the enemys. The goal of this problem can be stated as: *Select the optimal configuration of allied unit destinations in relation to the enemy forces and each other, given a current position for each unit in the conflict.*

Problem Complexity Analysis.

Given that each unit has the same number of possible positions N, and there can be up to M units, then there are N^M combinations of positional configurations. In order to ensure that the perfect solution is found, it would take $O(N^M)$ operations to search the space.

This assumes that the displacements for each of the units are integers. If the (X,Y) displacements are in the form of real numbers, then there are an uncountably infinite number of possible combinations. In order to ensure that the calculations within this experiment can converge to a solution quicker, the displacements are utilizing a range of integers instead of a range of real numbers.

Definitions.

In order to fully define the objective functions, a list of terminologies is required to represent the problem space. The following definitions are used throughout subsequent equations in this document:

A:	A list of all allied units where $a_i(x, y) \in A$ provides the location of the individual unit on the map.
E:	A list of all enemy units where $e_i(x, y) \in E$ provides the location of the individual unit on the map.
D_E:	The desired Euclidean distance between an allied unit and an enemy unit. This is typically the range of the weapon on the allied unit.
D_A:	The desired interval Euclidean distance between allied units.
C_A:	Euclidean distance to the closest ally unit from the current unit
C_E:	Euclidean distance to the closest enemy unit from the current unit

Objective Functions.

With the general concepts of the problem defined in the previous section, it is possible to now define the various objective functions for optimization. This is required in order to develop an MOEA. This section outlines each of the objective function equations and an explanation of the purpose they serve in the agent.

OBJECTIVE 1: Distance from Enemy

Equation (1) describes a mean-squared error from all allied units to their closest enemy units. For this, the function uses all *known* enemy locations. This objective function reduces dependency on the removal of Fog of War from the battlefield, a technique typically implemented to greatly assist RTS agents [13]. This objective is implemented with a mean-squared error to create a natural flow towards more desirable positions. As the unit gets closer to the desired distance, the functional value lowers itself into a valley. Within the valley, there are numerous optimal values

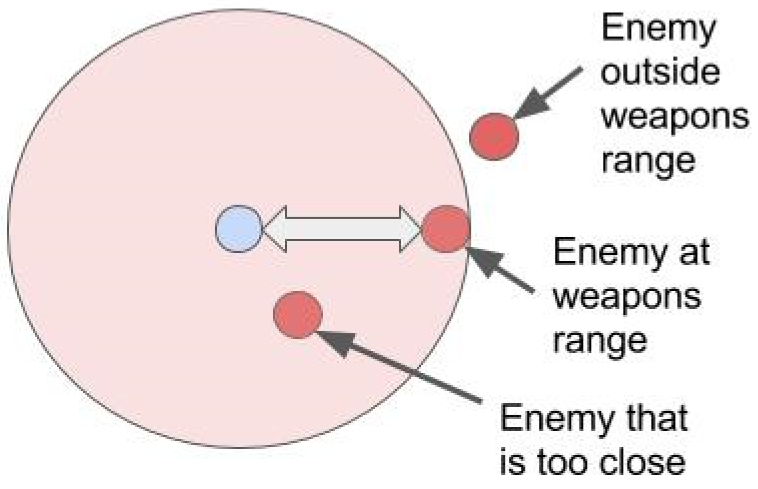


Figure 13. Demonstration of Objective 1: Distance From Enemy

that can be chosen but not all locations in the valley can be defined as the best as the floor of the valley is somewhat rugged. This can be visibly seen in Figure 14. This function operates similarly to the to the Boids concept of separation, as discussed in Section 2.8.

$$OBJ_1 = \frac{\sum^{|A|}(D_E - C_E)^2}{|A|} \tag{1}$$

OBJECTIVE 2: Distance between allies

This second objective function is similar to the first. The major difference being that instead of enemies it is spacing against, the function only cares about the allied units. The design of both functions (1) and (2) have dynamic algorithmic capabilities. This is that the function can be re-evaluated by adjusting the desired distance between units. This would allow for a more fluid adjustment of the agent in battle to situations as they occur. This can be visibly seen in Figure 15. This functions similarly to the Boids concept of cohesion, discussed in section 2.8.

$$OBJ_2 = \frac{\sum^{|A|}(D_A - C_A)^2}{|A|} \tag{2}$$

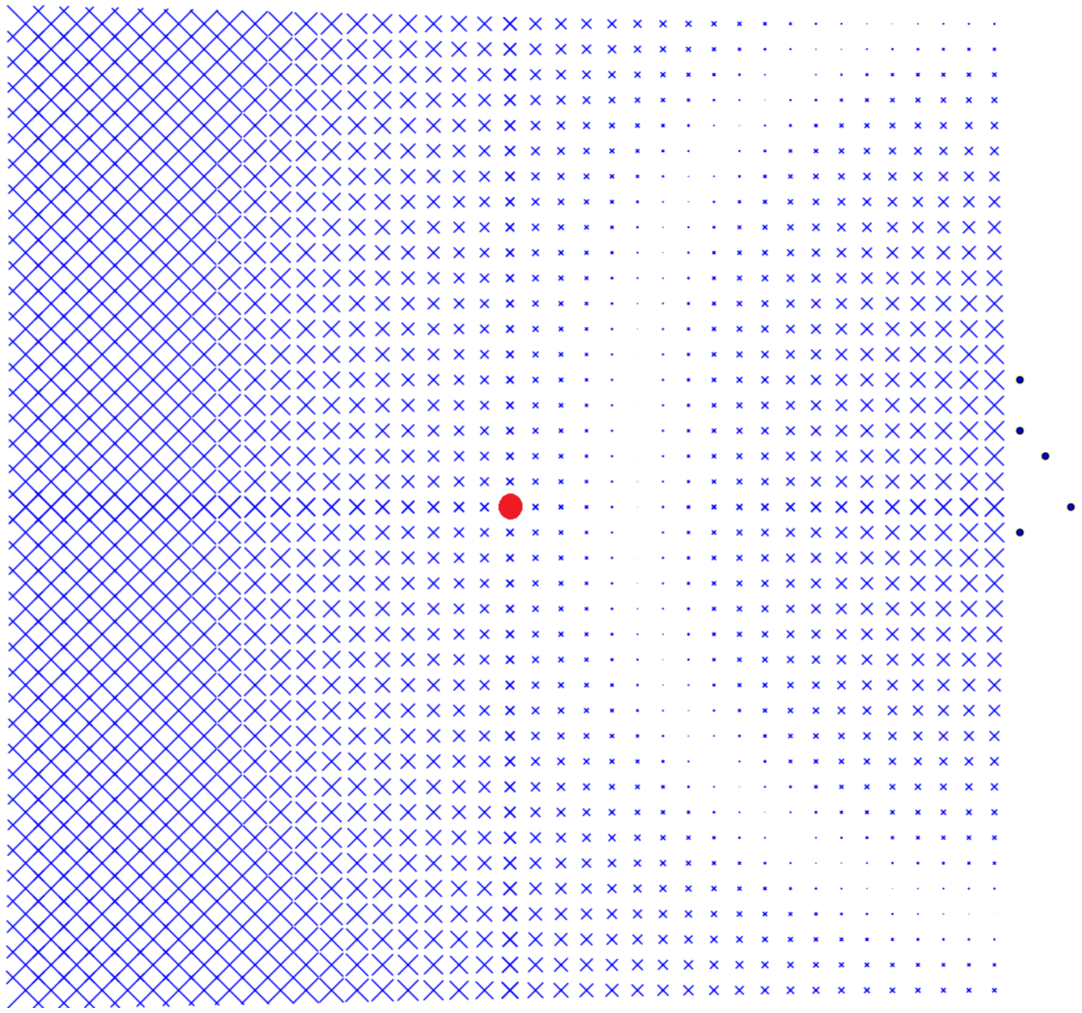


Figure 14. Heat Map of a Single Unit in Relation to Enemies. Each "X" has a size equal to the value of the objective. The white "trench" seen on the right side shows possible good areas of movement.

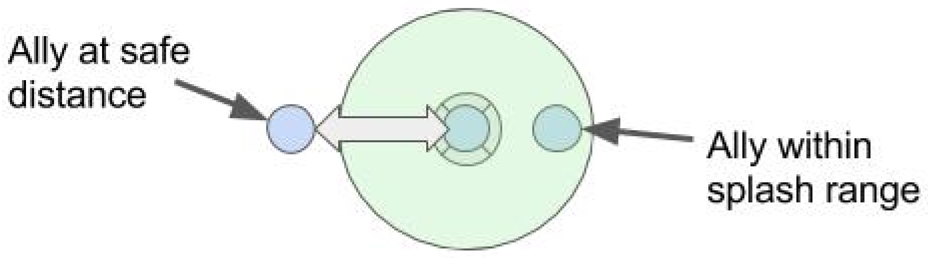


Figure 15. Demonstration of Objective 2: Distance Between Allies

$$OBJ_3 = OBJ_1 \times OBJ_2 \times \sum^{|A|} DistanceTraveled \tag{3}$$

OBJECTIVE 3 - Minimizing Distance Traveled

The third function's purpose is to prevent units from traveling farther than they need to in order to reach their destination. If a unit is traveling, then it cannot fire on the enemy. Therefore, if less effort is placed into movement then more effort can be placed into firing weapons. Additionally, this function prevents units from crossing paths needlessly. If left unchecked, two units that can both reach the same optimal locations could be chosen to travel to either. This also avoids an issue of the units having to navigate around one another, which adds to the transit time.

Putting it all together

All the objective functions together create a unique perspective for each unit. Each of these units are going to make their decisions based on two major factors: the distance to the closest enemy and the distance to the closest ally. (Figure 16). This, combined with minimizing the total distance traveled, guides the algorithm towards an effective solution.

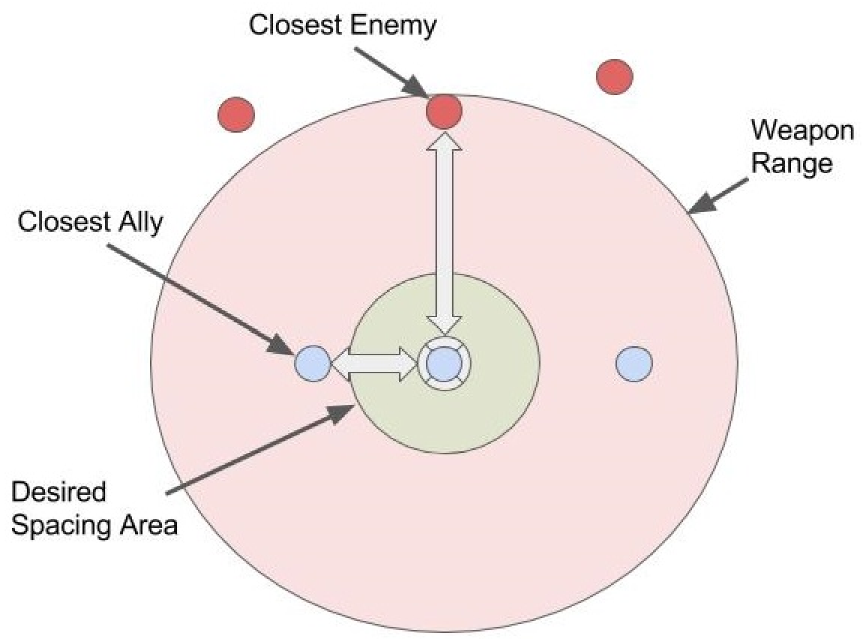


Figure 16. Visual demonstration of OBJ1 and OBJ2 placed together

Evaluation Algorithm.

Algorithm 1 demonstrates the algorithmic pseudo code for the objective functions. The purpose of this evaluation algorithm is to take the inputs of unit displacements and apply them to the current positions of allied units. Then return the evaluation functions in relation to not only the enemy units but also the other allied units.

Low Level Design.

With the problem domain defined, the next important step is to test via MOEAs if a solution can be found. Implementing it directly into the Spring engine does not provide immediate feedback for testing and debugging, therefore implementing it in an isolated environment helps to ensure that the algorithm is capable of solving simple static scenarios before transitioning it into a real-time environment of the Spring engine.

Algorithm 1 Analysis Algorithm for Objective Functions

```
1: Step A: Receive Inputs:
2:    → Starting allied locations $A$
3:    → Starting enemy locations $E$
4:    → Desired distances $D_A$ and $D_E$
5:    → Movement distance $M$
6:    → Decision variables $X$ (See step 2 for generation)
7: Step B: Define evaluators for each function
8:    → Apply decision variables to $A$ to receive
9:          new set of locations $A'$
10:   → Perform the actions in the functions below
11: Function 1:
12: for Each unit in $A'$ do
13:      Set $C_E$ to the smallest distance to a single enemy
14:      Add $(D_E - C_E)^2$ to result variable $F1$
15: end for
16:   Divide $F1$ by the number of units in $A'$
17: Function 2:
18: for Each unit in $A'$ do
19:      Set $C_A$ to the smallest distance to another ally
20:      Add $(D_E - C_E)^2$ to result variable $F2$
21: end for
22:   Divide $F2$ by the number of units in $A'$
23: Function 3:
24: for Each Unit in $A'$ do
25:      Calculate distance between $A'_i$ and $A_i$
26:      Add distance to result variable $F3$
27: end for
28: Step C: Return function values $F1$, $F2$, $F3$
```

Defining a Genetic String.

Where the previous section described a problem mathematically, a genetic string is required to implement an evolutionary algorithm. This can be done fairly simply by providing a set of displacements from the starting position of evaluation. The sequence can be represented with a string of numbers double the length of allied units in the squad. The first half of the string is a sequence of displacements in the X direction on a grid, the second half of the string is the Y displacements. This genetic sequence is represented in Equation 4.

$$[X_1, X_2, \ldots, X_N, Y_1, Y_2, \ldots, Y_N] \tag{4}$$

Creating a Sandbox.

The testing arena for the algorithm is a flat, 2 dimensional euclidean plane with no complex features. In a designed scenario, 5 units for each side are being pitched against each other. The positions are the real numbers on the plane and distances are the straight line distance between two locations. A scenario can be devised in this sandbox by providing the parameters of: unit locations, weapons ranges, movement distances and desired spacing. The result of a solution evaluation is an satisfactory positioning of all the units in the movement space.

MOEA Software Selection.

There exists a plethora of software that can solve multi-objective problems through the usage of MOEAs. Picking the correct software means not finding the one that is implementing these algorithms the best, but the one that streamlines integration and implementation into the Spring engine.

```
from PyGMO import *

prob = custom_problem()              #Declares problem to be evaluated
algo = algorithm.nsga_II(gen=5)      #Declares algorithm to use
pop = population(prob, 30)           #Declares population size
pop = algo.evolve(pop)               #Evolve the population with algorithm
```

Figure 17. An example of solving a problem through *PyGMO*

Gruber, who wrote the previous modifications to the AFIT agent, chose *PyGMO* [6] for his problem implementation. The reasonings given by Gruber in his thesis were that *PyGMO* is writen in Python, the same language of the AFIT agent, as well as having implementations of NSGA-II, SPEA2 and NSPSO [37].

Before beginning on this project, some time was taken to confirm Gruber's decision to utilize *PyGMO* in the AFIT agent. While the software feels incomplete in many areas, the algorithms that this research effort utilizes are implemented well. Implementing a new problem did not take much effort to accomplish and processing was incredibly fast. It is concluded that maintaining the usage of *PyGMO* is clearly the best option, as implementing a new piece of MOEA software with a different programing language into the AFIT RTS AI agent would take too much unnecessary effort to accomplish for very little benefit.

Using *PyGMO* is rather straight forward. First one defines a problem to be evaluated and an algorithm to be the evaluator while giving various parameters such as problem dimensionality to the problem and the number of generations in the algorithm. Next, a population is defined with the number of members and the problem in reference. Then, the algorithm evolves on the population to perform its operations. Figure 17 demonstrates this process in code. One thing of note is that this whole process takes only a few lines of code to accomplish, allowing for the flexibility of defining the problem or algorithm. This includes both items included in the *PyGMO* install as well as user defined problems and algorithms.

```
from PyGMO import *

prob = custom_problem()               #Declaring the problem
algo = algorithm.spea2(gen=10)        #Declaring algorithm for use
pop = population(prob,20)             #Declaring a population
isl = island(algo,pop)                #Placing the pop onto a single island
isl.evolve(1)                         #Evolving a singular island
archi = archipelago(algo,prob,8,20)    #Group of 8 islands w/ 20 members
archi.evolve(10)                      #Evolving the group 10 times
```

Figure 18. An example of islands and archipelagos in *PyGMO*

A unique implementation of solving problems in *PyGMO* is the islands and archipelagos method [37]. This allows for isolated evolutions of populations for more unique solutions. In *PyGMO*, using the islands evaluation method can be seen as in Figure 18.

Considered Algorithms for Testing.

For the purpose of this effort,comparison trials are performed to determine the best algorithm between NSGA-II, SPEA2 and NSPSO based on their computational performance and the ability to converge to a satisfactory solution.

These algorithms have been chosen amongst the MOEAs provided by *PyGMO* due to their quality and calculative performance, as discussed in Chapter II. Rejected MOEAs included the S-Metric EMOA, which provides a very rough and inaccurate solution due to the S-Metric approximation mechanism. Another rejected alternative was the MOEA-D, which in *PyGMO* is the Parallel Decomposition (PADE) algorithm. PADE has an issue as of this project where it would not function when provided a custom design.

