



**SCOUTING IN REAL-TIME STRATEGY
GAMES: THEORY, METHODS, AND
IMPLEMENTATION**

THESIS

Rosemberg Ortiz, Capt, USAF
AFIT-ENG-MS-17-M-056

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-17-M-056

SCOUTING IN REAL-TIME STRATEGY GAMES:
THEORY, METHODS, AND IMPLEMENTATION

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

Rosemberg Ortiz, B.S.C.S., M.A.

Capt, USAF

March 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-17-M-056

SCOUTING IN REAL-TIME STRATEGY GAMES:
THEORY, METHODS, AND IMPLEMENTATION

THESIS

Rosemberg Ortiz, B.S.C.S., M.A.
Capt, USAF

Committee Membership:

Dr. G. Lamont
Chair

Dr. B. Borghetti
Member

Dr. L. Merkle
Member

Abstract

The real time strategy (RTS) game environment is a strong platform for simulating complex strategical and tactical problems. The overall goal of this research is to develop artificial intelligence (AI) RTS planning platform for military critical decision-making education. The agent should have the ability to perform at an expert level as well as to challenge and assess a players critical decision-making ability or skill-level. The nature of time sensitivity within the RTS environment creates very complex situations. Each situation must be analyzed, and commands must be given to each tactical unit before the scenario on the battlefield changes and makes the decisions less relevant. This particular investigation effort of Air Force Institute of Technology (AFIT) RTS AI platform, focuses on constructing a novel approach to scouting within the RTS environment using multi-objective evolutionary algorithms (MOEAs) for finding a “best” scouting solution.

To create and evaluate this scouting unit, three distinctive phases occurred. First is the development of a mathematical model describing the problem space of the scout unit within a RTS combat scenario. This model allows for the second phase to occur, a successful offline MOEA implementation that solves the scouting problem by providing a set of good solutions. Finally, the third phase focuses on experimenting with the Spring RTS Engine scout units with scripted commands in order to determine the best performing unit when placed against various tactical scenarios. The selected scout unit is then integrated with the MOEA solution to direct the unit to the “best” path found. Experimental results validated the first two phases. The third phase resulted in allowing the selection of the best performing scout unit and partial integration with the MOEA into the AFIT RTS AI platform.

Acknowledgements

I would like to express my sincere appreciation to my faculty advisor, Professor Gary Lamont, for his guidance, patience, and support throughout the course of this thesis effort. His insight and experience is truly appreciated. I would, also, like to thank my wife for her love and encouragement in my weakest moments.

Rosemberg Ortiz

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	ix
List of Tables	xii
I. Introduction	1
1.1 Real Time Strategy (RTS) Games	1
1.2 Decision Making in RTS	2
1.3 Scouting in RTS	2
1.4 Multi-Objective Evolutionary Algorithms	3
1.5 Research Goal	4
1.6 Research Objectives	4
1.7 Research Approach	5
1.8 Thesis Organization	5
II. Background	7
2.1 Real Time Strategy Games	7
Popular RTS Games	9
RTS Development Platforms	13
2.2 RTS Artificial Intelligence Research	16
Challenges	17
Techniques	17
2.3 Previous AFIT Agent Developments	18
Adaptive Response - Weissgerber	18
Strategy Optimization - DiTrapani	19
Build Order Optimization - Blackford	19
Tactics Optimization - Gruber	19
Tactical Maneuvering - Ball	20
2.4 Decision Making Applied to RTS AI	20
Strategic Decision Making	20
Tactical Decision Making	23
RTS AI Agent Problem Solving	25
2.5 Related Work on Scouting in RTS	26
Scouting Methods	26
Scouting Approaches	27
2.6 Multi-Objective Evolutionary Algorithms	30
MOEA Software Packages	31

	Page
Description of MOEA Algorithms	32
2.7 Chapter Summary	35
III. Methodology for the Scouting Problem	36
3.1 High-level Design: Defining the Problem Domain	37
Orienteering Problem	37
Multi-Objective Orienteering Problem	38
Multi-Objective Scout Orienteering Problem	38
3.2 MOEA Design Domain	43
3.3 MOEA Implementation	44
MOEA Software	44
Selecting an MOEA	45
3.4 Integration into the AFIT RTS AI Platform	46
AFIT RTS AI Platform	46
Scout Integration	47
3.5 Chapter Summary	51
IV. Design of Experiments and Testing of Scouting	52
4.1 Introduction	52
4.2 Experiment Equipment	52
4.3 Finding the Best MOEA	53
Testing MOEAs	53
Metrics for Analysis	54
4.4 Integrating the Selected MOEA into the AFIT RTS AI Platform	54
Maneuvering a Scout Unit	55
Testing the Scout Units	55
4.5 Chapter Summary	58
V. Analysis of Results	59
5.1 Introduction	59
5.2 Results from MOEA Evaluation	59
Analyzing the Hypervolume Metric	59
Analyzing the Spacing Metric	60
Pareto Front Structure	61
Analyzing the Execution Time	62
Overall Analysis of MOEA Testing	64
5.3 Results from Testing the Scouting Units	65
Evaluating the Scenarios	66
Summary of the Scenarios	68
Implementation of MOSOP in AFIT RTS AI platform	68
5.4 Chapter Summary	68

	Page
VI. Conclusion and Final Remarks	70
6.1 Conclusion	70
6.2 Future Work	71
6.3 Final Remarks	72
Appendix A. Results of MOEA Experiments	73
1.1 MOEA Experiment Results	73
Bibliography	80

List of Figures

Figure		Page
1	Real-Time Strategy Tree [1]	3
2	Screenshots of Partially Observable Map in StarCraft (left); Fully Observable Map in Balanced Annihilation (right)	8
3	A screenshot of Dune II [2]	10
4	A screenshot of WarCraft (left) and StarCraft (right)	12
5	A screenshot of Total Annihilation [3]	13
6	Spring Engine - Balanced Annihilation Screen Shot	14
7	RTS Strategic Planning Tree [1]	23
8	RTS Tactical Planning Tree for Targeting [4]	24
9	Scouting Procedure (adapted from [5])	28
10	HAMMER incorporated into a partially-observable game [6]	29
11	Example Pareto Front with Different Population Sizes (5, 10, 20, 50)	30
12	Example Graph of Multi-Objective Orienteering Problem with five nodes	39
13	Example of Decision Space (Genotype) and Objective Space (Phenotype) - adapted from [7]	42
14	An example of solving a problem through <i>PyGMO</i> [8]	45
15	AFIT Multi-Scale Agent Architecture [1]	47
16	The testing map “The Pass”	48
17	Fast Scout Kbot - Flea (left); Light Scout Vehicle - Jeffy (right)	49
18	Example of Scout Finding the Enemy’s Base	57

Figure		Page
19	Example of Scout Avoiding Contact	57
20	Example of Scout Outside LOS of enemy	58
21	MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 5	60
22	MOEAs Spacing Comparison Sample - Pop. 20 Gen. 5	61
23	NSGA-II Sample Pareto Front - Pop. 20 Gen. 4	62
24	SPEA2 Sample Pareto Front - Pop. 20 Gen. 4	63
25	NSPSO Sample Pareto Front - Pop. 20 Gen. 4	63
26	MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 5	64
27	MOEAs Execution Time Comparison Sample - Pop. 100 Gen. 5	65
28	Survivability Example of Flea Scout	67
29	MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 2	74
30	MOEAs Spacing Comparison Sample - Pop. 20 Gen. 2	74
31	MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 2	75
32	MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 3	75
33	MOEAs Spacing Comparison Sample - Pop. 20 Gen. 3	76
34	MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 3	76
35	MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 4	77
36	MOEAs Spacing Comparison Sample - Pop. 20 Gen. 4	77
37	MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 4	78

Figure		Page
38	MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 5	78
39	MOEAs Spacing Comparison Sample - Pop. 20 Gen. 5	79
40	MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 5	79

List of Tables

Table	Page
1 Scouting Methods Survey	27
2 Characteristics comparison of Jeffy and Flea	49
3 Overview of hardware and software used in the experiment	52
4 Hypervolume results - Pop. 20 Gen. 5	60
5 Spacing results - Pop. 20 Gen. 5	61
6 Execution time results - Pop. 20 Gen. 5	64
7 Summary of Scripted Scenarios - default vs. scout	67
8 Summary of Scripted Scenarios - moea vs. scout	68
9 Summary of Scripted Scenarios - moea2 vs. scout	68

SCOUTING IN REAL-TIME STRATEGY GAMES: THEORY, METHODS, AND IMPLEMENTATION

I. Introduction

This investigation effort advances the development of the Air Force Institute of Technology (AFIT) Real Time Strategy (RTS) Artificial Intelligence (AI) platform [1, 8, 4, 9, 10] by exploring theories, methods and implementing a scouting unit. Particularly, this work continues the use of Multi-objective Evolutionary Algorithms (MOEAs) [1, 8, 4] to find the path a scouting agent takes to gather, or reveal, critical map information, thus improving decision making. This study makes use of RTS games because they are an excellent platform that are used by many AI researchers. It offers a versatile framework to create and evaluate AI techniques for highly complexed environments [11, 12]. The following sections expand on some of these themes and present the research goal, objectives and approach.

1.1 Real Time Strategy (RTS) Games

Real-time strategy (RTS) games are a genre of computer and video console games that provide an abstraction of real-world military warfare, where the player takes the role of a commander and manages forces and resources, while simultaneously plans defenses and attacks in real-time [13]. RTS games are considered strategic simulations of battlefield scenarios. These simulations vary in complexity and can mimic the complexity of real-life environments with simpler dynamics in a finite and smaller world. RTS games are complex enough to study key and interesting problems such as decision making under uncertainty or real-time adversarial planning [11, 14].

Adding a scout unit to the AFIT RTS AI agent ensures the commander can explore the environment in a timely manner, collect adequate information for decision making, and adjust the strategy to implement in order to defeat the enemy as quickly as possible.

1.2 Decision Making in RTS

Decision making is integral to military operations and artificial intelligence. In both, decision-making can be regarded as a problem-solving activity terminated by a solution deemed to be satisfactory, or a process that involves the analysis of a finite set of alternatives described in terms of evaluative criteria [15]. The U.S. military uses techniques such as the Observe, Orient, Decide, and Act (OODA) model and the Military Decision Making Process (MDMP).

In AI and RTS there is the Markov decision process, Bayesian classifications, and multi-objective optimality, to name a few [11, 16, 1, 8, 4]. The scouting unit in this research aims at greatly contributing to decision making by revealing valuable information of the environment.

1.3 Scouting in RTS

Army Field Manual 793 defines scouting as the act of collecting timely information about the enemy’s order of battle while avoiding contact [17]. It is a form of reconnaissance used to explore outside an area occupied by friendly forces to gain information about natural features and enemy presence. In RTS, a scout is a unit (usually a worker) that is sent to explore the map, thus revealing information of the environment hidden by the fog-of-war (FoW). This concept attempts to resemble Carl von Clausewitz definition of FoW, which is uncertainty in situational awareness experienced by participants in military operations [18]. In RTS, for each player, whether

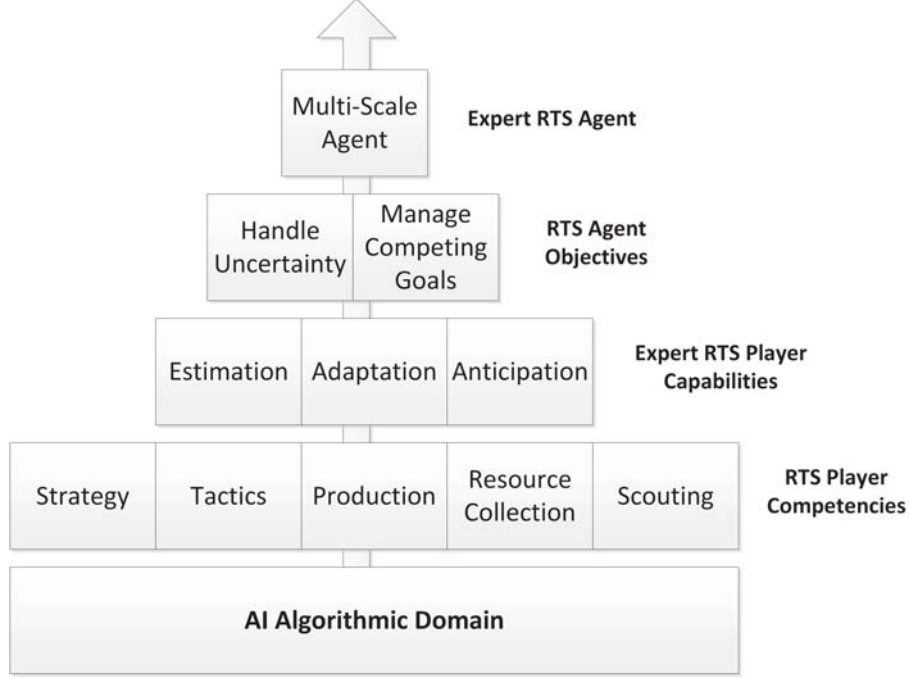


Figure 1. Real-Time Strategy Tree [1]

human or AI, FoW means only a certain range of view of the map is visible, mainly where the units stands [19]. Therefore, without sending reconnaissance units to the opponent's territory to gather information, a has difficulty coming up with the right tactical plan to conquer or counteract the opponent [20]. Blackford [1] suggests that scouting is among several RTS player competencies that are key to winning a RTS game to include strategy, tactics, production or build order, and resource collection. Figure 1 illustrates these competencies, plus the capabilities and objectives a RTS player should have in order to reach an expert RTS playing level [1] .