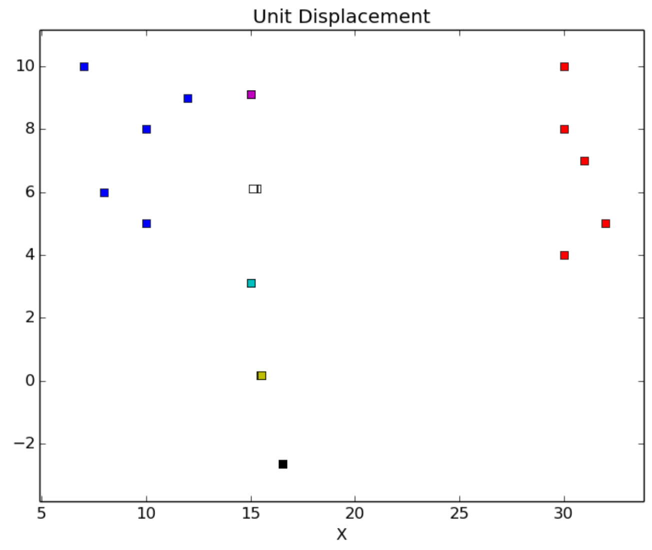


Figure 19. Example of Results from Sandbox Implementation

Structure of Results.

The sandbox results provide two valuable pieces of data: A visual representation of the output and various metrics for measurement. The metrics include execution time, hypervolume, spacing and non-domination count.

The objective is to utilize this evaluation technique to find an algorithm configuration that converges to a desirable solution in a short amount of time. If the solution takes too long to find, the resulting answers likely will be unsuitable for the changed scenario (Section 2.1).

3.2 Phase II - Integration into AFIT Agent

This phase takes the previously defined algorithm and implements it into AFIT RTS AI agent. It discusses the various modules within the AFIT RTS AI agent, as well as implementing the algorithm into the proper module. This evaluation also discusses how to issue commands to units within the Spring engine such as moving and attacking.

AFIT RTS AI Agent.

The AFIT RTS AI agent is a collection of projects provided by previous AFIT students, discussed in more detail. As it provides a working framework for interfacing with the Spring engine and uses various other tactics, a simple modification is all that is required to provide adequate testing.

The following sections discuss the structure of the AFIT RTS AI agent and how the algorithm designed in this project interfaces with it.

Structure/Design.

The agent used in this research effort is the AFIT RTS AI agent. Originally developed by DiTrapani [9], this agent is composed of several smaller modules that communicate to one another as it progresses through the game. DiTrapani's build allows for focus on both strategic and tactical levels of the game's execution. The various different structures, as seen in Figure 20, each have a unique role within process of execution of the agent. Activities such as managing units, controlling build orders, and determining the strategy are all important aspects required for a "skilled" play within an RTS game.

Gruber's research focused on development of the tactical aspect of the agent [6]. There is a python file entitled *group.py* that holds various tactical options. These tactical options start with simple scripts, such as attack closest, attack weakest, as well as more complex items like Gruber's targeting MOEA and this new positional MOEA.

There are several key pieces that make the tactical component work. Currently with the Fog of War turned off the agent knows exactly where the enemy units are at all times, therefore it can make decisions based on perfect information. Knowing this, there are some simple functions that are helpful provided by the agent's structure

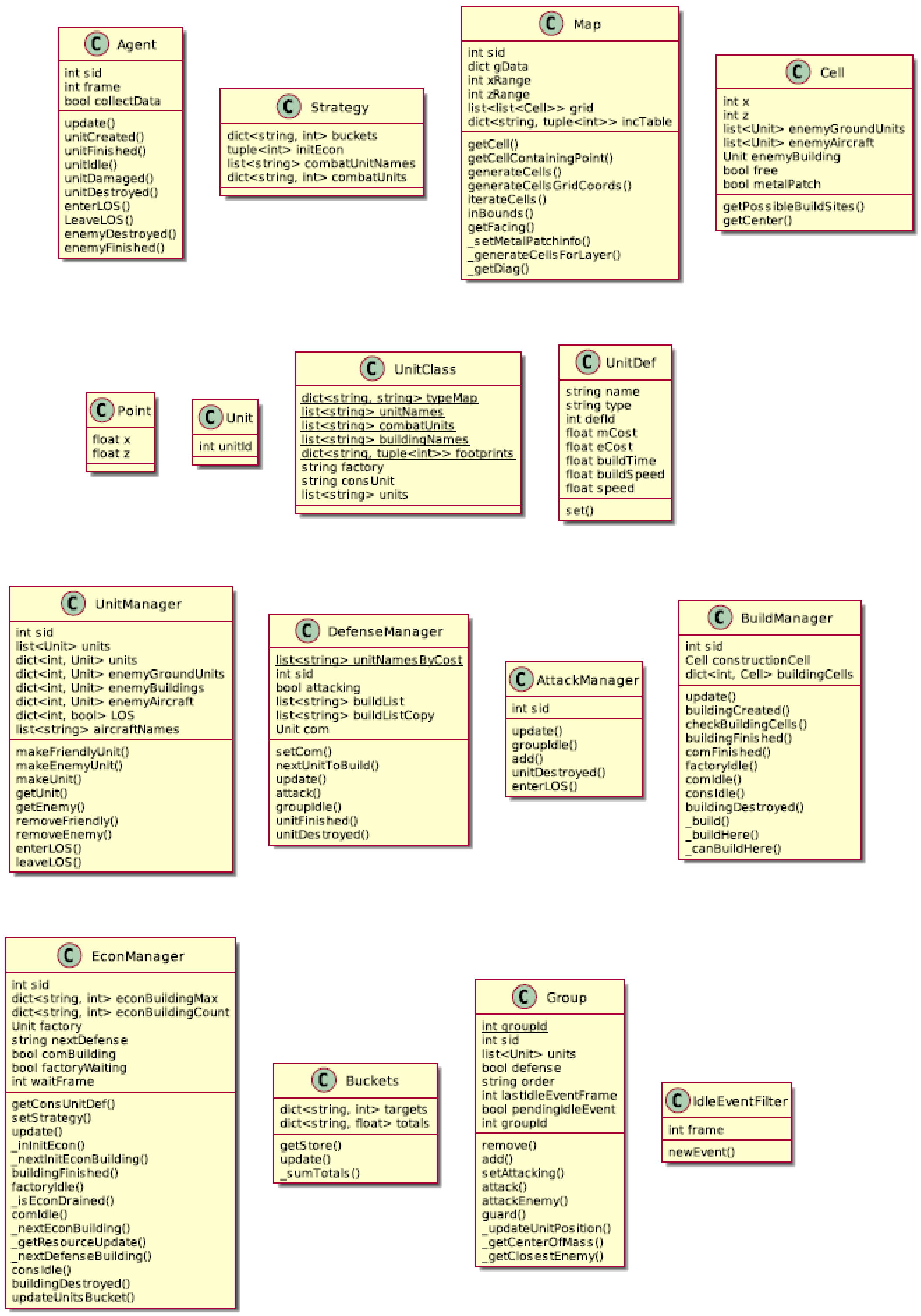


Figure 20. An overview of the AFIT agent's various components as developed by DiTrapani [9]

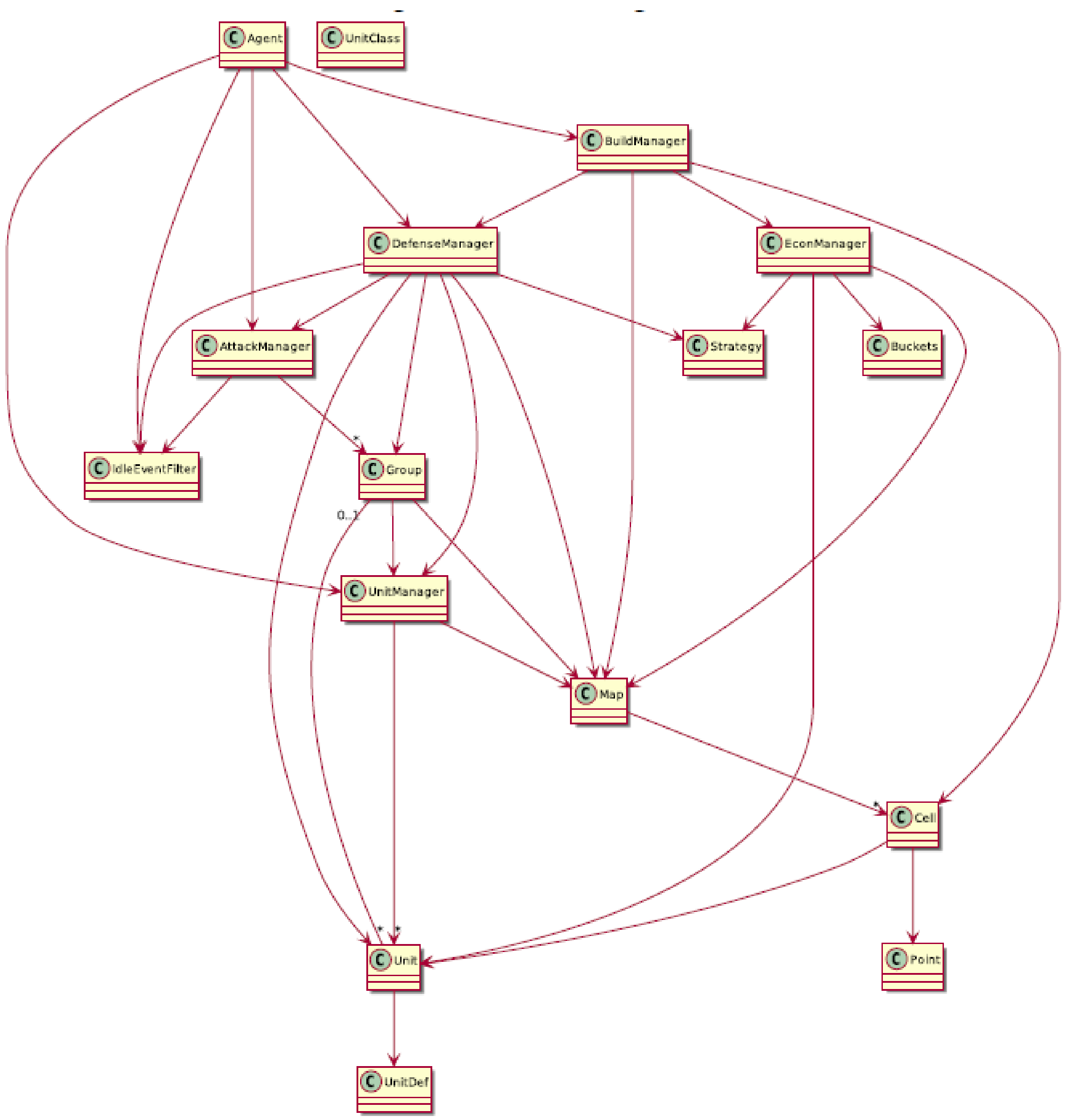


Figure 21. An overview of the connections between the AFIT agent's various components [9]

such as: *Assign move* or *Assign attack* as well as qualifiers if a unit is within Line of Sight of an enemy.

The AFIT RTS AI agent can also be configured for various other aspects beforehand. Things such as the number of players (which is always set to 2), the map being used for the experiment and the tactical strategies being employed can be adjusted within the configuration file.

PyGMO Integration with AFIT Agent.

With the effort to produce a PyGMO algorithm in the sandbox, the resulting algorithm can be treated as a black box where the inputs are passed in and the agent reads the results. This means the agent must translate the information present in the game into a format that is recognizable by the algorithm. For this problem, an $n \times 2$ array is built where n is the number of units.

The results from the PyGMO algorithm are determined by choosing the "champion" member of the population and applying those results to the locations that the units were when the calculation started.

Controlling Units in the Spring Engine.

The Spring engine maintains the several simple unit commands: Move and Attack. Many games implement more advanced commands, but in all units an RTS game can be commanded to do those two things.

In order to move a unit, the agent takes the results of the algorithm and adds them to the unit's current location to provide a new coordinate pair. Each of these pairs generated for the agent's units are then provided to the units through a move command (Figure 22).

```
# Attack Command
cdata.clb.attack(Scenario ID, Unit ID, Target ID)
# Movement Command
cdata.clb.move(Scenario ID, Unit ID, X Coordinate, Z Coordinate)
```

Figure 22. Issuing commands to units

3.3 Phase III - Real-time execution in Spring RTS

This final phase consists of performing functional testing to ensure that the algorithm performs as desired within the Spring engine. This consists of picking the map, analyzing the unit choice and a discussion on the "quirks" found while performance testing.

Functionality Testing in the Spring Engine.

With Phase II complete, the algorithm has been integrated into the AFIT Agent and is capable of performing basic functionality testing. The objective for this phase of development is to ensure that the agent is performing as desired. If the agent is not performing as desired, then there cannot be an effective experiment.

For this, the choices of map and unit used as well as a testing scenario must be established. Then, through running functional testing trials, factors such as movement range, distance from enemy and distance between allies can be determined to properly configure the agent and prepare it for experimentation.

Map Choice.

The map chosen for testing is the same map used in all previous AFIT Agent tests: The Pass. This map is a simple design where the north and south sides of the map are only connected by a narrow passage through impassable terrain. This passage creates a funnel effect and is prime for testing mechanics that involve tactical

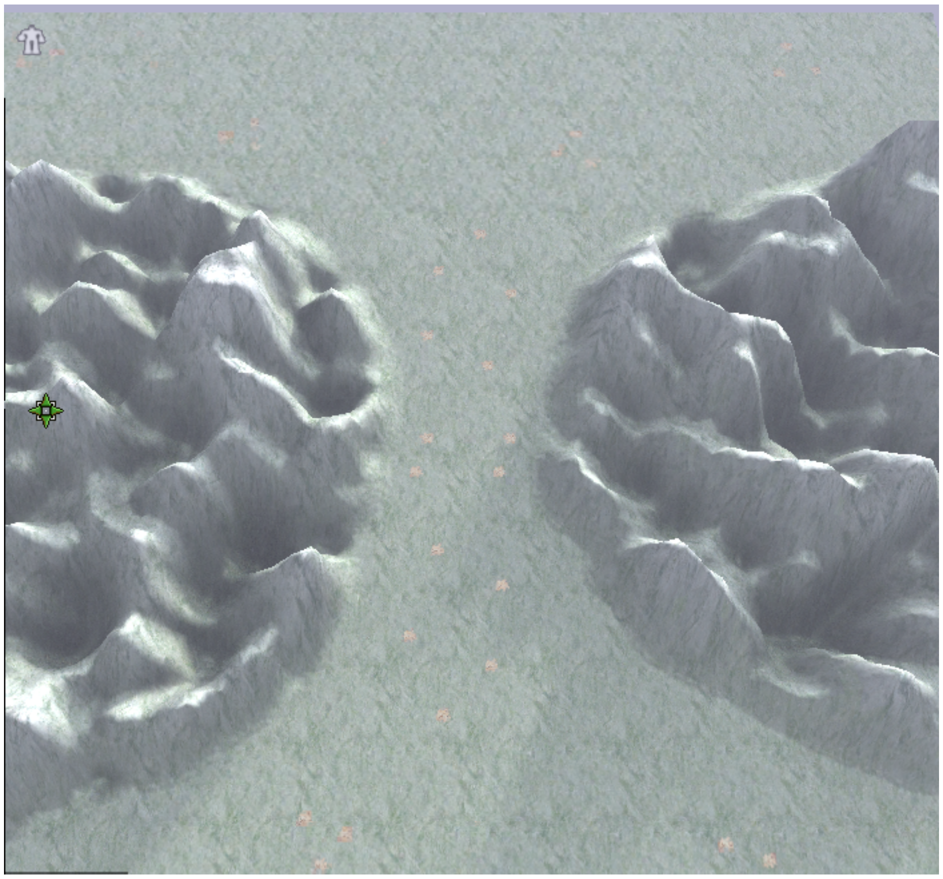


Figure 23. The testing map "The Pass"

combat, as enemy units are guaranteed to encounter one another at some point, as they cannot take different routes to avoid one another (Figure 23). While many other maps exist within the Balanced Annihilation mod of the Spring engine, this map is among the least complex amongst all the maps [46]

Unit Selection.

The unit of choice for this test is known as the stumpy tank [47]. This unit has an ordinance that explodes on impact to provide an area-of-effect damage capability. This capability allows for multiple enemy units to be damaged at the same time.

The current build order of the full AFIT agent necessitates the usage of this unit for testing. Blackford's build order agent [5] optimized building a set of stumpy tanks as quickly as possible. Given that the overall objective of this is for the AFIT Agent to utilize this algorithm on the tactical deployment of units, it's a prudent idea to ensure that the algorithm performs well with this focus.

Establishing a Scenario.

The scenario of choice for testing in the Spring engine is a pitched battle between equal sized forces. Each side has equal forces to ensure that both sides and tactical agents are on equal footing. At the beginning of simulation, each agent receives an equal number of units on opposite sides of the pass. The agents are then turned on, causing the units travel towards one another and fight against one another.

Each agent has Fog of War removed in order to know where the other agent's forces are located. Victory is be awarded to whichever agent has units remaining at the end of combat.

Adjustments for a Real Time Environment.

Through functional testing of the agent, it was noticeable that there were aspects of the Spring engine's real-time environment that could not be simulated in the sandbox. These aspects are described in the following sections.

Limiting Fast Agent Calls.

The AFIT agent calls this algorithm quite fast. So fast that the units are frequently given new move orders. This is fine if the units are not in position yet, but should the units be currently in position, then the new move orders actually interrupt the act of firing upon the enemy, as the unit is constantly attempting to move to a slightly new location and is distracted from firing.