1.4 Multi-Objective Evolutionary Algorithms

Multi-Objective Evolutionary Algorithms (MOEAs) are methods used in the area of multiple-criteria decision making for the purpose of solving problems with multiple objectives. This is done by analyzing the potential results with regard to several competing objectives [21]. As opposed to single objective problems, where the focus

is on maximizing a single aspect of strategic importance, through either an equation or metric, a MOEA balances multiple such aspects, which often times involve tradeoffs, in order to make a decision. RTS games are an excellent area for the study of MOEAs due to the many different objectives that can be evaluated in its large complex space.

1.5 Research Goal

The goal of this investigation is to study different scouting theories and methods used in RTS research, as well as develop and implement a new approach to scouting for the AFIT RTS AI.

1.6 Research Objectives

In order to achieve the previously stated goal, this investigation is organized to follow these specific objectives:

1. Design a multi-objective mathematical model representation and algorithmic solution to execute scouting in a RTS environment. This allows the evaluation of various scenarios and weight potential algorithmic solutions against one another.
2. Evaluate the offline performance of Multi-objective Evolutionary Algorithm within the scouting problem domain. This promotes the selection and employment of the most effective algorithm to the online evaluation in RTS.
3. Evaluate the online performance of two scripted scout units against other scripted attack and defense agents found within the AFIT RTS AI platform. Additionally, evaluate the selected MOEA from the scouting problem domain within the AFIT RTS AI platform. This permits the validation of the scouting algorithm, thus ensuring it can find the best path and reduce the possibility being killed.

1.7 Research Approach

This research builds upon previous work conducted by Blackford [1], Gruber [4], and Ball [8] which showed that integrating MOEA techniques into the Spring Engine platform greatly improve the performance of the AFIT RTS AI platform in both, strategic (build order) and tactical (targeting) decision making. At the strategic level, Blackford optimized the planning tool utilized to select the best build-order (strategy) to defeat the enemy. At the tactical level, Gruber and Ball optimized the targeting choices of the units under the agent’s control and the position of the units as they are maneuvered across the environment, respectively [4, 8]. This research expands the tactical realm by adding scout unit functionality.

This research effort begins by defining the scouting domain through the design of a mathematical model and understanding the search space, thus building effective evaluation techniques. An offline simulation is developed to ensure that the algorithm is functioning properly with various MOEA alternatives. Once the scouting problem has been defined and an algorithm has been designed, it is integrated with the scouting units in the AFIT RTS AI platform. Consequently, a test of the algorithm is performed to provide the desired functionality of the scout unit against various other scripted agents, to include Gruber’s previously developed targeting agent [4] and Ball’s positional agent [8]. Thus ensuring a proper application in a RTS AI combat situation.

1.8 Thesis Organization

This thesis is organized as follows: Chapter II provides a background into Real Time Strategy games, decision making, artificial intelligence, an overview of RTS Tactics and Multi Objective optimization, and the state of the art on scouting. Chapter III covers the development of the scout orienteering 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 platform.

II. Background

In order to advance in the integration of a scouting unit into the AFIT RTS framework, it is imperative to cover a brief review of related concepts and previous works. This chapter provides a background on RTS games and related AI research efforts. It follows with an overview of the different levels of RTS AI and general strategies. It provides an understanding of decision making techniques and how they are used at different levels within the RTS environment. Additionally, it offers an overview of previous AFIT research efforts and a discussion of scouting RTS AI research techniques and approaches. It concludes with an overview of multi-objective optimization problems, some of the algorithms used, and the frameworks available.

2.1 Real Time Strategy Games

Real-time strategy (RTS) games are a genre of computer and video console games that provide an abstraction of real-world military warfare. In RTS games, the player takes the role of a commander and manages forces and resources, while simultaneously plans defenses and attacks in real-time [13]. RTS games are considered strategic simulations of battlefield scenarios. These simulations vary in complexity and can mimic the complexity of real-life environments with simpler dynamics in a finite and smaller world. RTS games are complex enough to study key and challenging problems such as decision making under uncertainty or real-time adversarial planning [11, 12, 14].

An RTS game typically starts with two or more players on different locations on a map at a main base and with a few workers. From there, the players must build their base of operations to acquire new units, then expand and train army units to destroy the enemy and enemy bases. RTS games are different from traditional



Figure 2. Screenshots of Partially Observable Map in StarCraft (left); Fully Observable Map in Balanced Annihilation (right)

strategy games such as Chess (which is turn-based), since opposing players can make moves simultaneously. In RTS games such as StarCraft, a player has very small amount of time to decide the next move, typically a 42 milliseconds (typical refresh rate), before the game changes state as opposed to Chess, where the player can take near 60 seconds before he/she makes a move [11]. Furthermore, the complexity of RTS games is very large in both state space size and in terms of number of actions available at each decision cycle (state space of Chess is estimated to be around 10^{50} , where in StarCraft is estimated to 10^{1685} , depending on the size of the map) [11].

Most RTS games are partially observable, meaning players can only see the part of the map that surrounds each unit - also known as *fog-of-war* (FoW). Other games, like in the Spring Engine game Balanced Annihilation, the map is fully observable, which means players are aware of the opponent's location at all times. The left image in Figure 2 shows an example of FoW in the bottom left corner of the image where only a portion of the map is revealed, as opposed to the right image that illustrates a fully observable map. Scouting, the act of collecting timely information about the enemy's order of battle while avoiding contact, is an essential tactic in RTS game-play as it allows for map discovery [11, 17]. For each player, whether human or AI,

only a certain range of view of the map is visible, mainly where the units stands. Therefore, without sending reconnaissance units to the opponent's territory to gather information, a player builds without information about the enemy and has difficulty coming up with the right tactical plan to conquer or counteract the opponent [20].

Popular RTS Games.

When most people think of strategy games, they usually reference turn-based board games such as Chess, The Battle for Normandy, and Nato Division Commanders, to name a few. However, back in the 1980's a subgenre, where the game does not progress incrementally in turns, was evolved as real-time strategy. Although not very popular, one of the first titles to mark this genre was Herzog Zwei, published in 1989 for the Sega Mega Drive/Genesis gaming system [22]. Nevertheless, Herzog Zwei was very influential in the making of Dune II, released on DOS in 1992 and regarded as a landmark in the development of RTS games. Several other ground-breaking titles, such as the Warcraft and StarCraft series, and Total Annihilation appeared in the 1990s, which firmly recognized the RTS genre. Today, RTS games are fundamental in PC/video gaming. A more comprehensive RTS game history is available in [13].

Dune II.