To correct this, an artificial limiter is imposed on the agent to where it only calculates new positions once every N agent calls. After some experimental tuning, it was found that $N = 3$ is a good selection as a lower number would still be too quickly and a larger number would have the agent react too slowly to the changing environment. This allows the units within the game time to perform the orders given to them as well as reducing the computational lag imposed upon the computing system.

Integration with Gruber's Agent.

While this agent is capable of performing effective actions on its own, the Spring engine's default targeting algorithm for a unit can be a liability. In instances of testing it was observed that the selected target of the unit would transfer target seemingly randomly and for a less optimal choice.

To correct this issue, taking advantage of the targeting algorithm provided by Gruber in his work is a potentially effective option [6]. This is accomplished by evaluating the current position of the units and, if the units are deemed to be an acceptable position, utilize Gruber's targeting instead of calculating new moves. This alternate agent configuration is tested in the experimentation in addition to just the simple placement of units.

Impassable Terrain.

For this project, impassible terrain was not considered for the positioning of units for the sake of simplicity of the code. This could pose issues in operations in other maps, but this is assumed to be an unlikely scenario. The risks for such a thing are going to be assumed negligible, with the ability to update the code in the event that it does appear to be a problem in the future.

3.4 Chapter Summary

This chapter covered the method of building this project from the very bottom of problem description and mathematical representation up through low level pseudo code and eventual implementation within a sandbox as well as the Spring engine itself.

The mathematical representation helps define the problem space. With that information, the complexity can be observed and defining the algorithm to find a solution becomes easier. Translating a pseudo code for that algorithm provides a simple code that is testable within an isolated sandbox. This sandbox model can be modified and adjusted until it converges to an adequate solution. This completed algorithm is then imported into the Spring engine where, after a bit of adjustment for variables, it can then be utilized for experimentation.

IV. Design of Experiments

4.1 Introduction

This chapter discusses the construction of the experiments performed to evaluate the effectiveness of the positioning problem. It details the steps required and the metrics used for the decision point of the selected MOEA. Also, these experiments measure performance of the MOEA based tactical positioning algorithm as modules added to the AFIT RTS AI agent. All testing within this section was created with reference to the recommendations of Barr [?].

4.2 Experiment Test Equipment

A good method of maintaining consistency amongst experiments is knowing what equipment was used to provide these results. Having this knowledge can assist in any follow-on testing related to this project. Table 1 contains a quick listing of these items.

The actual environment was constructed within a Virtual Machine (VM) in an attempt to create a flexibility with moving between systems. This allows for several computers already running the more common Windows based platforms to run a Linux based system for experimentation. It also allows multiple machines to run the same software simply by copying an image between them, giving a flexibility to work on a project regardless of location. All tests were run on the same hardware set to prevent any discrepancies between processing power.

Table 1. **Overview of hardware and software used in the experiment**

Hardware	Model	Software	Version
Processor	i7-4770K @3.5GHz (4 Cores, 8 Logical)	VMWare	6.0.3 build-1895310
VM Allocated CPU	2 cores @ 3.5GHz	Ubuntu	14.04 LTS
Memory	32GB DDR3	PAGMO/PyGMO	1.1.5
VM Allocated RAM	4GB DDR3	Spring RTS	98.0.1-516-ga626219

4.3 First Experiment: Finding the best MOEA

First Experimental Objective.

This experiment covers the decision process of choosing an MOEA for this software. Since there are various MOEAs available from PyGMO [37], is prudent to identify which one performs the most effectively for this tactical positioning problem.

Test Establishment.

This test compares the effectiveness between: NSGA-II, SPEA2 and NSPSO. These algorithms are covered in more detail in Section 2.9. These algorithms were chosen because they are thoroughly implemented in the PyGMO software, where other, less popular algorithms are not guaranteed to perform as intended.

Sequence for Testing.

1. Establish metrics for evaluation
2. Run 10 instances for NSGA-II with positioning agent on team 0
3. Run 10 instances for NSGA-II with positioning agent on team 1
4. Compile results for NSGA-II for data analysis
5. Repeat steps 2-4 for SPEA2 and NSPSO

Each instance for an algorithm on a team is run 10 times, as an individual instance provides between 25 and 45 individual algorithm executions, providing approximately 800 executions between the different team positions. Additionally, 20 scenario runs provides a strong perspective on how the algorithm performs in terms of survival. Frequently, it is suggested that "30 test runs" are required for an adequate test, but that primarily implies that each test run only provides 1 data point per trial [48]. In this experiment, each trial provides multiple data points, one for each agent call during a trial.

Metrics for Analysis.

The following metrics are used to measure the success of the various MOEAS:

- **Remaining units alive:** How many units are alive at the end of combat? A negative number means the opponent won with that many units alive (i.e.: -3 represents opponent victory with three surviving units)

- **Hypervolume:** The volume of space between a reference point and the population's known Pareto front. As the known Pareto front travels closer to true Pareto front, it moves farther away from the reference point. This means a larger hypervolume number describes a solution that has traveled closer to the true Pareto front [49].

- **Spacing:** A measure of the average distance between population members on the Pareto Front. This spacing measure is a method to determine is clustering is occuring amongst the population. A larger spacing ensures a wider diversity amongst the population [50].

- **Non-Dominance Count:** A measure of how many members of the population are not dominated by any other member of the population. The assumption of this is that the more non-dominated members existing in the population, the closer the population is to the Pareto front [51].

- **Execution Time:** This measurement is used to distinguish the running times between each MOEA. Ideally, an algorithm should run as fast as possible. However, each calculation costs an action and algorithms with lower complexities sacrifice accuracy of a solution for fewer operations. This means there is a balance between a quality of solution and speed at which that solution is acquired. This is heavily influenced by the No Free Lunch theorem [52], where there isn't such a thing as a "free action" in search and everything has a cost.

- **Pareto Front Structure:** A visual product comparing the outputs of the objective functions to one another. This is done on a 3-dimensional scale, comparing all the functions to one another, as well as a 2-dimensional scale that compares each of the three objective functions to one another.

Structure of Results.

The results are reported in several parts. The first part is a new string into a log file. This string contains various metrics such as start time, execution time, and the values for each of the metrics discussed.

Additionally, two images are constructed. The first image is of the current positioning of the units on the map. The second image is the breakdown of the Pareto front.

Experimental Hypothesis.

During previous experiments in the sandbox, a non-real time environment, NSGA-II was found to converge the most effectively of the three MOEA algorithms [?]. The biggest concern is that the convergence was in a static environment. Being in a real-time environment means that calculations must be performed significantly faster and still provide adequate performance. If the calculation takes too long, the situation may have changed enough to where the calculated results are no longer relevant.

With a basic knowledge of how each MOEA operates, NSGA-II appears to be the best equipped to handle this problem. Despite its tendency to run longer than the other two algorithms, it also had the ability to converge more efficiently. By comparing each of three algorithms through a more detailed observation besides just an visual performance test, this experiment has two possible results of either confirming the prediction of validating NSGA-II being the most effective algorithm for this problem or displaying that another algorithm is more adequate for the problem.

4.4 Second Experiment: Determining Effectiveness of the Positioning Algorithm in the Spring Engine

Second Experimental Objective.

This experiment is purposed to evaluate the effectiveness of the tactical positioning algorithm in the AFIT RTS AI agent. By evaluating the performance via metrics, this test confirms if the algorithm designed for this research effort is effective enough for frequent tactical execution against an opponent.

Test Establishment.

This test compares the effectiveness of already implemented strategies within the AFIT agent against the new Positional MOEA and the Hybrid MOEA that combines the Positional and Targeting capabilities into one agent. It uses the following process to generate samples for evaluation:

1. Select a tactical option and pair it against each other tactical option.

2. Run the trial 15 times from one positioning (One Team 0, The other Team 1), then swap the teams and run another 15 tests.
3. For each trial, record the results from the metrics for the match up.

This experiment utilizes the following metrics to determine the effectiveness of a set of tactics:

- **Units Alive:** This is the average of the surviving units in a match up. A negative number would represent a loss with that many enemy units surviving and a value of zero would represent a tie (all combat units are dead).
- **Surviving Hit Points:** The total amount of hit points remaining amongst all the units of the tactics in that particular match up. A value of 0 means that the tactical option did not win.

Tactical Algorithms in the Second Experiment.

- **Default:** This is the Spring engine's default engagement algorithm.
- **Proximity:** All units are told to attack the nearest unit to the group
- **Weakest:** All units are told to attack the weakest unit in the enemy's formation
- **Targeting MOEA:** Gruber's targeting selection algorithm that utilizes an MOEA to choose the best combination of targets
- **Positioning MOEA:** This project's positioning focused MOEA that chooses the desired positions for each unit based on the enemy's formation.
- **Hybrid MOEA:** A combination of the targeting and positioning MOEA algorithms. In particular, the algorithm positions the units and once all units are in approximately the correct position, they are assigned targets via Gruber's targeting MOEA.

Structure of Results for the Second Experiment.

The AFIT Agent outputs the metric values in a simple JSON format [53] that can be imported into *RStudio* for analysis. Each algorithm outputs a snapshot of the opponent's unit status (Count, HP totals) while being identified with the timestamp of the current system time.

The results of the metrics are placed into a table where one tactical option is evaluated against another, different tactical option. A tactical option is not compared against itself, as it makes it difficult to differentiate the advantages of the scenario.

Experimental Hypothesis.

Through initial implementation testing of the algorithms, there is a noticeable improvement when the positional algorithm is placed against other algorithms. The noticeable downside is that the positional algorithm could be computationally intensive. There are two possibilities for this experiment, each of them represented as a hypothesis:

- **Hypothesis 1:** The positional MOEA alone provides a performance improvement for all other solo algorithms.
- **Hypothesis 2:** The hybrid MOEA provides a marginal improvement over the pure positional MOEA as well improve the computational lag that the pure positional algorithm introduces.

While the metrics cover all of Hypothesis 1, it only covers majority of Hypothesis 2. The remaining portion concerning computational lag is be somewhat subjective and requires additional evidence to distinguish between the two if there is no obvious difference in metric factors between the positional MOEA and the hybrid variant.

4.5 Summary of Experiments

This chapter covered the design of the experiments for this research effort. Both of these are important to determine not only the most effective MOEA for the positioning problem but also measuring the overall performance that this new tactical algorithm has when compared to other existing options.

V. Analysis of Results

5.1 Introduction

This chapter discusses the analysis of the results from the experiments performed in Chapter 4. The analysis begins with comparing and contrasting the effectiveness of the NSGA-II, SPEA2 and NSPSO algorithms against one another. The results of this analysis determines the most effective MOEA for the algorithm developed. Once the MOEA from the first experiment is chosen, it is then used to test its performance against other tactical options. The results of this second experiment demonstrate how effective the positional MOEA is for the AFIT RTS AI.

5.2 Results of MOEA Analysis Experiment

This section covers the analysis of various MOEA alternatives found within PyGMO. It compares the effectiveness of NSGA-II, SPEA2 and NSPSO. Each algorithm is compared by the structure of their Pareto front, hypervolume, spacing, non-dominance count, execution time and survival rate.

Quality of Pareto Fronts.

One thing to ensure before any further analysis is conducted is that the Pareto fronts generated by these MOEAs have a consistent quality. If there is little consistency between them, then the algorithm parameters should be reconsidered. The following segments are a visual evaluation of the Pareto fronts. Each sample was chosen is an example visual representative of the performance that both demonstrate good and bad qualities of the results.

The NSGA-II Pareto fronts, as shown in figure 24 for this problem tend to demonstrate a uniform, well distributed curve for the situation. Visually, the non-dominated

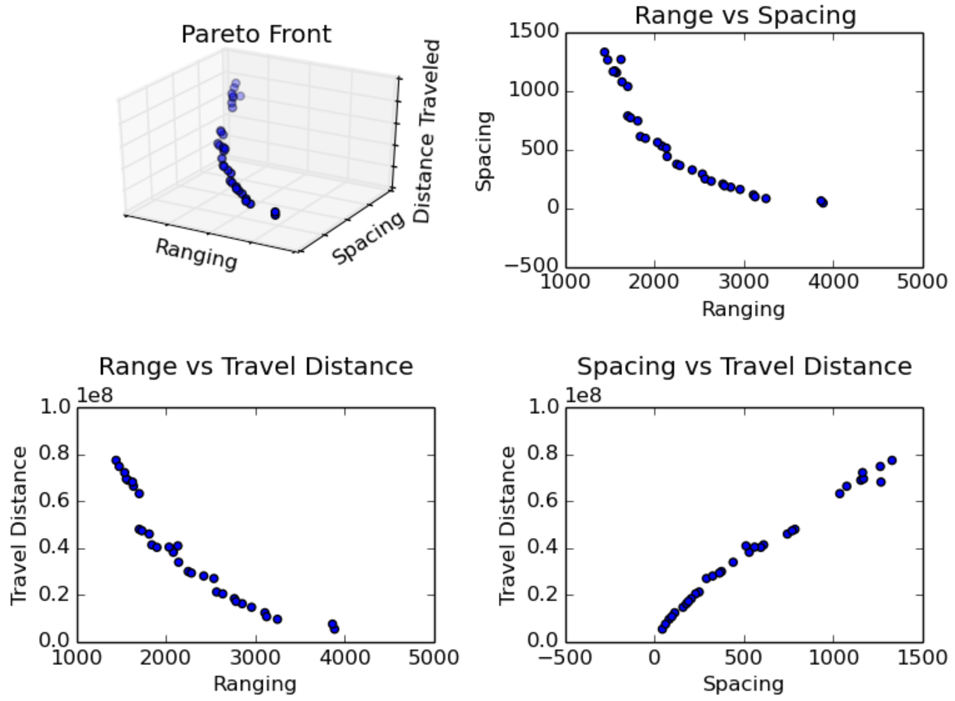


Figure 24. NSGA-II Sample Pareto Front

count appears to be high and outliers appear to be minimized. Overall, these fronts appear to be of a good quality.

The sample SPEA2 Pareto fronts (figure 25) tend to show a more fragmented curve. A clear decrease in the uniformity of the line can be observed as well as an inconsistency of the spacing. Visually, this Pareto front demonstrates an inconsistency of convergence, likely because the index used as ”good” solutions as well as the population size are both not big enough. Increasing those sizes would decrease computational performance, a very challenging trade off.

The NSPSO Sample, demonstrated in figure 26 shows a very loose Pareto front. While the algorithm maintains range consistently in relation to the other two objective functions, the relationship between Spacing and Travel Distance is very inconsistent. This is not surprising given the nature of NSPSO, as it is a very loose and fast algorithm. The speed can assist in providing more timely results.

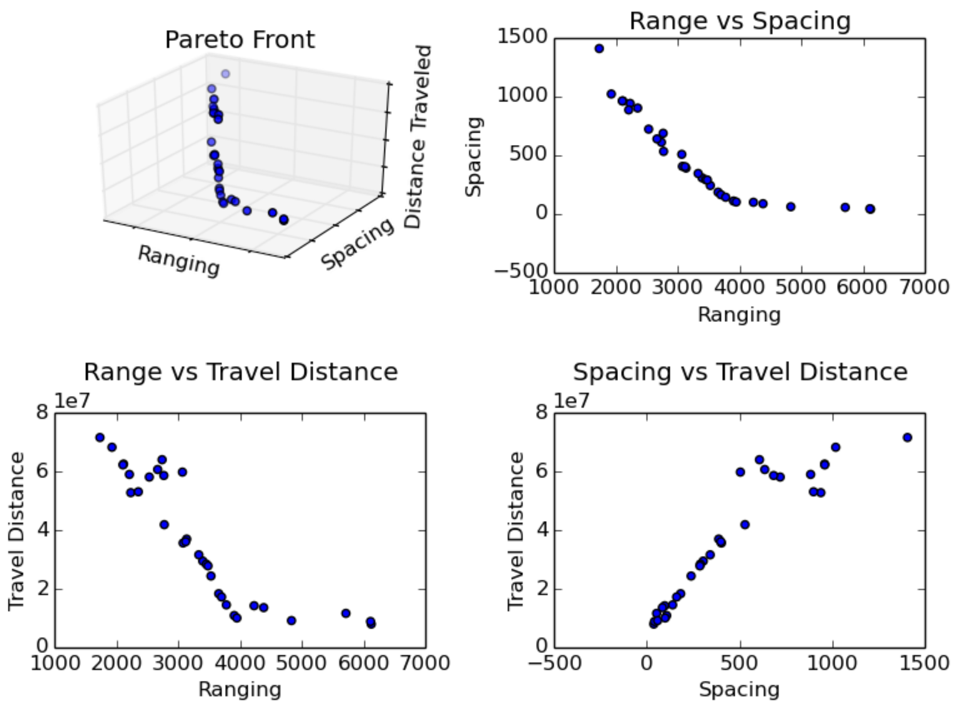


Figure 25. SPEA2 Sample Pareto Front

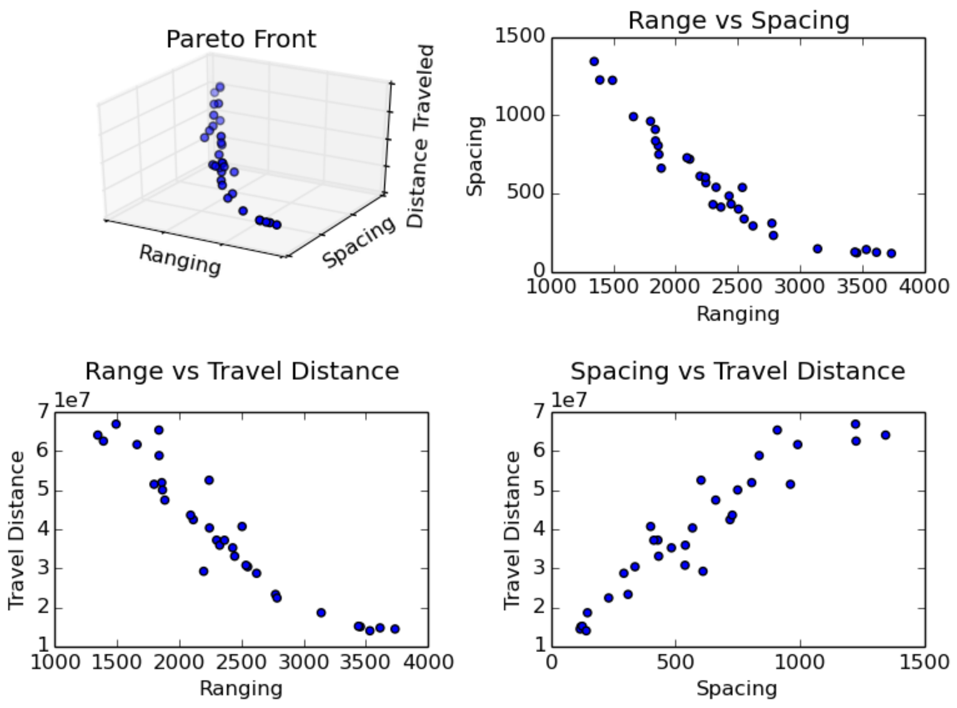


Figure 26. NSPSO Sample Pareto Front

Table 2. Statistical comparison of the Hypervolume results

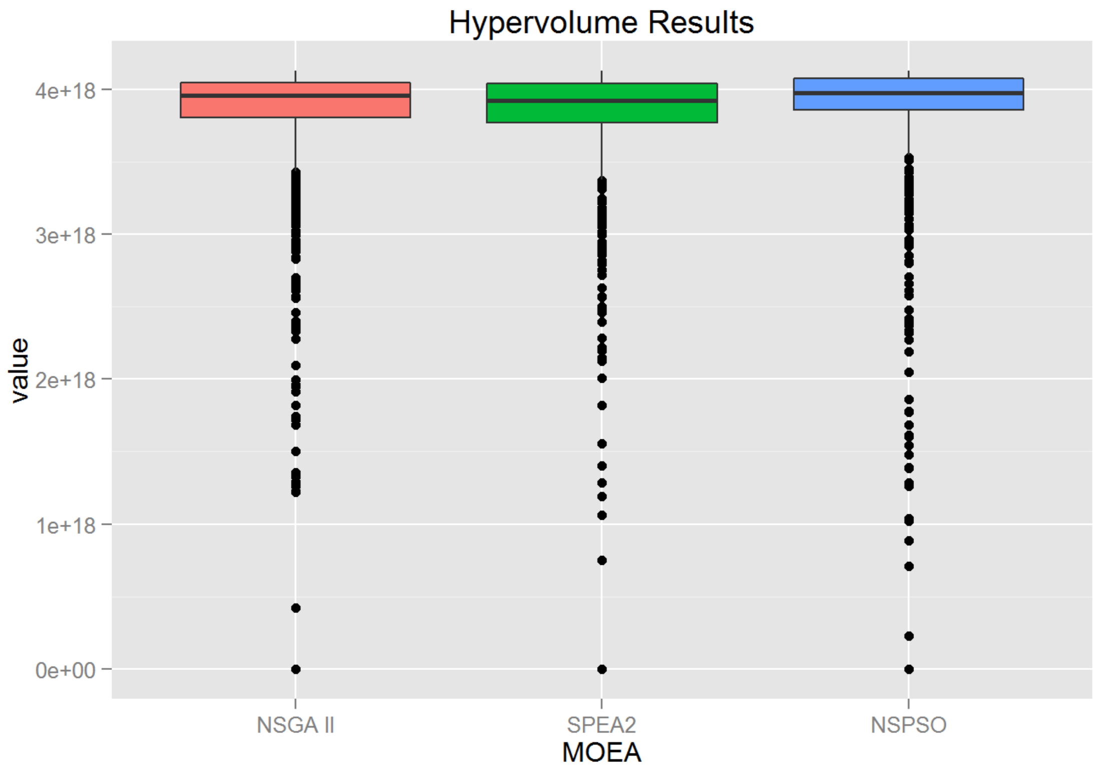
MOEA	Mean	Median	Variance
NSGA-II	3.732359e+18	3.955033e+18	5.228547e+35
SPEA2	3.742747e+18	3.922347e+18	4.125611e+35
NSPSO	3.824883e+18	3.976105e+18	3.235554e+35

Analyzing the Hypervolume Metric.

The Hypervolume results, visualized in figure 27 as a boxplot, when placed next to one another appear quite similar to one another. The top image in figure 27 shows the entirety of the dataset in a boxplot form. The trailing portion of outliers below the whiskers of the three boxes are ignored for the bottom figure in figure 27 to get a closer perspective on the bulk of the data points.

The outliers in the top portion of Figure 27 demonstrate the variety of possible hypervolume outcomes. A hypervolume measurement is taken from a reference point within the search space and measured towards the Pareto front. If the pareto front is closer to the reference point, the measurement is smaller. However, this also means the objective functions are also larger since both OBJ_1 and OBJ_2 are aggregates of the values of each unit's position, lowering the hypervolume value.

This analysis shows that the hypervolume metric doesn't really distinguish well between the various MOEAs. The primary issue of calculating the Hypervolume metric in *PyGMO* is that the reference point is "fixed" in the code. If any points in the Pareto front exceed the fixed reference point, the code errors out and does not provide a proper hypervolume metric. This means that there are values greater than represented in this dataset that were collected by the MOEA's operation. If this experiment were to be repeated, the hypervolume metric should likely be left out unless one is confident that no values can exceed the reference point. A suitably large reference point for this could likely be found, but then the resulting numbers would



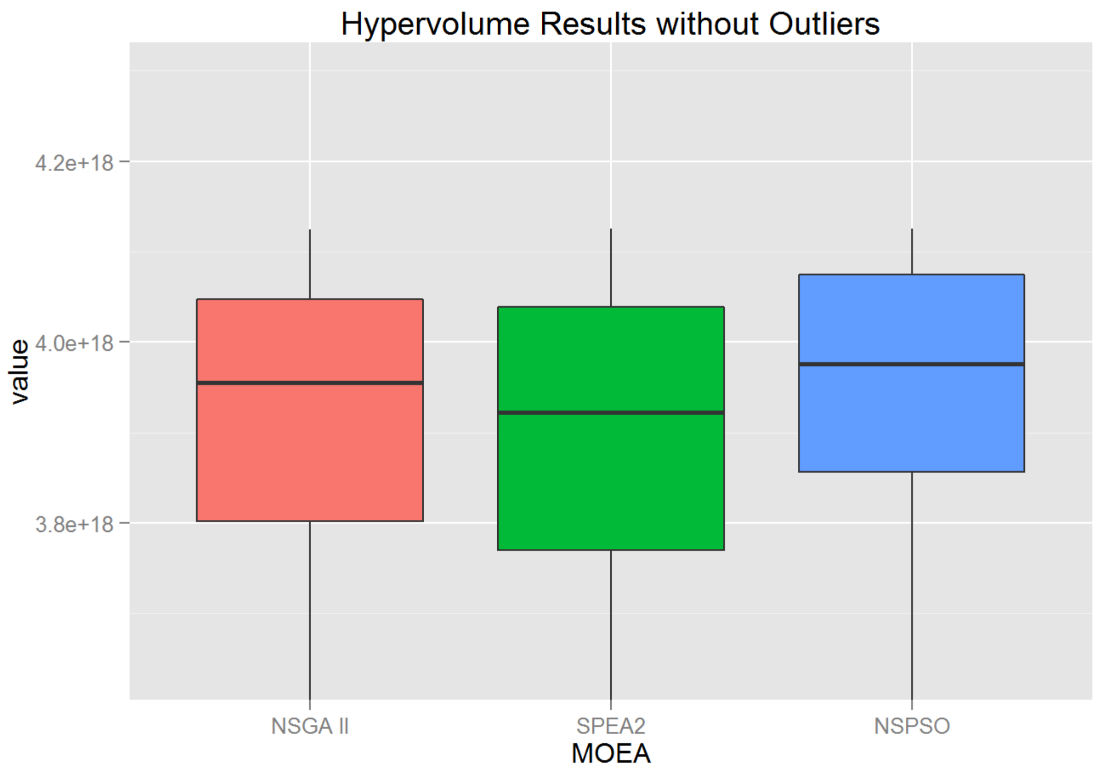


Figure 27. Above: Total look at hypervolume results; Below: Focused perspective of hypervolume results

Table 3. Statistical comparison of the spacing results

MOEA	Mean	Median	Variance
NSGA-II	73.20702	59.61942	4283.717
SPEA2	100.9999	66.63743	38513.53
NSPSO	137.6303	106.6501	14366.28

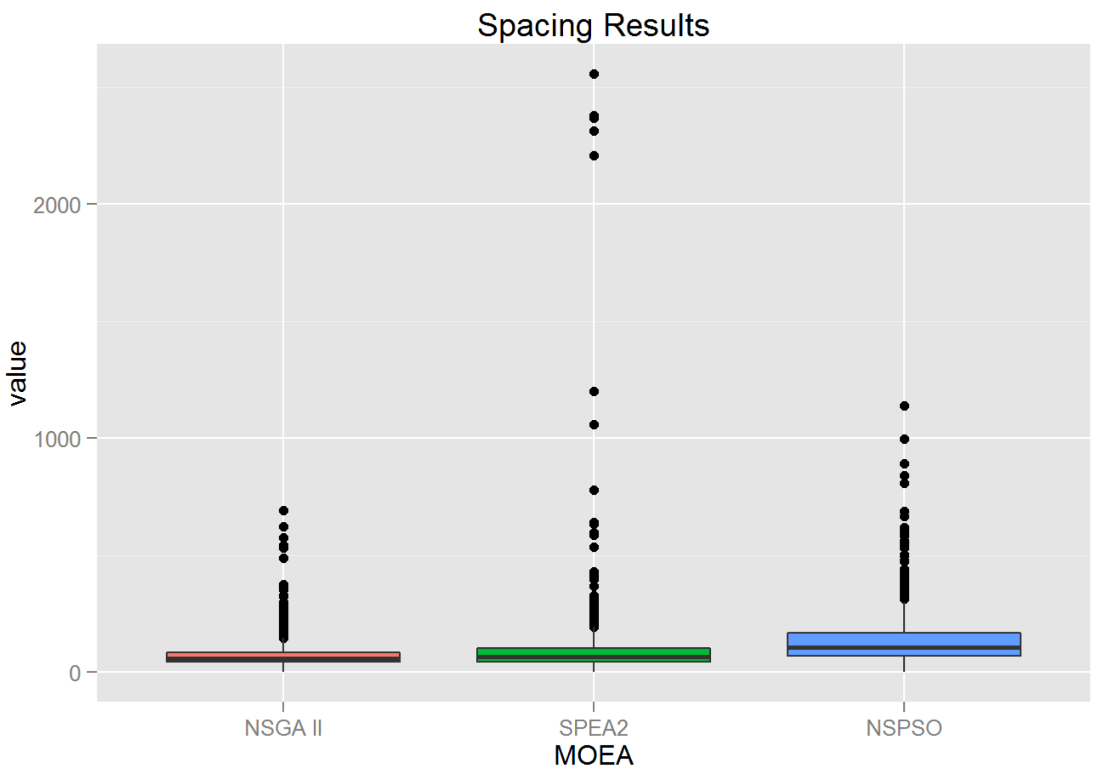
be unreasonably large and could possibly lose some fidelity when comparing them to one another.

Analyzing the Spacing Metric.

The spacing metric provides a consistent, uniform diversity amongst the population. In this metric, a lower number is more desirable as it demonstrates a population that is exploring a local area of the population space, allowing it to be guided towards a more desirable solution much easier.

Figure 28 demonstrates the spacing results of the experiment for each of the three MOEAs. In the bottom image in Figure 28, all three metrics are capable of presenting adequate spacing. However, the range in which the bulk of the values varies greatly amongst the MOEAs. NSPSO is clearly the least desirable as it has the largest range of spacing results. SPEA2 has a distinct improvement over NSPSO, but is not as effective as NSGA-II.

After analyzing these results of the spacing metric, it appears that this metric contains some distinguishing factors that can be used to differentiate the three MOEAs. In particular, the NSGA-II metric demonstrates a significantly more compact distribution of the results. In comparison, NSPSO has a much wider distribution of results and SPEA2 has the potential for outliers significantly worse than any other value within the testing dataset.



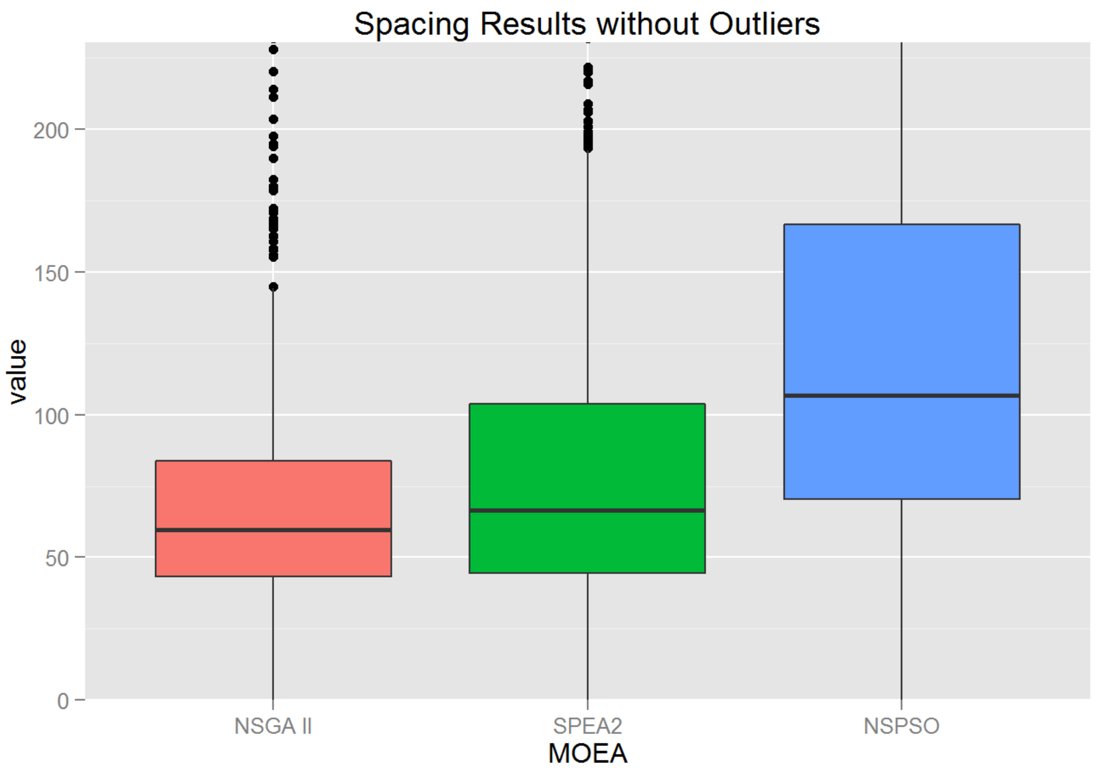
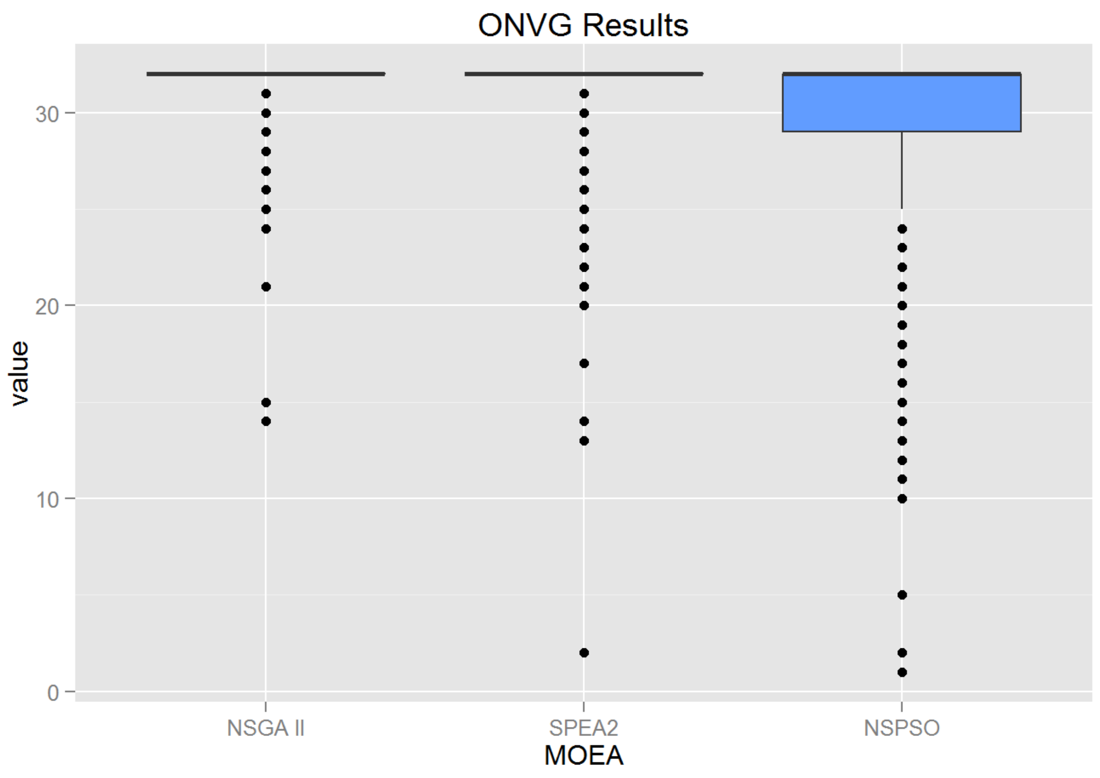