Dune II: The Building of a Dynasty, was released in 1992 and stands as the prototype of RTS since it combined the concepts of its predecessors into a solid package with concepts and mechanics still used today. Themed after Frank Herbert's *Dune* science fiction novel series, it was the first game that provided the resource gathering concept for building new units as well as the base construction philosophy, as opposed to Herzog Zwei where it was just a matter of conquering bases [2, 22]. Dune II, Figure 3, also introduced the concept of technology tree, a series of dependencies the player



Figure 3. A screenshot of Dune II [2]

is required to satisfy to access more powerful units and buildings, with a trade-off of requiring more investment to reach those items, a concept that enhances skills in decision making; nevertheless, the idea of scouting had not been adapted within this game. Additionally, Dune II, although with certain flaws, contributed to AI advancement by allowing researchers to evaluate different machine learning techniques, and is recognized as a starting point for future RTS AI agents [2].

WarCraft and StarCraft.

WarCraft and *StarCraft*, Figure 4, are two of the most popular series in the RTS genre. Developed by Blizzard Entertainment, these two games are use strongly in AI research. *WarCraft*, released in 1994, was the first of the series and provided an “out-of-this-world,” high-fantasy conflict between Humans and Orcs (a race from planet Draenor). Sequels of this series basically added other universes and races such as the Night Elves and the Undead to play with. This series held true to the RTS tenants

and contributed with an additional resource capability, the *supply* concept. Through this concept a player builds an infrastructure to support their armies and increase their technology. 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 [23].

StarCraft, debuted in 1998, was originally intended to be another version of *WarCraft*, but in space. Nevertheless, it evolved into a life of its own as it received an unprecedented high amount of popularity and acclaim [24]. *StarCraft* balances three factions against one another: the Terrans, adaptable and mobile human-like race; the Protoss, an ancient and powerful enigmatic alien race that's fewer in numbers; and the Zerg, an ever-evolving insectoid 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. Scouting, although not prominent, became a tactical technique within the original *StarCraft* game and a focus of researcher in the area of imperfect information and uncertainty [20, 19, 5, 25]. Overall, *StarCraft* is still a pillar of RTS even today, and with its sequel *StarCraft II* frequently seen at professional level video game tournaments [26].



Figure 4. A screenshot of WarCraft (left) and StarCraft (right)

Total Annihilation.

Total Annihilation, Figure 5, released in 1997 by Cavedog Entertainment, provides a large-scale variation of the RTS concept introducing 3D units and terrain. *Total Annihilation* is set in the midst of a futuristic galactic war, where battles take place on the surface of planets and moons. Similarly to previously discussed titles, the efforts of the player are centered on constructing a defensive base and an offensive force, and conquering opponents. However, what made this game unique, as compared to WarCraft or StarCraft, is that resources are limitless and are accumulated at a constant rate rather than in small batches. 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 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]. This research uses a modification of this title called *Balanced Annihilation*, created for the open source RTS engine, *Spring*.



Figure 5. A screenshot of Total Annihilation [3]

RTS Development Platforms.

RTS games provide an excellent player interface to a real-time simulation environment, but AI agent development within the genre has not always been favorable. Most games lack an effective AI, since many are mainly scripted with complete information of the environment, therefore, players often feel there is a disadvantage and consider their computer opponents as cheaters. Researchers have come together and identified challenges and techniques to develop a much smoother agent, thus creating simulation software and platforms for various RTS games in different programming languages [11, 16]. The following are just a few, to include *Spring RTS Engine* which is the platform for the AFIT RTS AI.



Figure 6. Spring Engine - Balanced Annihilation Screen Shot

Spring RTS Engine.

Spring RTS Engine, also referred to as the Spring Engine or Spring, is an open source RTS environment that was originally intended for gameplay of Total Annihilation in three dimensions. However, this engine has evolved to support many other games with more modern and flexible features and is highly used for research. There are many options that Spring can provide, the most useful of which are visualization, unit customization and the open-source interface, which allows a large number of skirmish (small battle, not fully integrated) AIs to be used for offline play and simulation [27].

One of this strong components in this engine is its visualization, since it allows for a 3-dimensional representation of the simulation in real-time. This, paired with the customization, permits many historical or real-world scenarios to be created, from the units involved to the terrain featured. By combining these components, Spring

animations can be created to demonstrate the battle as it happens, as opposed to using numeric representations, thus enhancing feedback of the scenario received as it is happening.

Balanced Annihilation, Figure 6, is an open source version, also referred to as modification or mod, of *Total Annihilation* for the Spring Engine [28]. This is the modification used in this research effort because it allows full adaptation of the AI and evolving it to the overall research goal. Other games within the Spring Engine community tend to avoid AI implementation in favor of focusing on Player vs Player combat [28].

Stratagus.

Stratagus is another free, cross-platform, game engine used to build RTS games such as *Wargus*. *Wargus* is an open-source modification of *WarCraft II* and is used extensively for its full suite of environment manipulation, many of which are unavailable in the actual *WarCraft II* engine. *Wargus* behaves as an extension to the *WarCraft II* environment, and requires many features available from the *WarCraft II* installation [29].

SparCraft.

SparCraft is also an open source environment is used to simulate the *StarCraft* game engine, and typically used in performance analysis of AI agents built for *StarCraft*. Moreover, *SparCraft* is use to replicate damage, armor, hitpoints and conduct research, but cannot account for collisions or area effect damage [30].

2.2 RTS Artificial Intelligence Research

Artificial Intelligence (AI) is concerned with the study and design of rational agents [31]. It is often categorized into two main disciplines; search and knowledge representation. RTS games offer a large variety of fundamental AI research challenges that are not present in other game genres studied by the AI community [11, 16]. This section provides a background of those challenges and some of the techniques used in RTS AI research thus far. Most of the challenges and techniques are studied using StarCraft as an environment, but it also applies to other environments such as Warus, and in this case, Balanced Annihilation on SpringRTS.

Many RTS games can be played in single player mode using built-in RTS Artificial Intelligence (AI) agents. The agent takes the place of a human opponent. It performs the same basic tasks of a human player. These tasks often include building a base, constructing units, and attacking the player. The purpose of commercial RTS AI is to provide the human player a challenging, entertaining opponent for off-line play or simple practice. Unfortunately, many commercial RTS strategy games have very poor AI and developers choose not to implement modern AI techniques for various reasons [11, 16]. Some of the reasons being:

- Perception that modern AI algorithms are too slow to implement in a real-time program.
- Developers do not know about the existence of AI techniques.
- Developers do not understand how to implement AI techniques.
- Harbored distrust of modern AI techniques over traditional simple video game AI such as state machines and scripted actions.

Additionally, AI in RTS games is usually seen as a low priority within developers. Most of the simulation logic must be implemented before the AI agents can be fully tested, which can be extensive and time consuming. Furthermore, commercial gamer attention is mostly focused on the multi-player game-play and graphics, hence, that is where the money goes [32]. Because AI is a low priority, commercial RTS AI developers most often employ a scripted AI or include cheating, such as perfect information or resource bonuses, in their agents [16]. Which is why there was a call in 2004 for further research in applying modern AI techniques to the domain of RTS games [16].

Challenges.