Figure 28. Above: Total look at spacing results; Below: Focused perspective of spacing results



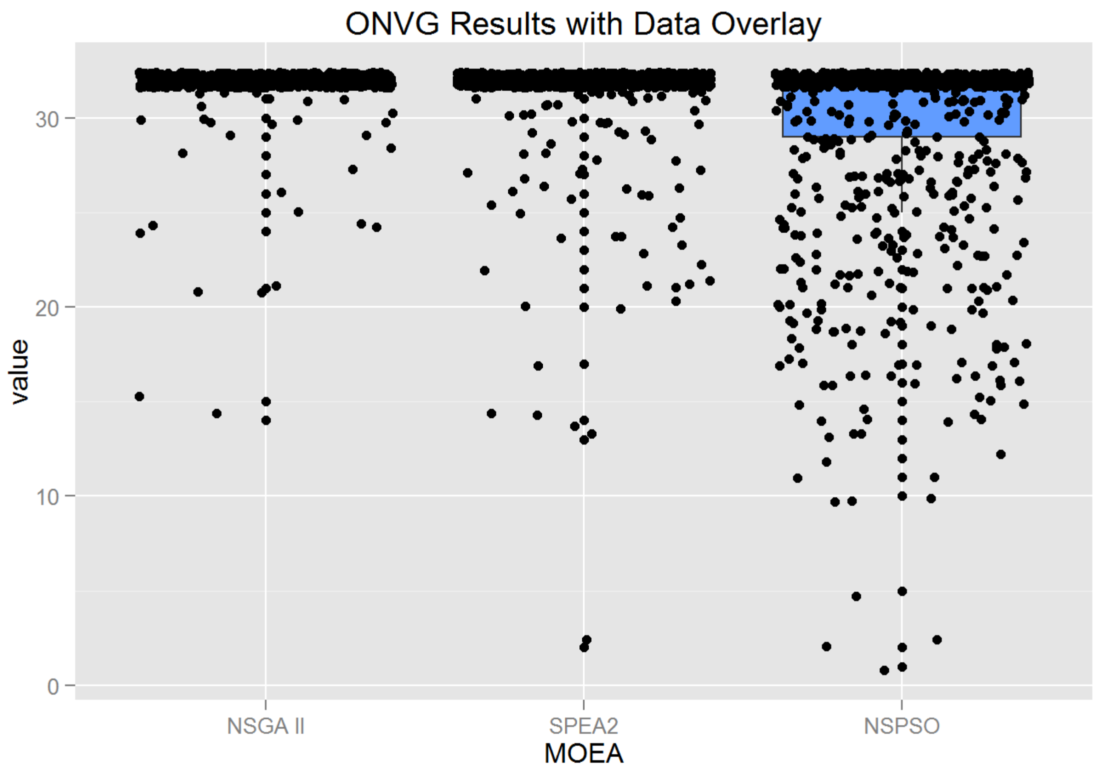


Figure 29. Above: Total look at ONVG results; Below: ONVG with data points overlayed

Table 4. Statistical comparison of the ONVG results

MOEA	Mean	Median	Variance
NSGA-II	31.82373	32	1.614799
SPEA2	31.51152	32	5.165957
NSPSO	29.40783	32	25.55205

Table 5. Statistical comparison of the execution time results

MOEA	Mean	Median	Variance	Shortest	Longest
NSGA-II	$46\mu s$	$42.9\mu s$	$0.37ns$	$25.9\mu s$	$333\mu s$
SPEA2	$42.4\mu s$	$39.8\mu s$	$0.89ns$	$24\mu s$	$586\mu s$
NSPSO	$40.2\mu s$	$39.1\mu s$	$0.11ns$	$24.7\mu s$	$159ns$

Analyzing Non-Dominance Count.

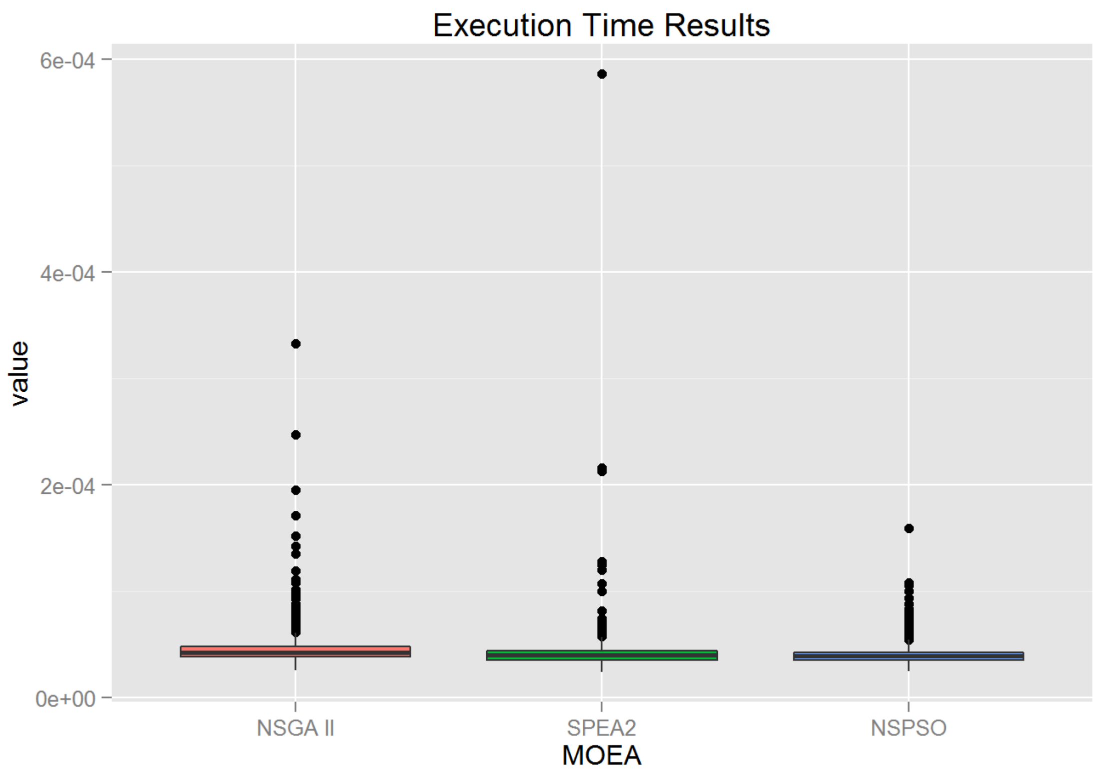
The Overall Non-Dominated Vector Generation (ONVG) is a count of all non-dominated members in PF_{KNOWN}. This count is useful because it demonstrates how many solutions in the population aren't rendered useless because a definitively better solution exists in the population.

Figure 29 demonstrates the results of the ONVG metric in this experiment. The top figure is a boxplot of the results and the bottom is the data overlayed ontop of it to demonstrate the spread of the data points throughout the results.

Both plots in figure 29 demonstrate a clear inadequacy of the NSPSO algorithm in this metric. This is not a surprising outcome, as the nature of a PSO algorithm involves a very loose population that is exploring the local search space [45]. Between SPEA2 and NSGA-II, NSGA-II performs better as it displays a noticeably lower variance than SPEA2 (Table 4)

Analyzing the Execution Time.

The execution time metric measures how much time an MOEA is spending on calculations. When the algorithm is calculating new positions, a faster calculation helps by not slowing the simulation down to the point to where the game itself is



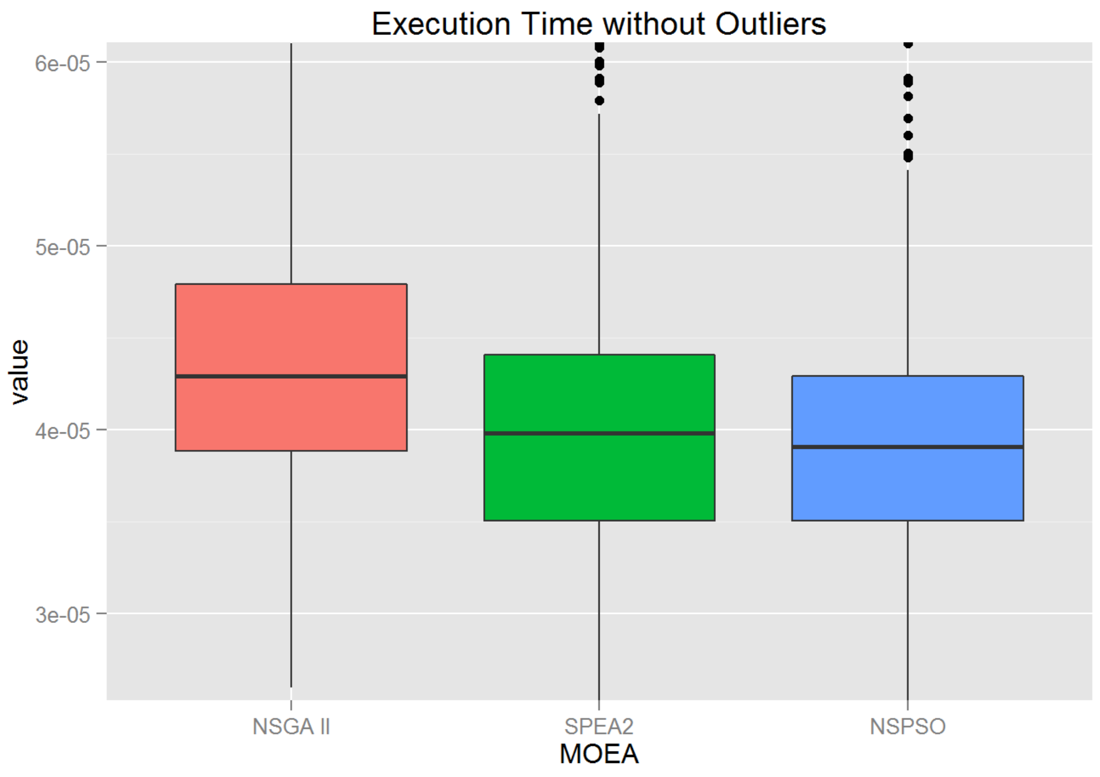


Figure 30. Above: Total look at execution time results; Below: Focused perspective of execution time results

unplayable. Additionally, if a calculation takes too long to perform it may become ineffective when applied as the situation may have changed to where the solution is no longer relevant.

Figure 30 shows boxplots of the entire data set (top) and a focused perspective on the dense part of the data set (bottom). The data is gathered using Python's *time.time*() function, which is a floating point number of the seconds since 1 January 1970 [54]. The results can be best represented in microseconds, where the spread can be seen in Table 5. Additionally, in the below of figure 30, it can be seen that the diversity of the execution times is fairly close to one another. However, there is definitely a clear differentiation between the three where NSPSO is the best followed by SPEA2 then NSGA-II.

Fundamentally, these results are unsurprising as the actual running time of these MOEAs reflect this as well. NSPSO consistently runs much faster than all the others. Even in it's slowest iteration, it performed 6 times slower than it's fastest. SPEA2 is the most diverse. On average, it performed middle of the road but had a significant amount of diversity. This is incredibly inconsistent nature of SPEA2 makes it quite undesirable. Finally, NSGA-II is consistently slower than the other three. However, it has a respectably low variance between the various iterations. In order for NSGA-II to be selected as the best option, other metrics must perform vastly better to justify the longer calculation time.

What this metric did is to confirm that the implementation of these MOEAs within *PyGMO* were done properly. If one algorithm had been chosen but turned out to have been implemented inefficiently, then this would cause much more delay than it should.

Table 6. Overview of the MOEA survival rates at the end of battle

MOEA	Mean	Maximum	Minimum
NSGA-II	4.7	7	3
SPEA2	3.5	6	-2
NSPSO	3.6	7	-2

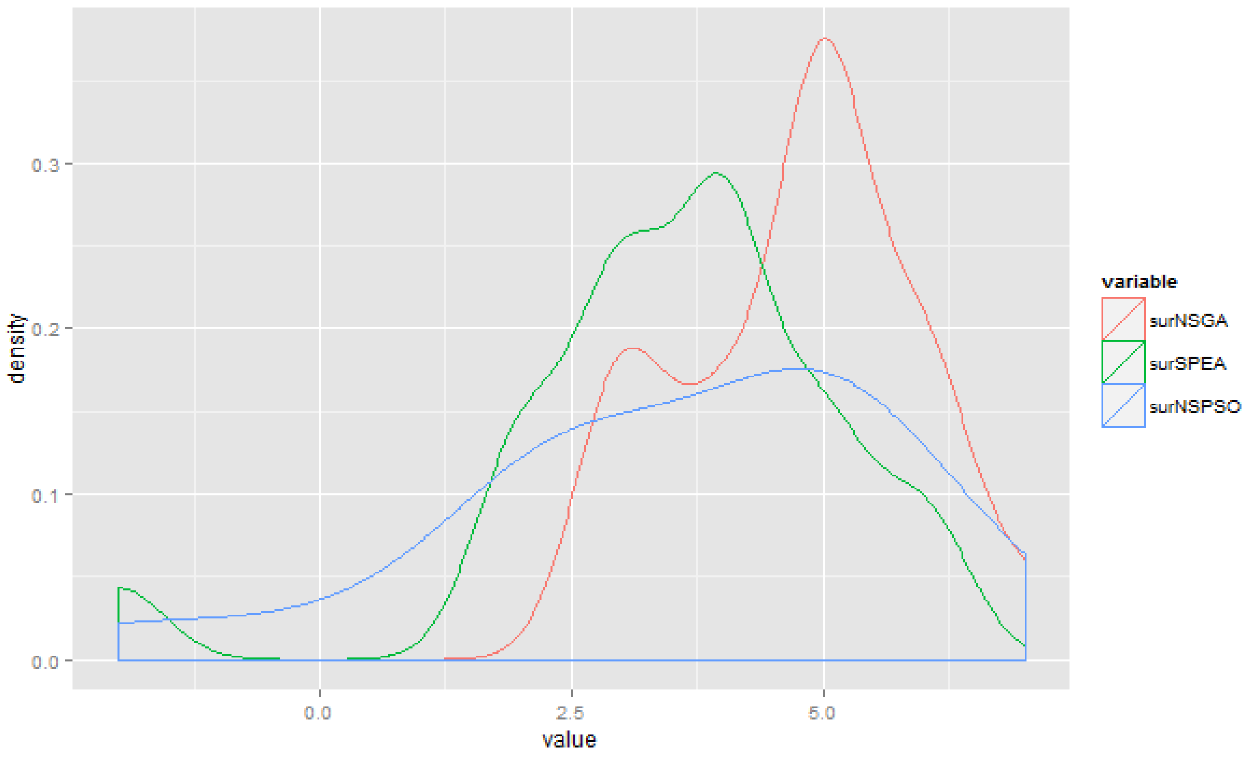


Figure 31. Comparison of the survival rate of the three MOEAs in Experiment 1

Table 7. A ranking of MOEAs according to metric results

	NSGA-II	SPEA2	NSPSO
Hypervolume	2	3	1
Spacing	1	2	3
ONVG	1	2	3
Execution Time	3	2	1
Survival Rate	1	3	2

Analyzing Survival Rate.

Survival Rate is recorded in a very simple manner, represented in Table 6. In the table, the number represents the number of surviving units at the end of the skirmish. If the number is negative, that means that the opponent won with that many units surviving.

These results show a clear result of NSGA-II performing well in this scenario with an expected survival rate of one whole unit greater than either of the other two MOEAs. Additionally, NSGA-II never lost a simulation in this experiment, as shown by the worst survival of 3 units. This demonstrates that while the NSGA-II algorithm does tend to run slower than the other two, as seen in the execution time results, the solutions it is choosing are consistently better.

Figure 31 shows the comparison of the three MOEAs survival rate as a histogram. It demonstrates NSGA-II has a higher density of large survival numbers. NSPSO displays a more diverse spread of results, and SPEA-II demonstrates a strictly worse output than NSGA-II.

Decision and Observations.

Table 7 ranks each of the three MOEAs against one another for all the metrics. Clearly, even without weighting of the metrics, NSGA-II demonstrates the most effective MOEA amongst the group. Differentiating between SPEA2 and NSPSO is difficult, but unnecessary as neither are the primary choice.

One major observation of these solutions is that the computational lag observed from this positional algorithm is not coming from the actual MOEA computation itself but the input buffer for commands into the Spring engine. While the MOEA does have some impact, as there is noticeable lag at higher generational counts for the algorithm, reducing the frequency of the agent calls (discussed in Section 3.4) removed the occurrences buffer errors in the Spring engine and cleaned up the lag.

Another major observation is the fine control of units, or the lack thereof. Unit control in the Spring engine consists of two commands: Attack and Move. When a unit is told to move, it attempts to move towards that location in only a forward motion. This means that if the desired destination is behind the unit, it makes a big, wide turn to achieve their position. If another unit is in the way, the moving unit rams into it and attempts to push through it. If two moving units collide with one another, they struggle to pass through one another in a conflict that much resembles a sumo match. Whenever this happens, it does impact the performance and can cause multiple units to bunch up together and become counterproductive to the goal of the positioning algorithm.