Early research in RTS AI identified seven challenges [33]: Adversarial real-time planning, decision making under uncertainty, opponent modeling and learning, spatial and temporal reasoning, resource management, collaboration, and pathfinding (see [33] for definitions). Recent research has identified several additional research challenges such as how to exploit the massive amounts of existing domain knowledge (i.e. build orders, strategies, replays, etc) [11]. Ontaon et al. [11] grouped these challenges in six main different areas: Planning, Learning, Uncertainty, Spatial and Temporal Reasoning, Domain Knowledge Exploitation, and Task Decomposition. All of these challenges pertain to the use of scouting in RTS, but goes beyond the scope of this investigation; for a general application see [11].

Techniques.

The RTS environment allows for researchers to apply a gamut of AI techniques. Some of the prevalent techniques are reinforcement learning, dynamic scripting, finite state machines, case-based planning, Monte Carlo planning, Bayesian networks, and evolutionary algorithms to name a few [9, 11]. DiTrapani describes these techniques

as they pertain to the overall research in [9]. Ontanon et al. describe these and other techniques as they apply to macro- and micro-management in RTS in their survey [11].

2.3 Previous AFIT Agent Developments

This investigation is an extension of the AFIT RTS Agent, which has been in development since 2010. The foundation of this development comes from the efforts of Weissgerber’s [10, 34] agent development in 2010 and DiTrapani’s [9] RTS framework and strategy in 2012. This effort, particularly extends the tactical optimization area with theories and methods for a scouting agent. Some of the most recent development comes from Blackford’s [1, 35] work in *strategy optimization* in 2013, and Gruber’s [4] and Ball’s [8] work in *tactics optimization* in 2015 and 2016, respectively. The overall goal with the AFIT RTS AI platform is to build and improve a customizable RTS AI agent that can be used as a means to train military members in strategic and tactical decision making. A brief background on each of these efforts is discussed in the next few sections.

Adaptive Response - Weissgerber.

Weissgerber’s agent, developed in 2010, reacts to current situations in an RTS game through analysis and action taken on a subset of strategic behaviors capable of encompassing the current state of the game. This results in an agent able to outperform scripted agents by analyzing the opponent’s previous performance and developing an active counter strategy. This resulting counter strategy was shown to be 100% effective against the scripted agent that was developed to be beaten by choosing decision paths that led to a winning condition through optimization of the analyzed features [10, 34].

Strategy Optimization - DiTrapani.

As a continuation of Weissgerber’s work, Di Trapani’s agent identifies an incoming wave of enemies and develops a strategy to optimize and counter a situation [9]. In his work, Di Trapani determines the advantages and disadvantages of 8 different strategies tested against each other. These strategies are identified as Infantry rush, Blitz, Bomber, Expansion, Tank Rush, Defend Artillery, Anti-Air and Turtle. The goal of each test was to identify counter strategies for in each situation. The results allowed Di Trapani to utilized various classifiers in order to predict the enemy’s strategy and select a counter strategy to employ [9]. This is important since findings from the scout can influence decisions at this level.

Build Order Optimization - Blackford.

Blackford’s project advances the work of DiTrapani by creating a method that optimizes the strategic decision making conducted in the early stages of a game. Particularly, the agent focuses on Build Order Optimization (“BOO”) through the use of a Multi Objective Evolutionary Algorithm (MOEA). Blackford was the first to work with MOEAs within the AFIT RTS AI platform, thus being able to optimize the build-order (or production) for a given faction in the game. Blackford created a strategic planning tool that combines a game simulator and an MOEA to produce a set of diverse and good build-orders for an RTS agent, which were then integrated into the AFIT RTS AI platform [1, 35]. This resulted in the MOEA out-manufacturing each of the AI agents for which it was tested against.

Tactics Optimization - Gruber.

Gruber’s efforts improves the combat effectiveness of the units that are built by Blackford’s agent. Gruber also makes use of MOEAs to optimize tactical decision

making, allowing an AI agent to make fast decision, using the “optimal” solutions created by the algorithm, to target units controlled by the opposing force. Gruber’s addition to the AFIT RTS AI platform demonstrated to be highly effective against other scripted options such as “attack closest” and “attack weakest” with the optimization outperforming these scripted options at 70% and 100%, respectively [4].

Tactical Maneuvering - Ball.

Ball’s thesis, the most recent addition, continues the exploration of tactical optimization by taking the created units, maximizing their value, and controlling their positioning [8]. Through the use of MOEA, Ball was able to effectively evaluate the tactical position algorithm with no required algorithmic training or need to perform tree search. Therefore, it added the capability to run this algorithm regardless of the map or terrain [8].

2.4 Decision Making Applied to RTS AI

Decision-making is regarded as a cognitive process to a problem-solving activity that terminates by a solution deemed to be satisfactory [15]. The U.S. military has three levels of decision making - strategic, operational, and tactical [36]. RTS gameplay uses two - strategic and tactical levels [11, 16]. This section provides a quick description of these two and how they apply to this RTS research effort.

Strategic Decision Making.

Decision making at the strategic level is that cognitive process that develops, and actions on, the long-term goals and objectives that influence an organization or unit [15, 36]. At the heart of any organization is the leader - Chief Executive, President, or Commander. This person is in charge of making the final decision. However, he or

she is not always the one that comes up with the full strategy to run and manage the organization's resources, they lean on a group of high-level managers or executives in their cabinet. In the U.S. military, or warfare as a whole, strategy is use to accomplish the goals of the military leader or country [36]. The United States Army Field Manual of Military Operations provides a series of 9 essential concepts [36]:

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 [36]. This is a similar approach taken in RTS gameplay [11, 16].

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 [1]. 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 7 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 takes behavioral or optimization planning approach. 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 [1].