5.3 Results of Positioning Problem Effectiveness Experiment

This section analyzes of the results produced from the effectiveness experiment. The raw results of the experiment can be found in Appendix A.

Table 8 displays the win-loss numbers of each match pairing across the 15 matches for that configuration, with ties not recorded as a win for either side. In this table, it's fairly apparent the effectiveness of the MOEA strategies, as they typically have an overwhelming win total in relation to their opposition. Tables 9 and 10 displays this as a win % across all matches. Overall, the Positional MOEA performed overwelmingly

Table 8. Win/Loss Results of Experiment 2

Top \ Bottom	Default	Weak	Proximity	Target MOEA	Position MOEA	Hybrid MOEA
Default	N/A	10-5	7-8	7-8	1-14	0-15
Weak	3-12	N/A	1-14	0-15	0-15	0-15
Proximity	4-11	10-4	N/A	7-6	4-11	1-14
Target MOEA	7-7	10-4	11-5	N/A	2-13	1-14
Positional MOEA	13-2	15-0	13-2	13-2	N/A	10-5
Hybrid MOEA	13-2	15-0	9-5	13-1	5-8	N/A

Table 9. Win Percentages for overall performance as well as against the Default, Weak and Proximity strategies. Bold text is the highest against that particular strategy

	Win %			
Method	Overall	vs Default	vs Weak	vs Proximity
Default	56%	–	73%	40%
Weak	11%	16%	–	16 %
Proximity	40%	40%	80%	–
Targeting MOEA	42%	50%	83%	56%
Positional MOEA	**83%**	90%	**100%**	**80%**
Hybrid MOEA	78%	**93%**	**100%**	76%

Table 10. Match up percentages vs Targeting, Positional and Hybrid MOEAs. Bold text is the best percentage against that method

	Win %		
Method	vs Targeting MOEA	vs Positional MOEA	vs Hybrid MOEA
Default	46%	10%	6%
Weak	13%	0%	0 %
Proximity	40%	20%	20%
Targeting MOEA	–	13%	6%
Positional MOEA	86%	–	**60%**
Hybrid MOEA	**90%**	**33%**	–

well, with the Hybrid MOEA not far behind. Surprisingly, the Targeting MOEA did not perform as well as expected.

Observed Weaknesses of Targeting MOEA.

The targeting MOEA had a very surprising underperformance compared to the results from Gruber's thesis [6]. There are several notable differences between this experiment and Gruber's, in particular the scale is not the same. This experiment chose a 10 vs 10 matchup, as opposed to Gruber's 25 vs 25. The rationale for the smaller unit count was that the positional MOEA struggled to adequately position units in sizes larger than 10 within a reasonable time. This is because the complexity of the positional MOEA is, at best, an $O(N^M)$ where M is the number of units and N is an integer number of possible positions for each unit. Compared to Gruber's algorithm, which is solving an NP Hard problem [6].

Finally, there is an issue of hanging when the targeting MOEA engages the enemy commander unit, which while it did not impact the results of this experiment, it is a curious development. This anomaly appears once the targeting MOEA's forces win their engagement and begin their attack on the enemy's commander unit. Upon closing within weapons range, one of the agent's units will fire and the simulation hangs, completing no more calculations.

Observed Behaviors of Positioning MOEA.

The positioning MOEA had some curious behaviors that appear upon execution. The algorithm functions correctly, as it determines the positions required for the units to be located, but the methods the units within the game take to travel to those locations are not the best. The units themselves are quite dumb when traveling, requiring faith that they will not encounter any problems. Some results from the built-in movement algorithm include, but are not limited to:

- Never traveling in reverse, instead forward and turning to their destination. Additionally, since the tanks choose to only travel at their maximum speed, their turning radii are overly large.
- Having no spacial awareness of where other tanks are located, causing them to run into one another. This counters the entire point of the algorithm
- Not moving when the distance to the new location is short. This is an acceptable and somewhat desirable behavior that allows the tank to stay still and continuing firing because of issues with engaging while moving.
- Moving and firing is a challenge for the default unit movement script in the Spring engine. When a unit receives an order to move, it centers its turret to the "forward" position, where the turret is pointing in the same direction the tank is traveling. Then as it travels, it reengages the enemy in movement, at a reduced accuracy. However, once it stops, it centers the turret yet again and then reengages the enemy. This action creates downtime as it cannot fire while the turret is reseting its turret to the centered position.

With all of these problems, it would take quite an effort to actually correct them within the source code for the game, either in Balanced Annihilation or the Spring engine itself. While this may prove to correct the issues, it could also invalidate previous work on the AFIT agent, as it is designed with the current concepts of movement in place.

Why the Positional MOEA/Hybrid MOEA Works.

This new positional assignment system works because of two major factors working in its favor:

1. Reducing the impact of enemy area of effect weapons by minimizing the number of targets affected by the splash damage
2. Exploitation of own area of effect weapons by luring the enemy forces to become closer together in a dense fashion

Figure 32 demonstrates the typical sequence of a battle for the Positional MOEA against the Default tactical option. As the battle flows, it can be observed that the enemy units tend to get grouped up closely on the left side. This makes those units an easy target that can be destroyed quickly. In comparison, the enemy force focuses the targets in the middle. This then divides their fire as there are targets on either side of the middle.

5.4 Summary

This chapter presented the results of the experiments of this research effort. Once it was determined that NSGA-II was the best algorithm to perform the Positioning MOEA, it was placed into the AFIT RTS AI agent's code and demonstrated that it performed exceptionally well against all other tactical options currently existing in the AFIT RTS AI agent.

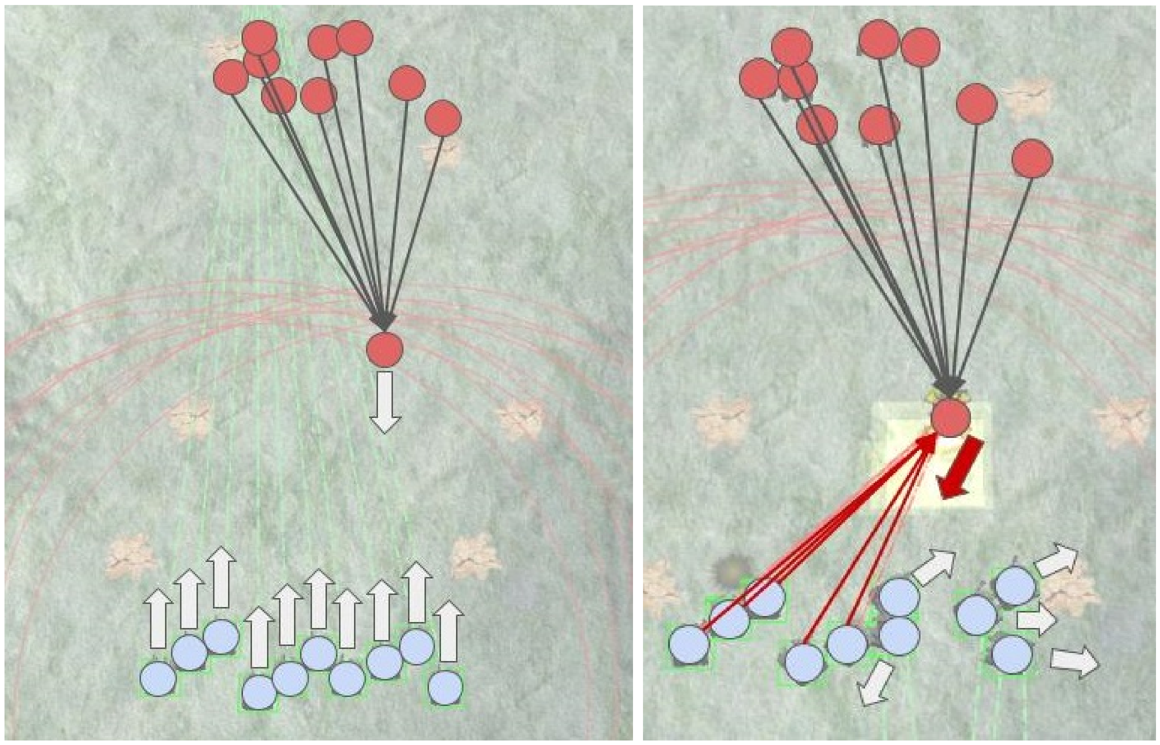
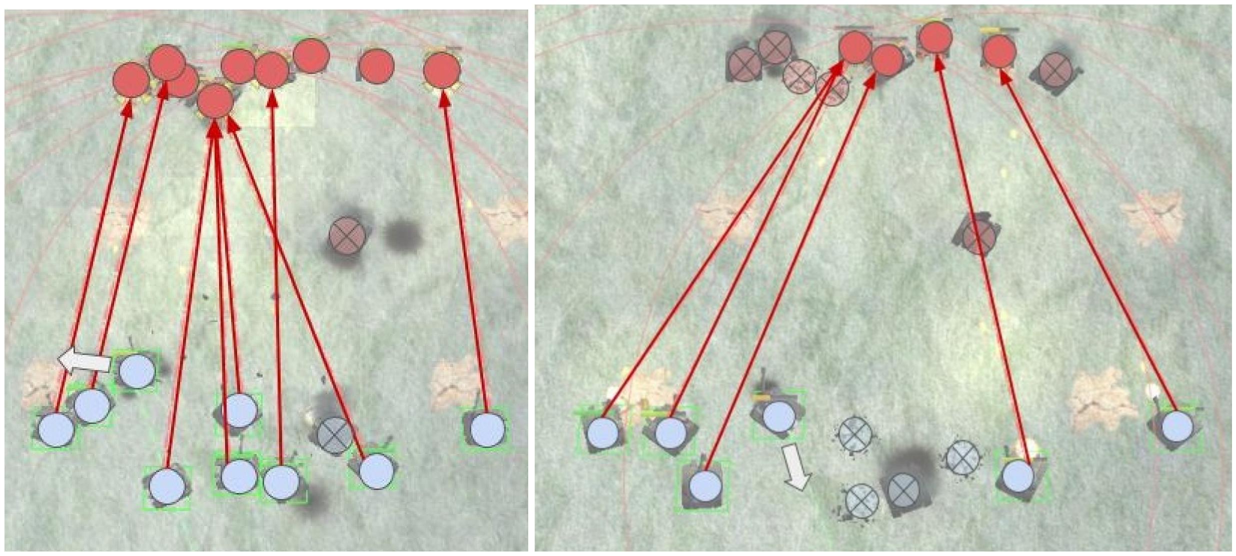


Figure 32. An example of the flow of a battle. Above left: Units first meeting at the start of battle. Above right: Positional MOEA is giving new movements to units that need to spread out. Below left: Since fewer new positions are needed here, all units are attacking. Below Right: Positional MOEA is victorious, as the remaining enemies are about to be destroyed



VI. Conclusion

6.1 Overall Conclusion of the Tactical Positional Algorithm

Overall, the idea and execution of the positioning algorithm was a success. Every objective presented in Chapter 1 was fulfilled effectively. Chapter 3 presents an effective mathematical representation of the problem domain, describing the problem space in a manner that is easily understandable as well as accurate.

Chapter 3 also discusses the implementation of offline testing. It confirmed the validity of the mathematical model. It also confirms that the designed algorithm for the problem space operates as intended. This gives confidence that there exists a solution that can be found as well as the effectiveness of an MOEA when applied to the problem.

Chapter 5 details the results of testing the tactical positional algorithm against various other tactical options. Despite the limitations of the Spring engine for fine-tune control of the units within the game (Chapter 5, Section 3), the Positional MOEA still vastly outperformed all other algorithms. The Hybrid MOEA took a hit to effectiveness of performance in battle for a tradeoff of computational performance. Both of these are very effective options for a tactical deployment.

A major advantage of utilizing these MOEAs is the flexibility of usage. There is no algorithmic training required nor is there a need to perform a tree search. This means that regardless of map or terrain, this positional algorithm can and will perform with the same functionality.

It should be noted that these two developed algorithms (positional and hybrid) are not an ultimate solution to the tactical problem. There are many different tactical options for the infinite number of scenarios. The results of this project should be utilized as a strong set of options for a mutable tactics pool in the future.

Concerning the tactical opportunities for the AFIT RTS AI agent, there are very few areas for improvement. A lack of complex tactical mechanics in the Spring engine, such as cover, along with the inability to fine-tune control of the units, means that there are adjustments required in the Spring engine. Implementing controls with more fidelity would improve the positional algorithm and potentially allow for even better results.

6.2 Future Work

There are variety of areas that can be improved upon for the AFIT RTS AI Agent. Several areas focusing in more "large scale" ideas would move this project to the next step

Scouting.

The biggest weakness of the AFIT RTS AI Agent is that it still "cheats" by knowing where the enemy is at all times by turning off Fog of War. In real world combat, however, there is no guarantee of such clarity of information. A good scouting method could gather information based off of the map in order to determine the enemy's base location as well as predicting their strategy. Dave Churchill has done similar work in this area with his UAlbertaBot [30], and would be a good starting point for research in this particular area.

Strategic Diversity.

Currently there is only one unit being built by the AFIT RTS AI Agent, the stumpy tank. This means there is only ONE option every time that's being employed. If an agent is going adapt to a player's skill level, there needs to be a wider variety of build orders available within the Balanced Annihilation game. This would involve a live, online identification of the enemy's strategy and adapting the agent's own strategy and perhaps even tactics on the fly. This would be an extension of the work performed by Blackford in implementing adjustments to the build order capability [5].

Potential New Platform.

The Spring engine is an open source option and has the implied flexibility of all open source programs. Conversation on the development front is relatively quiet, however. Continuing work may mean future students working on this project have difficulty getting support from the Spring engine's development team. There may be an engine in existence that the AFIT RTS AI Agent can be ported to in order to improve the ease of development. Any future platform that this work would be transitioned to must have a robust method of modifying an AI agent in order to implement a diverse system like the AFIT RTS AI Agent.

6.3 Final Remarks

The experiments performed in this research effort confirmed the hypothesis that the positional MOEA is an effective tactical alternative to the already existing tactical options. The complexity of the problem leaves much to be desired in terms of computational performance. Overall, this effort can contribute greatly to training in tactical situations by providing a diverse, unique movement strategy for each situation.

Appendix A. Results of Experiment 2: MOEA Effectiveness

This appendix contains the raw data of the second experiment: MOEA Effectiveness. Firstly, in table 11, it can be seen the wins and losses of each tactical option against each other version. The format for the table has each row being the *top* tactical option and each row is the *bottom* tactical option. Since an option cannot be paired up against itself, the cells that are the same matchup are listed as *N/A*. While each of the *top-bottom* matchups where run 15 times, ties are not recorded. Therefore, each cell that does not add up to 15 matches do so because of ties within the skirmish.

Each subsequent table in this appendix is the tabular form of each of the matchup experiments. These tables contain the remaining Hit Point(HP) values for each surviving unit at the end of the match. This means that most entries have a formatting where one combatant have a set of numbers and the other have 0. Entries relating to Targeting MOEA have a slight deviation from this style of output. This is due to an error where the Targeting MOEA's units would cause a client freeze whenever they attempted to engage the enemy commander. Since there are no new entries to the log, it still has the HP values of their units against the remaining units of their opposition. Majority of the time this resulted in a 5-1 variation which can be easily assumed to be the Targeting MOEA's victory.

Table 11. Win/Loss Results of Experiment 2

Bottom / Top	Default	Weak	Proximity	Target MOEA	Position MOEA	Hybrid MOEA
Default	N/A	10-5	7-8	7-8	1-14	15-0
Weak	3-12	N/A	1-14	0-15	0-15	0-15
Proximity	4-11	10-4	N/A	7-6	4-11	1-14
Target MOEA	7-7	10-4	11-5	N/A	2-13	1-14
Positional MOEA	13-2	15-0	13-2	13-2	N/A	10-5
Hybrid MOEA	13-2	15-0	9-5	13-1	5-8	N/A

Table 12. Default (top) vs Weak (bot) Raw Results

Default(Top)	Weak (Bot)
0	1862
0	1052,130
1662,1751	0
0	1179,1440,1548
1182	12,1211
757	0
1248,1142,1617,1626	0
1727	0
1433,1705,1343,1607,1508	0
1680,982,1277	0
0	1645,1732,1517
1458,743,1372,1803	0
1446	0
1767,1860	0
1384,1635,1393,1131	0

Table 13. Summary of Default (top) vs Weak (bot)

	Default (Top)	Weak (Bot)
Wins	10	5
Best Survival Rate	5 units	3 units
Worst Survival Rate	1 unit	1 unit
Average Survival Rate	2.8 units	2.2 units
Average Total HP	4023.9	1517
Average Individual HP	1437.1	1211.6

Table 14. Weak (Top) vs Default (Bot) raw results

Weak (Top)	Default (Bot)
0	1568,1615,1604,1744,1512
1676,422,1614	0
0	1633,1172,1789
0	1625,1709,1637,1794
0	1446,1433,1514,1679
964	0
0	1894,1661,1740
1116	1374,1555,1495,1406
0	1291,1777,1391
0	1723,1724,1043
0	1471,1540,1576,1601
0	1767,720,1209
0	1661,1652,1727
436	1849,1318
1660,1617,1596	0

Table 15. Weak (Top) vs Default (Bot) results summary

	Default (Bot)	Weak (Top)
Wins	12	3
Best Survival Rate	5 units	3 units
Worst Survival Rate	2 units	1 unit
Average Survival Rate	3.4 units	2.3 units
Average Total HP	5873.6	3183
Average Individual HP	1708.5	1364.1

Table 16. Default (Top) vs Proximity (Bot) raw results

Default (Top)	Proximity (Bot)
1568,1434	0
633,1310	0
1552,1481	0
0	1255,1278,1311
1437,1581,1028	0
1003,1310,1308,330	0
0	818,1444,868
0	1476
0	1350,1102,1665,1619,864
0	1108
0	656,1429,1750
1558	537,323,920,891
0	50,1613,1446,1544,480,500
1357,943,1540	0
1259,1416,964,1464,618	0

Table 17. Default (Top) vs Proximity (Bot) results summary

	Default (Top)	Proximity (Bot)
Wins	7	8
Best Survival Rate	5 units	6 units
Worst Survival Rate	2 units	1 unit
Average Survival Rate	3 units	3.25 units
Average Total HP	3,650.5	3537.125
Average Individual HP	1,216.8	1088.3

Table 18. **Proximity(Top) vs Default (Bot) raw results**

Proximity (Top)	Default (Bot)
1060,1307,835,901	0
0	1308,1183,639,511
1203	1261,1256
710,548,1184	0
0	991,151,1540
0	1151,1428,621
579,1051,590,1662	0
0	286,1202
0	1174,311,1413,1320,801
0	1044,1236,1387,1782,1535,989
0	864,99,1343
631,1021,40	0
0	374,307
0	1301,1753,1495,282,1598,838,832,1443
0	186,747,1810,1677

Table 19. Default (Bot) vs Proximity (Top) results summary

	Proximity (Top)	Default (Bot)
Wins	4	11
Best Survival Rate	4 units	8 units
Worst Survival Rate	3 units	2 unit
Average Survival Rate	3.5 units	3.8 units
Average Total HP	3,029.7	3,951.7
Average Individual HP	1,216.8	1034.9

Table 20. Default (Top) vs Targeting MOEA (Bot) raw results

Default (Top)	Target MOEA (Bot)
0	179,1659,1700,1483
262,586,99	0
1041	1738,1410,1440,1647,881
1698,1084,1148,361,139,428	0
895,985,1395	0
0	1696,1612,1073,528,1204,1108
0	455,966,1271
732,1025,20,1615,1429,920	1744
0	1728,1189
1077, 461	0
437	1182
69	710
536,820,1659,1045	1691
298	1328
653,427,456	0

Table 21. Default (Top) vs Targeting MOEA (Bot) results summary

	Default (Top)	Target MOEA (Bot)
Wins	7	8
Best Survival Rate	6 units	6 units
Worst Survival Rate	2 units	1 unit
Average Survival Rate	3.8 units	3 units
Average Total HP	3,075.2	3523.6
Average Individual HP	797.29	1225.6

Table 22. Target MOEA (Top) vs Default (Bot) raw results

Target MOEA (Top)	Default (Bot)
858,1702,777,1770	500
965,1224,156	943
0	44,1188
414	785,451
0	1695,828,499,1062
95,1396,1869	410
1782,1603	992
1145	1128,64,521,1048
444,299,1575	1341
1601,1738,1264,1610,1603	1123
1615,1528,1757	334
1670	902,1302,217,1105,414,1262
842	1389,514,498,264,1495
1255	1162
0	738,847,1378,1675

Table 23. Targeting MOEA (Top) vs Default (Bot) results summary

	Target MOEA (Top)	Default (Bot)
Wins*	7	7
Best Survival Rate	5 units	6 units
Worst Survival Rate	2 units	2 unit
Average Survival Rate	3.2 units	3.8 units
Average Total HP	4747.2	3330.4
Average Individual HP	1444.8	863.4

Table 24. **Default (Top) vs Positional MOEA (Bot) raw results**

Default (Top)	Positional MOEA (Bot)
0	1480,1610,1234,1103,1281,1535,1122,1400
0	1814,1573,1646,1629,1658
0	1446,1675,1365,1464,1753,1324
0	1472,1440,105,1654,330
1023,93	1755,997,1498,1753,1634,1537
0	791
31,455	1370
409	1771,1255,1784,1605,1703
0	1607,258,1426,502
0	1702,1721,1746,1288,1657,1708
0	1477,1711,1465,1772
0	1386,1500,454,1604,1439,1607
842	1573,1601,1438,1018,986
0	989,1729,1504,1612
0	1709,1569,1697,1620,1756,1573

Table 25. **Default (Top) vs Positional MOEA (Bot) results summary**

	Default (Top)	Positional MOEA (Bot)
Wins	1	14
Best Survival Rate	2 units	8 units
Worst Survival Rate	2 units	1 unit
Average Survival Rate	2 units	5 units
Average Total HP	486	6840.4
Average Individual HP	243	1348.8

Table 26. Positional MOEA (Top) vs Default (Bot) raw results

Positional MOEA (Top)	Default (Bot)
1290,1754,1691,1612,1607,1761	0
1239,1740,260,1730,1351	0
0	771,1192,1315
1460,1552,1664,1649	0
1825,1741,1764	0
0	209,903,325,999
1545	0
1573,157,1497,1543,1672	0
1668,1711,1256,1609,1625	0
1766,1703,715,1663,1452,1577	0
1410,1702,1408,1054,1243	0
1810,1195,1693,1527,1288,1684	0
1714,1713	0
997,1519,1632	0
1249,1688,1588,1769,1595	0

Table 27. Positional MOEA (Top) vs Default (Bot) results summary

	Positional MOEA (Top)	Default (Bot)
Wins	13	2
Best Survival Rate	6 units	4 units
Worst Survival Rate	1 units	3 unit
Average Survival Rate	4.3 units	3.5 units
Average Total HP	6458.4	2857
Average Individual HP	1499.2	816.28

Table 28. Default (Top) vs Hybrid MOEA (Bot) raw results

Default (Top)	Hybrid MOEA (Bot)
0	1606,446,1754,1672,1808
0	1588,1492,549,1367
0	1694,1526,1591,162,1244
0	779,1710,1639,1215,1621,1767
0	814,1483,1709,1633
0	1523,1784,1697,1719
0	1779,1719,577,1500
0	1615,1708,1617,1669,1666
0	1632,1832,11,1723
0	1657,1586,1682,1555
0	1728,1528,1605,1633,1500
0	1239,1634,1196,1548
0	1604,1559,1745,1686,1446
0	1519,1629,1812,1600
0	1302,1620,1783,1750

Table 29. Positional MOEA (Top) vs Default (Bot) results summary

	Default (Top)	Hybrid MOEA (Bot)
Wins	0	15
Best Survival Rate	0 units	6 units
Worst Survival Rate	0 units	4 unit
Average Survival Rate	0 units	4.4 units
Average Total HP	0	6,653.4
Average Individual HP	0	1,489.3

Table 30. Hybrid (Top) vs Default (Bot) raw results

Hybrid MOEA (top)	Default (Bot)
0	1463,1181,897,1350
1728,882,1712	0
257,1540,1398,1652	0
857,1706,1188,1426,1729	0
930,531,1335,1220	0
1622,1732,1670,1594,1623	0
1703,1696,1088,1481,1703	0
1647,1696,1612,1597,1277	0
1789,1814,1666	0
1719,603,1782,1703,1584	0
661,1895	0
837,1566,1033,1378,1663,1746,1744	0
1593,8,1672,1409,790,1458	0
1246	464,576,958,347
1558,1232,1506,1529,1321,1640	0

Table 31. Positional MOEA (Top) vs Default (Bot) results summary

	Hybrid MOEA (Top)	Default (Bot)
Wins	13	2
Best Survival Rate	7 units	4 units
Worst Survival Rate	2 units	4 unit
Average Survival Rate	4.6 units	4 units
Average Total HP	6,517.7	3,618
Average Individual HP	1412.1	904.5

Table 32. Weak (Top) vs Proximity (Bot) raw results

Weak (Top)	Proximity (Bot)
0	1583,788,1469,1474,1638,590,877
0	1440,1769,1650,1615,1459
90,1015,786	0
0	432
0	1671,1405,1913
0	1669,1156,1548,1736,1593,1600
0	1790,1570,1837
0	1642,1772
0	249,710,442,1646
0	96,347
0	1410,1678
0	1618,1650,1623,1609,1367,1608
0	1558,1688,1777
0	1557,1729,1438,689,1706
0	1746,1313,1661,1545

Table 33. Weak (Top) vs Proximity (Bot) results summary

	Weak (Top)	Proximity (Bot)
Wins	1	14
Best Survival Rate	3 units	7 units
Worst Survival Rate	3 units	1 unit
Average Survival Rate	3 units	4 units
Average Total HP	1891	5296.1
Average Individual HP	630.3	1373

Table 34. Proximity (Top) vs Weak (Bot) raw results

Proximity (Top)	Weak (Bot)
1601,1665,1732,1788	0
1529,1213,1035	0
0	683,452,946,350
1737,1450	0
1777,1192,1587,1071,1660	0
942,880,1176,1490,1573	0
0	1407,1761,1763,1558,1770
1769,1539,37,1790,1472	0
1752	0
691,1766,1630,1052,1311	0
682,1316	0
1756	0
0	859,1007
0	903,1474
469	139

Table 35. Proximity (Top) vs Weak (Bot) results summary

	Proximity (Top)	Weak (Bot)
Wins	10	4
Best Survival Rate	5 units	5 units
Worst Survival Rate	1 units	2 unit
Average Survival Rate	3.3 units	3.25 units
Average Total HP	4566.1	3,733.2
Average Individual HP	1383.6	1,148.6

Table 36. Weak (Top) vs Target MOEA (Bot) raw results

Weak (Top)	Target MOEA (Bot)
21	1714,1716,1705,1612
1103	1818,1411,1125,892,1554
841	1696,1660,1522,1626,1196
17	1635,1713,1778,1477
*4	1584,1462,1580,1155,1539
1193	1641,1736
79	641,1718,1113,1541,870,1786
254	1693
593	1716,1513,1617,1709
270	1393,1709,1415,951
306	1572,694
54	1769,1627,1612,1715,1630
479	1260,721,1494
294	1768,1722,1747
32	1679,1502,849,1299,609,1430,1328

Table 37. Weak (Top) vs Target MOEA (Bot) results summary

	Weak (Top)	Targeting MOEA (Bot)
Wins	0	15
Best Survival Rate	0 units	7 units
Worst Survival Rate	0 units	1 unit
Average Survival Rate	0 units	3.8 units
Average Total HP	0	5,863.9
Average Individual HP	0	1,465.9

Table 38. **Target MOEA (Top) vs Weak (Bot) raw results**

Target MOEA (Top)	Weak (Bot)
1627,1876,984,909	464
0	1604,1419
1115,1582,1617,1306	68
0	353,786,1560
1659,1478,1330	985
1431,1329,716,588	1365
1122	958
1662,1773,1657,1064	1573
1063	529
1577,957,1636,1657,1484,1727	882
0	793
1161,1012,1840	1262
1402,1840	1043
1198,1106	586
1646,10,443	1239
641	73,564,1653,823,1108

Table 39. **Target MOEA (Top) vs Weak (Bot) results summary**

	Target MOEA (Top)	Weak (Bot)
Wins	10	4
Best Survival Rate	6 units	5 units
Worst Survival Rate	2 units	1 unit
Average Survival Rate	3.3 units	2.7 units
Average Total HP	4,639.9	2,684
Average Individual HP	1325.6	976

Table 40. Weak (Top) vs Positional MOEA (Bot) raw results

Weak (Top)	Positional MOEA (Bot)
0	1711,1750,1664,1766,1594
0	1395,1818,1627,1714,1481,1692
0	1714,1400,1464
0	1786,1696,1794,1634
0	1638,1521,1626
0	1543,1841,1477
0	1674,1538,1397,1516,1610
0	1555,1639,1577,1542,1141,1468
0	1501,1683,1643,1596
0	1608,1590,1556,1690
0	1866,1666
0	1555,1636,1484,196,1109
0	950,1534,1768
0	1744,1848,1641
0	1569,1536,1760,1623

Table 41. Weak (Top) vs Positional MOEA (Bot) results summary

	Weak (Top)	Positional MOEA (Bot)
Wins	0	15
Best Survival Rate	0	6
Worst Survival Rate	0	2
Average Survival Rate	0	4
Average Total HP	0	6223
Average Individual HP	0	1555.7

Table 42. **Positional MOEA (Top) vs Weak (Bot) raw results**

Positional MOEA (Top)	Weak (Bot)
1691,1697,1695,1604,1828	0
1647,1721,1505,1716	0
1686,1512,1629,1465	0
1771,1707,1697,1664	0
1699,1785,1693,1767	0
1524,1739,1700,1618,1246	0
1676,1559,1417,1492,1699	0
1494,1524,1594,1584	0
1442,1763,1601,1197	0
1753,1305,1443,1600,1639,1563	0
361,926,1568,1695,1422,1687	0
1599,1622,1361,1570,1108	0
941,1536,1095,1508	0
1204,1371,1607,1614,1129	0
1626,1614,1598,1696	0

Table 43. **Positional MOEA (Top) vs Weak (Bot) results summary**

	Positional MOEA (Top)	Weak (Bot)
Wins	15	0
Best Survival Rate	6	0
Worst Survival Rate	4	0
Average Survival Rate	4.6	0
Average Total HP	7,053.9	0
Average Individual HP	1,533.4	0

Table 44. **Weak (Top) vs Hybrid MOEA (Bot) raw results**

Weak (Top)	Hybrid MOEA (Bot)
0	1475,1543,1614,1609,939,1695
0	1686,1774,1769,1316,1710
0	1682,1657,1593,1369,1599
0	1003,1619,1759,1638
0	1682,1622,1727,1628
0	1834,1714,1392,934,1642
0	1708,862,1595,1497
0	1758,1712,1641,1615,1636,1599
0	1600,1521,1728
0	1302,1590,1591,1633,1410,1625,1644
0	1493,1572,1763,1594,1590
0	1616,1646,1205,1428
0	1659,1640,1753,1714
0	1562,1606,1522,1749,1582,1680
0	1628,1769,1469,1846

Table 45. Weak (Top) vs Hybrid MOEA (Bot) results summary

	Weak (Top)	Hybrid MOEA (Bot)
Wins	0	15
Best Survival Rate	0	7
Worst Survival Rate	0	3
Average Survival Rate	0	4.6
Average Total HP	0	7,578.7
Average Individual HP	0	1,647.5

Table 46. Hybrid MOEA (Top) vs Weak (Bot) raw results

Hybrid MOEA (Top)	Weak (Bot)
1824,1765,1641,1627,1681,1621	0
1718,1559,1659,1741,1540	0
1497,1570,806,1385	0
639,1692,1543,967,1614	0
1608,1564,1734,1531,1102,915,1598	0
1667,1484,1712,866,1478,1666	0
1766,1632,1752,1654	0
1683,1574,1605,1399,1598,1622,1661,1700	0
1576,1692,1530,1707,1607,1636	0
1280,1786,1728,1410	0
853,1657,1715,1560,1804	0
1698,1359,1624,1584,907,1618	0
1725,1752,1801	0
1695,1366,1611,1539,1758	0
1680,1698,1299,1660,1536	0

Table 47. Hybrid MOEA (Top) vs Weak (Bot) results summary

	Hybrid MOEA (Top)	Weak (Bot)
Wins	15	0
Best Survival Rate	8	0
Worst Survival Rate	3	0
Average Survival Rate	5.2	0
Average Total HP	8,138.7	0
Average Individual HP	1,565.1	0

Table 48. Proximity (Top) vs Targeting MOEA (Bot) raw results

Proximity (Top)	Targeting MOEA (Bot)
463,694,465,999,441,961,298	0
1807	0
69,558	0
756,55	1357,1055,527,1428,1530,1467
8	455,1396,68,248,797
1326,1000,1471,1006,1331	0
1590	1388
1420	1651
1655	983,1009
463	699,796
479,989,1716,743,338,899,1249,1188	0
1524	0
52	1624,1723,959
1386,58,588,1170	0
175	1106
6	716,1393,1349

Table 49. Proximity (Top) vs Targeting MOEA (Bot) results summary

	Proximity (Top)	Targeting MOEA (Bot)
Wins*	7	6
Best Survival Rate	8	6
Worst Survival Rate	1	2
Average Survival Rate	4	3.5
Average Total HP	3,602.2	3,596.5
Average Individual HP	900.5	980.8

Table 50. **Targeting MOEA (Top) vs Proximity (Bot) raw results**

Targeting MOEA (Top)	Proximity (Bot)
903,569,622,160	0
1607,1500,1569,1796	0
468	1468,1330
260,1050,1097,23,714,514,1051	0
979,1437,1255,1477,1304	0
325	0
823	185,1825,754,369
782,168	1645,1841,1663
66,703,902,808,364,1074	0
211,423	1290,1022,754
1536,947	0
1235,586,726,1387	0
665,321,1123,577,269,705,978	0
955	1504,1351
1602,1209,914,538,518,1425,1450	583
1011,381,445,102	0

Table 51. **Targeting MOEA (Top) vs Proximity (Bot) results summary**

	Targeting MOEA (Top)	Proximity (Bot)
Wins	11	5
Best Survival Rate	7	6
Worst Survival Rate	1	2
Average Survival Rate	4.6	3.5
Average Total HP	4,076.2	3,596.5
Average Individual HP	886.1	980.8