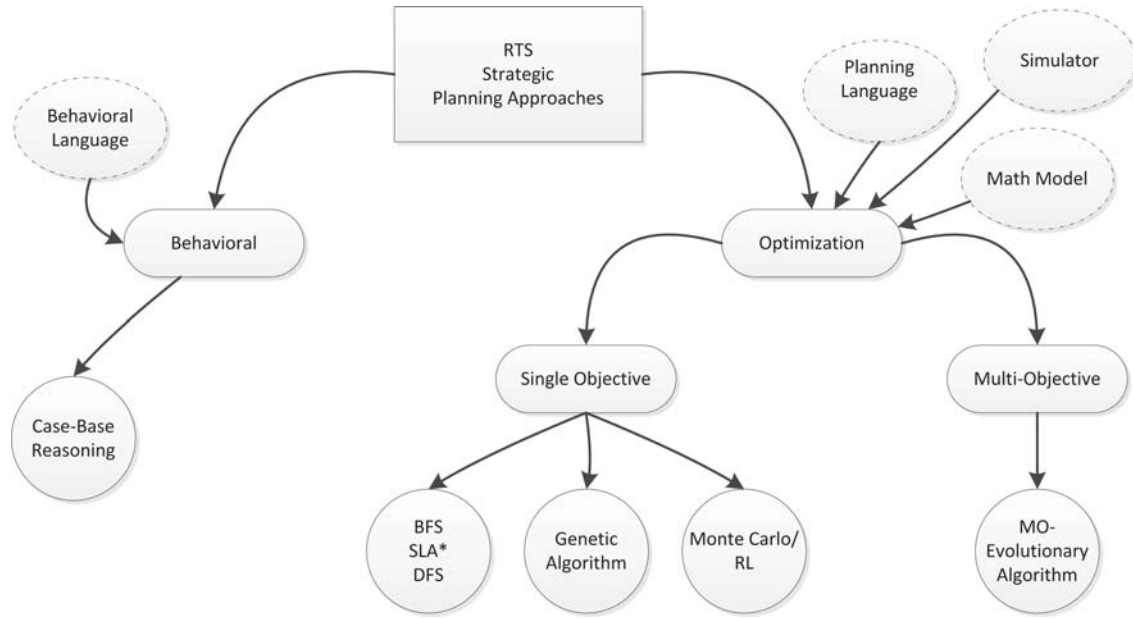


Figure 7. RTS Strategic Planning Tree [1]

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.

Tactics in RTS Games.

Within an RTS game, this level is concerned with making decisions at the group or squad level. Actions such as choosing what direction for a group to attack the enemy base from, selecting a target for a group to attack, and arranging a squad formation are examples of tactical-level AI. This level is usually focused on combat and is the one used for scouting. 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

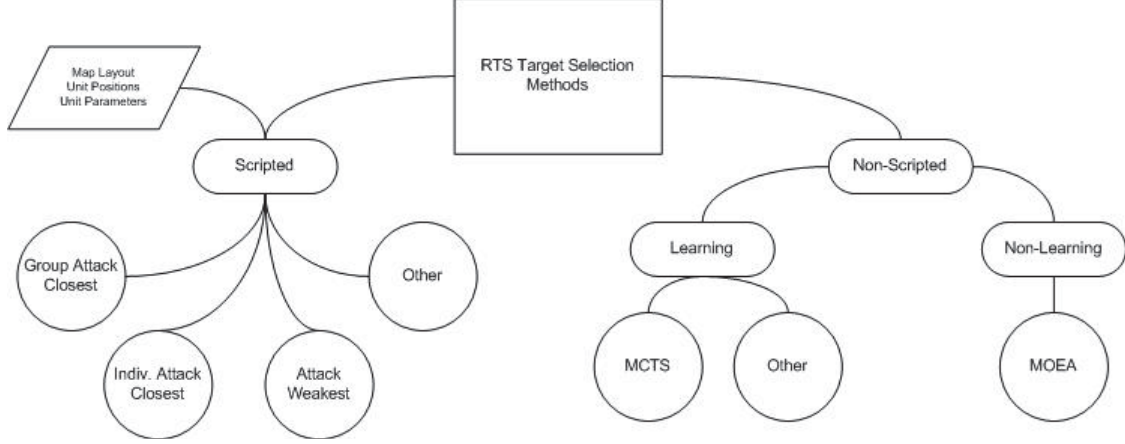


Figure 8. RTS Tactical Planning Tree for Targeting [4]

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 8 shows a 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 an entire force attacking the closest enemy, or each individual unit attacking their closest enemy are run at this level. Additionally, there is the ability to run a tactic 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, and can be exploited, given knowledge of the script being executed.

On the other side of the decision tree, there exist various tactical options to compensate for the simplicity of scripted methods. Learning methods can be implemented through the use techniques such as the Monte Carlo Tree Search (MCTS) to analyze, a priori, the state of the battle and make an appropriate decision [37]. 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 is MOEAs, which provide the ability to create a tactical decision based off of the optimized information to attempt to outperform a human player [4].

RTS AI Agent Problem Solving.

RTS games have several interesting problems for an agent to solve, many of which have large, rugged search spaces [16]:

Economy and Resource Management: An agent must make decisions about what units and buildings to construct. It must also plan a base layout and carefully select its building locations to give it a strategic advantage. The construction of resource-collecting structures, such as solar panels or metal extractors must be planned ahead of time so that the appropriate resources are available when needed.

Combat: The player must assign units to attack groups. If there are n units and m attack groups, then the player has $O(m + n)$ choices. Also, when an attack group engages an enemy's attack group, the player must assign units to attack opponent units.

Path finding: Whenever a unit is moved, a path must be computed. This computation must take into account static obstacles (buildings, mountains, crevices, rivers, etc.) and dynamic obstacles (other units). Optimally solving a single instance of this problem without any time constraint is not difficult. However, an RTS game can have several hundred units in play. Solving hundreds of dynamic path-finding problems in real-time becomes a difficult problem. Furthermore, just avoiding obstacles is not enough. An agent should also generate path trajectories that avoid unwanted enemy confrontations while placing the agents

attack group in an optimal attack position to strike its target.

Imperfect/partial Information: Most RTS games have imperfect or partial information, meaning players or AI units can only see the part of the map that surrounds each unit - also known as *fog-of-war*. Typically, an agent does not know what lies ahead unless it has a unit scouting the area. This effort focus on the functionality of a scout in a perfect information environment.

2.5 Related Work on Scouting in RTS

Army Field Manual 793 defines scouting as the act of collecting timely information about the enemy’s order of battle while avoiding contact [17]. It is a form of reconnaissance used to explore outside an area occupied by friendly forces to gain information about natural features and enemy presence. In RTS, a scout is a unit (usually a worker in StarCraft) that is sent to explore the map, thus revealing information of the environment hidden by the fog-of-war. For each player, whether human or AI, only a certain range of view of the map is visible, mainly where the units stands [19]. Therefore, without sending reconnaissance units to the opponent’s territory to gather information, a player builds in the blind and has difficulty coming up with the right tactical plan to conquer or counteract the opponent [20].

Scouting Methods.

Several works provide a list of strategies that can be applied to scouting in RTS [5, 25, 38, 19, 17]. Table 1 is a survey of the state-of-the-art in scouting, and suggest that overall, there are three main functions a scout needs to be able to do: (1) find the opponent’s base as soon as possible, thus allowing for a quick revealing of opponent’s expansion; (2) collect information about the opponent’s technologies and units; and

Table 1. Scouting Methods Survey

Reference	RTS Game	AI Bot	Approach	Methods / Strategies
Army Field Manual 7-93	N/A	N/A	Military Tactics	(1) Avoid contact with enemy forces and local population (2) Obtain timely information (3) Restrict mobility in area of operations (4) Understand the enemy's order of battle
Park et al. (2012)	StarCraft	Xelnaga	Machine Learning using WEKA	(1) Find the opponent base (2) Survive as long as possible (3) Collect opponent's structures/units information (4) Extract important information from the observed raw data (5) Generate a prediction model from the information.
K. Nguyen et al. (2013)	StarCraft	IceBot	Potential Field Pathfinding	(1) Find the enemy's main base (2) Move randomly to enemy base and gather information (3) Avoid being killed
Si et al. (2014)	StarCraft WarCraft	N/A	Hierarchical Multi-criterion Decision-Making (MCDM) Next Best View (NBV) Pathfinding	(1) Perceives the surrounding environment (2) Gathers terrain data (3) Avoids damage
Weber et al. (2011)	StarCraft	EISBot	Particle-based Prediction	(1) Track location of enemy and predict potential trajectory
LiquidPedia.com	StarCraft 2	N/A	Micro-management	(1) Uncover enemy gameplay decisions (2) Reveal the expansions of the opponent (3) Learn the location of the enemy units (4) Facilitate the prediction of technology choices (5) Assist gameplay decisions

(3) avoid being killed or damaged. These three functions are in line with what most expert players suggest in actual gameplay [39].

Scouting Approaches.

Table 1 also presents the different approaches that are being used for scouting in the different RTS frameworks. Park et al. [25], for example, use machine learning to classify the data collected by their bot, Xelnaga, via replays and/or direct plays with human players to predict the opponent's strategy. Nguyen et al. [5] use potential fields, a technique they adopted from Hagelback and Johanson, and pathfinding as approaches to move around the map without being detected while collecting information [38]. They suggest that it is imperative to maintain appropriate distance in order to capture key information, and that staying too far from the opponent's territory can result in a scouting failure as the agent would not be able to collect any information. Figure 9 (adapted from [5]) shows the scouting procedure a unit follows for decision

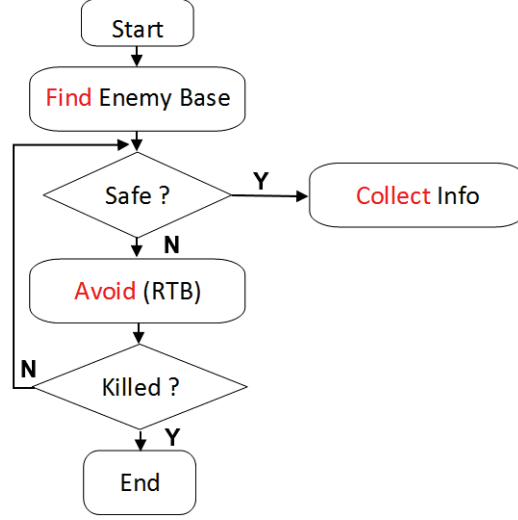


Figure 9. Scouting Procedure (adapted from [5])

making: Start by moving to enemy base, if the area is safe, collect information, if not safe (enemy present), avoid by returning to base (RTB); if unit is killed, end the procedure, otherwise continue scouting.

Si et al. propose an AI algorithm that focuses on scouting spatial environment, primarily collecting map information and recognizing features thus optimizing travel distance using the next-best-view (NBV) scouting strategy framework [19]. Within the framework, they also propose a hybrid map representation method, as well as a hierarchical candidate-position-identification mechanism using multi-criterion-based candidate-view evaluation strategy. The hierarchical-frontier candidate identification strategy filters potential positions before evaluating them, focusing more in grid and segment gathering since it allows for reconstruction of the outline of the map territory [19].

Another approach found in literature is state estimation using particle filters. Weber et al. propose a particle-based approach to track the location of enemy units that have been scouted by including a movement model for predicting their potential

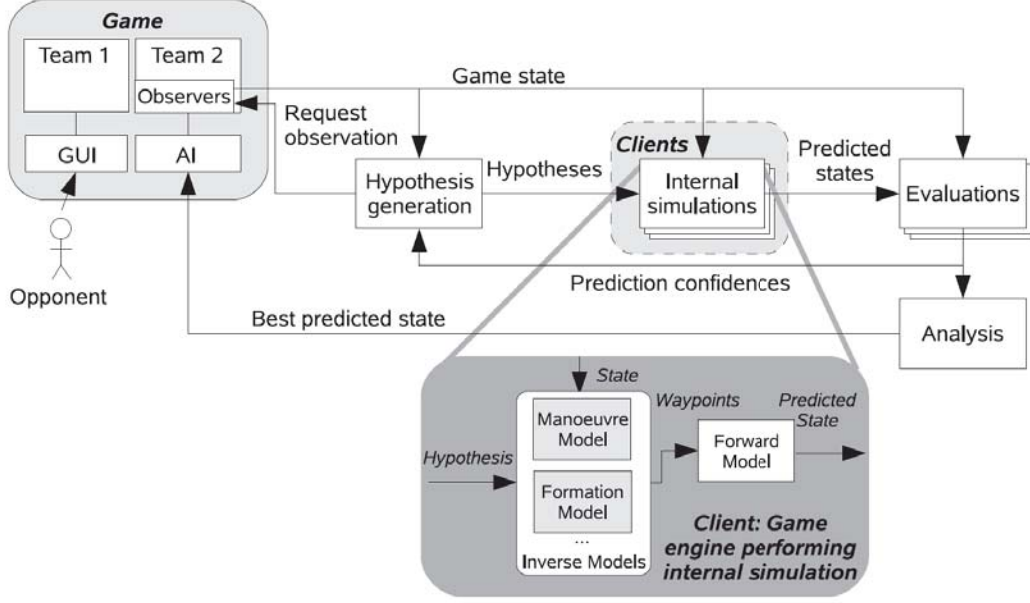


Figure 10. HAMMER incorporated into a partially-observable game [6]

trajectory and a decay function that gradually reduces the agent’s confidence in the prediction [40].

Not focusing on scouting, but also dealing with prediction models in RTS games, Butler et al. use simulation-theory concepts in their architecture dubbed HAMMER - Hierarchical Attentive Multiple-Models for Execution and Recognition of actions, see Figure 10. HAMMER is comprised of three main components: the inverse models (plan generators), which take in current world state and generates different parameters to set implicit and explicit target goals; the forward models (predictors), essentially provide an estimate of events to occur if the generated plan is followed; and the evaluator makes comparison of the results and assigns confidence values for selection of the best prediction [6]. This is similar to the use of MOEAs to assist the decision maker with various potential solutions.