Table 52. Proximity (Top) vs Positional MOEA (Bot) raw results

Proximity (Top)	Positional MOEA (Bot)
0	912,475,1509,911,1745,1488
939,16,975	0
0	39,1572,1432,1589,886,1469,1603
0	1589,1643,1572,1601,1587,1106,1794
0	1669,1751
23,332,362	0
519	1816
919,400,590,912	730
0	657,745,946,728,1170,980
0	1607,1151,1551,217,1316
0	1686,1701
0	1757,1560,1831,255
0	299,1883,1175
0	1317,763,1809
0	1670,1236,1318,636
316,514,1088,871	0

Table 53. Proximity (Top) vs Positional MOEA (Bot) results summary

	Proximity (Top)	Positional MOEA (Bot)
Wins*	4	11
Best Survival Rate	4	7
Worst Survival Rate	3	2
Average Survival Rate	3.5	4.4
Average Total HP	2,064.2	5,499.1
Average Individual HP	589.7	1234.5

Table 54. Proximity MOEA (Top) vs Proximity (Bot) raw results

Positional MOEA (Top)	Proximity (Bot)
0	445,440,127,867,91,611
113,1568,1325,1157	0
1600,948,1373	0
1189,225	0
1744,536	0
1734,141,1834	0
0	434,589
554	0
1783,1627,1582,910	0
677,1753,1585,1688	0
1351,1761,1682,1222,1328,1036	0
1026,1108,1619,1678,927	0
1037,1272,1443,695,873	0
1688,1739	0
1652,1393,1607,1755	0

Table 55. Positional MOEA (Top) vs Proximity (Bot) results summary

	Positional MOEA (Top)	Proximity (Bot)
Wins	13	2
Best Survival Rate	6	6
Worst Survival Rate	1	2
Average Survival Rate	3.4	4
Average Total HP	5,230.7	1,827.5
Average Individual HP	1,278.6	456.8

Table 56. **Proximity (Top) vs Hybrid MOEA (Bot) raw results**

Proximity (Top)	Hybrid MOEA (Bot)
0	1589,1548,950,617,510,1198
0	1720,755
429	1724
0	1047,876
0	1433,1645,766,1069,1606
0	661,1511,1671,1616,1264,569,1586
0	1544,791,1240
92	1857
0	1383,1125,863,946,1052
0	750,1623,573,982
0	1007,1174,862,1348,917,1235
0	1056,507,1118,1219
0	1286,1098,1527,1562,1631,1567
582,855	0
0	1268,1512,1622,1240,1769

Table 57. **Proximity (Top) vs Hybrid MOEA (Bot) results summary**

	Positional MOEA (Top)	Proximity (Bot)
Wins	1	14
Best Survival Rate	2	7
Worst Survival Rate	2	1
Average Survival Rate	2	4
Average Total HP	1,438	4,939
Average Individual HP	719	1,213.1

Table 58. Hybrid MOEA (Top) vs Proximity (Bot) raw results

Hybrid MOEA (Top)	Proximity (Bot)
469,1661,540,1783	0
1757,275,1625,1556,608,429,1640	0
0	521,736,793
1799	294,472
1842	0
0	644,328,1265
298,1581,590,1019	0
0	1545,957,1453,1574,1309,1349,1092
1367,1067,529,228,14,1421	0
1715,1582,320	0
224,1667,1609,142,605,973	0
559,1788,358,304,750	0
0	1548,767,330
1724,1576,1637,348,1690	0
0	554,519

Table 59. Hybrid MOEA (Top) vs Proximity (Bot) results summary

	Hybrid MOEA (Top)	Proximity (Bot)
Wins	9	5
Best Survival Rate	7	7
Worst Survival Rate	1	2
Average Survival Rate	3.4	4
Average Total HP	4,651.8	3,610
Average Individual HP	1,021.1	902.5

Table 60. **Targeting MOEA (Top) vs Positioning MOEA (Bot) raw results**

Targeting MOEA (Top)	Positional MOEA (Bot)
0	775,947,391,617
0	1722,1035,753,1463,1546
0	587,757,1240,1000
0	904,1541,566
363,369,638	978
0	694,1046,368,546,476
0	529,504,1522,1870,1607,251
0	395,1734,1200,768,42,1527
0	1452,1207,986
0	794,1294,759,1223
0	696,1601,1593,1092,1084,1698
1135,78,478,268	1509
0	110,1743,1194,622
0	903,1584,699,1306
0	1569

Table 61. **Targeting MOEA (Top) vs Positional MOEA (Bot) results summary**

	Targeting MOEA (Top)	Positional MOEA (Bot)
Wins	2	13
Best Survival Rate	4	6
Worst Survival Rate	3	1
Average Survival Rate	3.5	4.2
Average Total HP	1,673.5	4,320.7
Average Individual HP	478.1	1,021.2

Table 62. Positioning MOEA (Top) vs Targeting MOEA (Bot) raw results

Positional MOEA (Top)	Targeting MOEA (Bot)
1714	0
1888,566	0
1777,1383,571,722	0
979,756,1203	0
1040,1636	0
722,1607,1693,1616	0
722,1014,752,1201,551,1480	0
1703,923,1615,951,1394	0
725,1117,1568,1079,437,1598,528,711	0
59	832,71,1153,1031,462,759
1295,231,1474,1166,726,28	0
56	287,267,430,77,325
1050,1162,808	0
501,1305,1069,1023	0
1194,84,1097,1403,1363,930	0

Table 63. Positional MOEA (Top) vs Targeting MOEA (Bot) results summary

	Positional MOEA (Top)	Targeting MOEA (Bot)
Wins	13	2
Best Survival Rate	8	6
Worst Survival Rate	1	4
Average Survival Rate	3.5	5
Average Total HP	4,883.7	4,320.7
Average Individual HP	1,154.3	1,021.2

Table 64. Targeting MOEA (Top) vs Hybrid MOEA (Bot) raw results

Targeting MOEA (Top)	Hybrid MOEA (Bot)
0	1134,1370,343,640,1464,945,1646,1529,1501
0	650,348,1594
0	1593,1452,7,1273,1394,1650
0	1538,1757,445,147,1704,1178
1640,797,5,300	1596
0	1583,1071,750,718,545
0	1609,1172,1448,1137,1525,970,536
0	1500,682,621,570,1621
0	1607,926,387
0	552,1382,981,1336,1214
0	1050,942,848,343,568
0	356,477,1762,1191,432
0	276,1176,528,1603,1764,294,1239
0	1491,599,570,1579,1424,1230,1022,1195,89
0	1240,1158,995,180,996,1548,1346

Table 65. Targeting MOEA (Top) vs Hybrid MOEA (Bot) results summary

	Targeting MOEA (Top)	Hybrid MOEA (Bot)
Wins	1	14
Best Survival Rate	4	9
Worst Survival Rate	4	3
Average Survival Rate	4	5.8
Average Total HP	2,742	6,089.7
Average Individual HP	685.5	1,039.7

Table 66. Hybrid MOEA (Top) vs Targeting MOEA (Bot) raw results

Hybrid MOEA (Top)	Targeting MOEA (Bot)
1499,1766,720,1770,1586	0
1153,1562,629,1356,1195,1056,1258,1423	0
1181,350,1318,340	0
1475,238,1171,816	0
564,1141,1687	0
1367,1370,1076	0
1328,88,1269,705	0
165,714,1354,1431,1381	0
1806,1531,1543,512,1345,620	0
442,1778,1193,923,1412	0
445,655,1205,873,1670	0
1355,1549,963	0
888,808,60,1690,1701,1612	0
82	543,1131,1251,749,949,1341

Table 67. Hybrid MOEA (Top) vs Targeting MOEA (Bot) results summary

	Hybrid MOEA (Top)	Targeting MOEA (Bot)
Wins	13	1
Best Survival Rate	8	6
Worst Survival Rate	3	6
Average Survival Rate	4.6	6
Average Total HP	5,246.8	5,964
Average Individual HP	1,118.1	994

Table 68. **Positioning MOEA (Top) vs Hybrid MOEA (Bot) raw results**

Positioning MOEA (Top)	Hybrid MOEA (Bot)
586,803,29,905	0
1004,338,595,845,1029	0
0	1415,448,689,611
839,1291,1195,62,934,1173	0
1213,1101,1322,1516,1079,904,775	0
846	0
612,809,795,1135,1299,530	0
0	1379,430,423
0	601,193
1489,289,1236,1251,1034,1452,997	0
1143,261,831,1409,601,1263,382,468	0
0	1096,399,920,98
569,1147,690,341,575,487,481,1045	0
276,448,938,638,253,1399	0
0	642,207,382,763

Table 69. **Positional MOEA (Top) vs Hybrid MOEA (Bot) results summary**

	Positional MOEA (Top)	Hybrid MOEA (Bot)
Wins	10	5
Best Survival Rate	7	4
Worst Survival Rate	1	2
Average Survival Rate	5.8	3.4
Average Total HP	4,895.7	2,139.2
Average Individual HP	844.0	629.1

Table 70. Hybrid MOEA (Top) vs Positional MOEA (Bot) raw results

Hybrid MOEA (Top)	Positional MOEA (Bot)
0	444,1233,1235,942,1235,1303
526	87,881
483,138,1538,149,531	0
0	1405,1307,739
355,30,675,215	0
0	887,483,856
0	1081,445,604,778
550,1145,1407	0
0	707,1115
373,699,1398	0
0	1410,8,1198,1345,501
1012,382,1047,564,328	0
0	1323,1190
-	-

Table 71. Hybrid MOEA (Top) vs Positional MOEA (Bot) results summary

	Hybrid MOEA (Top)	Positional MOEA (Bot)
Wins*	5	8
Best Survival Rate	5	6
Worst Survival Rate	3	2
Average Survival Rate	4	3.3
Average Total HP	2,603.7	3,092.7
Average Individual HP	650.9	916.3

Bibliography

1. John R. Boyd, "The essence of winning and losing," `http://www.danford.net/boyd.essence.htm`, Accessed: March 10, 2015.

2. "Dune II - wikipedia, the free encyclopedia," `http://en.wikipedia.org/wiki/Dune_II`, Accessed: March 10, 2015.

3. Cavedog Entertainment, "Total annihilation," `https://en.wikipedia.org/wiki/Total_Annihilation`, Accessed: April 24, 2016.

4. "Sparcraft image from github," `https://github.com/davechurchill/ualbertabot/wiki`, Accessed: April 24, 2016.

5. Jason M Blackford, "Online build-order optimization for real-time strategy agents using multi-objective evolutionary algorithms," M.S. thesis, Air Force Institute of Technology, 2014.

6. Donald A Gruber, "Tactical ai in real time strategy games," M.S. thesis, Air Force Institute of Technology, 2015.

7. "Boss image from github," `https://github.com/davechurchill/ualbertabot/wiki`, Accessed: April 24, 2016.

8. Craig Reynolds, "Boids," `http://www.red3d.com/cwr/boids/`, Accessed: April 24, 2016.

9. Lyall J Di Trapani, "A real-time strategy agent framework and strategy classifier for computer generated forces," M.S. thesis, Air Force Institution of Technology, 2012.

10. Carlos Coello Coello, Gary B Lamont, and David A Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems, 2nd edition*, Springer Science & Business Media, 2007.

11. "Real-time strategy - wikipedia, the free encyclopedia," `http://en.wikipedia.org/wiki/Real-time_strategy`, Accessed: March 10, 2015.

12. Malcolm Gladwell, *Blink: The power of thinking without thinking*, Hachette Digital, Inc., 2007.

13. "Fog of war - wikipedia, the free encyclopedia," `http://en.wikipedia.org/wiki/Fog_of_war`, Accessed: March 10, 2015.

14. Blizzard Entertainment, "Blizzard entertainment: Classic games," `http://us.blizzard.com/en-us/games/legacy`, Accessed: March 10, 2015.

15. Blizzard Entertainment, "Starcraft," `http://us.blizzard.com/en-us/games/sc/`, Accessed: April 24, 2016.

16. Blizzard Entertainment, "Starcraft ii world championship series," 2016.

17. Relic Entertainment, "Company of heroes," `http://www.companyofheroes.com/`, Accessed: April 24, 2016.

18. "Spring rts engine," `http://springrts.com`, Accessed: March 10, 2015.

19. "Balanced annihilation - spring," `https://springrts.com/wiki/Balanced_Annihilation`, Accessed: May 01, 2016.

20. "Welcome to the air university," `http://www.au.af.mil/au/soc/sos.asp`, Accessed: March 10, 2015.

21. "Wargus — home," `www.wargus.sourceforge.net`, Accessed: March 10, 2015.

22. “Sparcraft - starcraft combat simulation,” `https://code.google.com/p/sparcraft/`, Accessed: March 10, 2015.

23. Carl Von Clausewitz, *On war*, Digireads.com Publishing, 2004.

24. U.S. Army, *Army Field Manual FM (3-0) Unified Land Operations*, 2012.

25. Guillaume Chaslot, *Monte-carlo tree search*, Ph.D. thesis, Maastricht University, 2010.

26. U.S. Army, *Field Manual 3-21.8 (FM 7-8) The Infantry Rifle Platoon and Squad*, 2007.

27. U.S. Army, *Field Manual 3-21.8 (FM 7-8) The Infantry Rifle Platoon and Squad*, 2007.

28. Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss, “A survey of real-time strategy game AI research and competition in starcraft,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 4, pp. 293–311, 2013.

29. Kurt Weissgerber, “Developing an effective and efficient real time strategy agent for use as a computer generated force,” M.S. thesis, Air Force Institute of Technology, 2010.

30. David Churchill, ,” `http://webdocs.cs.ualberta.ca/~cdavid/rts_research`.

31. “Starcraft AIb competition,” `http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/`, Accessed: March 10, 2015.

32. Buro Churchill, “Portfolio greedy search and simulation for large-scale combat in starcraft,” *CIG 2013*, 2013.

33. Buro Churchill, “Incorporating search algorithms into rts game agents,” *AIIDE*, 2011.

34. Spronck Heijden, Bakkes, “Dynamic formations in real-time strategy games,” *IEEE*, 2008.

35. Bjorn Gmeiner, Gerald Donnert, and Harald Kostler, “Optimizing opening strategies in a real-time strategy game by a multi-objective genetic algorithm,” *Research and Development in Inteligent Systems, XXIX*, 2012.

36. “Multi-objective optimization - wikipedia, the free encyclopedia,” `http://en.wikipedia.org/wiki/Multi-objective_optimization`, Accessed: March 10, 2015.

37. “Welcome to PyGMO,” `http://esa.github.io/pygmo/`, Accessed: March 10, 2015.

38. “MOEA framework, a java library for multi-objective evolutionary algorithms,” `http://www.moeaframework.org/`, Accessed: March 10, 2-15.

39. “ParadisEO, paradiseo home page,” `http://paradiseo.gforge.inria.fr/`, Accessed: March 10, 2015.

40. “ParadisEO, paradiseo documentation,” `http://eodev.sourceforge.net/`, Accessed:April 20, 2016.

41. “jMetal web site,” `http://jmetal.sourceforge.net/`, Accessed: March 10, 2015.

42. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

43. Eckart Zitzler, Marco Laumanns, Lothar Thiele, Eckart Zitzler, Eckart Zitzler, Lothar Thiele, and Lothar Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," 2001.

44. Yang Liu, "A fast and elitist multi-objective particle swarm algorithm: NSPSO," in *Granular Computing, 2008. GrC 2008. IEEE International Conference on*. IEEE, 2008, pp. 470–475.

45. R. Kennedy, J.; Eberhart, "Partical swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*. IEEE, 1995, pp. 1942–1948.

46. "Spring engine maps," `https://springrts.com/wiki/Maps`, Accessed: May 11, 2016.

47. "Balanced annihilation v7.60 - armstump," `http://imolarpg.dyndns.org/modinfo/ba760/armstump.html`, Accessed: March 10, 2015.

48. "What is the rationale behind the magic number 30 in statistics?," `https://www.researchgate.net/post/What_is_the_rationale_behind_the_magic_number_30_in_statistics`, Accessed: May 11, 2016.

49. Lpez-Ibez Paquete Vahrenhold Beume, Fonseca, "On the complexity of computing the hypervolume indicator," *IEEE Transactions on Evolutionary Computation*, 2009.

50. Corne Knowels, "On metrics for comparing nondominated sets," .

51. Schwarz-Bernt Middendorf Moritz, Reich, "Refined ranking relations for selection of solutions in multi objective metaheuristics," .

52. David H Wolpert and William G Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

53. "Ecma-404 the json data interchange standard," `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`, Accessed: May 11, 2016.

54. "15.3. time time access and conversions," `https://docs.python.org/2/library/time.html`, Accessed: April 26, 2016.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 16-06-2016	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2014 — June 2016

4. TITLE AND SUBTITLE A Multi-Objective Approach to Tactical Manuvering Within Real Time Strategy Games	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Ball, Christopher D., Capt, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-MS-16-J-004
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A:
Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States

14. ABSTRACT

The real time strategy (RTS) environment is a strong platform for simulating complex tactical problems. The overall research goal is to develop artificial intelligence (AI) RTS planning agents for military critical decision making education. This particular research effort of RTS AI development focuses on constructing a unique approach for tactical unit positioning within an RTS environment. By utilizing multiobjective evolutionary algorithms (MOEAs) for finding an "optimal" positioning solution, an AI agent can quickly determine an effective unit positioning solution with a fast, rapid response. The resulting agent does not requires the usage of training or tree searches to optimize, allowing for consist effective performance across all scenarios against a variety of opposing tactical options.

15. SUBJECT TERMS

RTS, Tactics, MOEA, Optimization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. G. B. Lamont, AFIT/ENG
U	U	U	UU	140	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636, x4718; gary.lamont@afit.edu

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18