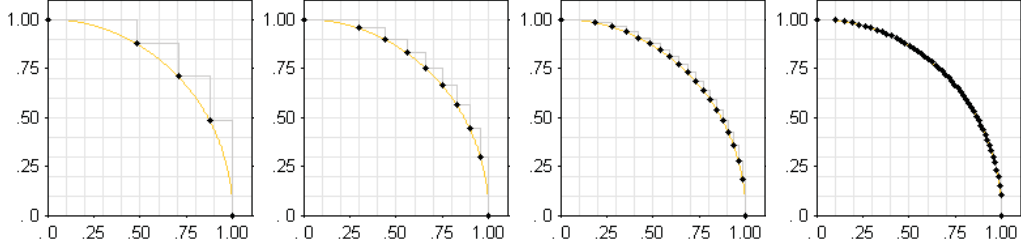


Figure 11. Example Pareto Front with Different Population Sizes (5, 10, 20, 50)

2.6 Multi-Objective Evolutionary Algorithms

Multi-Objective Evolutionary Algorithms (MOEAs) are methods used in the area of multiple criteria decision making for the purpose of solving problems by analyzing the potential results with regard to several competing objectives, which is a motivation for this effort [21]. As opposed to single objective problems, where the focus is on maximizing a single aspect of strategic importance, through either an equation or metric, a MOEA balances multiple such aspects. Weighting the objective, which often times involve tradeoffs, MOEAs enhances decision making, 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 [7]. An example of a Pareto front with 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 Multi-objective Optimizer, or PaGMO, is a C++ based algorithm platform developed by the European Space Agency that emphasizes parallel processing of common MOEA software through the utilization of an island model. PyGMO is a variant of PaGMO that implements a Python interface into the PaGMO system [41]. This is the framework of choice as it provides a smooth implementation in Python with the Spring RTS Engine.

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 [42].

ParadisEO.

ParadisEO is a C++ based MOEA software package that builds its design around modules for different focuses in MOEA structures [43]. The EO variant focuses on Evolving Objects. It allows for various “components” to be utilized based on the problem requirements [44].

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 [45].

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 are 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 Deb [46]. This algorithm was developed to be an improvement to the original NSGA algorithm. NSGA and 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 [46].

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 [47]. 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. The worst time complexity is by the truncation operator of $O(M^3)$, but on average the complexity will be $O(M^2 \log M)$ [47].

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 [48].

Particle swarm optimization (PSO) is a single objective algorithm inspired by behaviors of insects and animals that swarm together [49]. 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 [46]. 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 non-dominated 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 $O(mN^3)$ [48].

2.7 Chapter Summary

This chapter covers the different topics related to this investigation effort. It provides a background on RTS games and related AI research efforts. It follows with an overview of the different levels of RTS AI and general strategies. It then provides an understanding of decision making techniques and how they are used at different levels within the RTS environment. Additionally, it offers an overview of previous AFIT research efforts and a discussion of scouting RTS AI research techniques and approaches. Finally, it concludes with an overview of multi-objective optimization problems, some of the algorithms used, and the frameworks available.

III. Methodology for the Scouting Problem

As introduced in the first two chapters, RTS games allow for a wide range of complexed problems to be studied. A RTS game starts with a main base and few workers. A player must use these workers to gather resources and produce more workers, then expand and train army units to destroy the enemy. Scouting, the act of collecting timely information about the enemy’s order of battle while avoiding contact, is an essential activity for RTS gameplay. This is due to the existence of the fog-of-war, which covers the majority of the game map. Each player only has a certain range of view around the unit. Therefore, without sending reconnaissance units to explore the map and gather information, a player builds in the blind and has difficulty coming up with the right tactical plan to conquer or counteract the opponent [5].

However, this research effort focuses on the ability for an AI unit to collect data in a perfect information environment as a foundation and testbed before bringing it to an imperfect one on future work. The scouting function for this investigation is abstracted as a mathematical model to find an “optimal” solution with the use of MOEAs. This chapter discusses the high- and low-level designs of the problem space, more specifically, the multi-objective scout orienteering problem (MOSOP). It begins with defining the problem and describing various aspects of the problem space mathematically. It then provides a description of the MOEA software and MOEAs used for the evaluation of the MOSOP. Finally, this chapter describes the real-time execution of scout units and the integration of the MOSOP within the AFIT RTS AI using the Spring Engine.

3.1 High-level Design: Defining the Problem Domain

Before a solution can be placed into the Spring Engine and the AFIT agent, it is best to understand what exactly is being solved. This section describes the problem domain to include the different objectives and constraints, provides diagrams and graphs for insight into the problem domain, defines the mathematical model of the MOSOP, and provides general concepts of interest.

Orienteering Problem.

Introduced by Tsilirigidis, the Orienteering Problem (OP) is a generalization of the traveling salesman problem (TSP) [50]. The OP actually comes from a class of TSP problems named the TSP with Profits or the Selective TSP [51]. Golden et al. proved that the OP is NP-hard, meaning no deterministic polynomial time algorithm has been designed to solve this problem to optimality [52]. The OP is based on an outdoor sport called Orienteering, which is played in heavily forested areas. Located in the forest are a set of control points each with an associated score. Competitors armed with compass and map are required to visit a subset of the control points. In the OP, a set of N vertices i is given, each with a score S_i . The starting point (vertex 1) and the end point (vertex N) are fixed. The time t_{ij} needed to travel from vertex i to j is known for all vertices. Not all vertices can be visited since the available time is limited to a given time budget T_{max} . The goal of the OP is to determine a path, limited by T_{max} , that visits some of the vertices, in order to maximize the total collected score. The scores are assumed to be entirely additive and each vertex can be visited at most once.

Multi-Objective Orienteering Problem.

The multi-objective orienteering problem (MOOP) is an extension of the OP [53]. Like the OP, the MOOP can be defined as a directed graph $G = (V, E)$ where $V = \{v_1, \dots, v_{n+1}\}$ is a set of vertices and $E = \{(v_i, v_j) : v_i, v_j \in V \text{ and } i \neq j \text{ and } i \neq n+1 \text{ and } j \neq 0\}$ is a set of edges. A certain number of K benefit values s_{ik} ($k = 1 \dots K$) is assigned to each vertex $i \in V \setminus \{v_0, v_{n+1}\}$. The starting vertex v_0 and the ending vertex v_{n+1} are assigned benefit values of zero for each objective [53]. Additionally, every edge $(v_i, v_j) \in E$ is assigned a cost value c_{ij} which we interpret as time to travel when going from vertex i to vertex j . Schilde et al. [53] refers to a MOOP *tour* as a path where the starting and ending vertex are the same point ($v_0 = v_{n+1}$).

Figure 12 is the example of a five-vertex graph created for this investigation. A small graph is selected for research purposes since the MOOP is a generalization of the TSP and it is known to have a complexity of $O(N^N)$, where N is the number of vertices [51, 50, 53]. Using MOEAs in a graph this size may not seem appropriate, since the values can be enumerated, but is great for testing to show if the problem is solvable.

Multi-Objective Scout Orienteering Problem.

In order to achieve the goal of maximizing the resources and enemy buildings discovered, while minimizing the risk of being found or killed, thus revealing information hidden in the fog-of-war as fast as possible, three objectives are modeled as the Multi-Objective Scout Orienteering Problem (MOSOP). The first two are maximization objective functions, which include the total resources (f_r) and enemy buildings (f_b) found. The other is a minimization objective function, the total enemy units (f_e). This investigation adapts the generic model presented by Schilde et al., which can be applied to any number of objectives [53].

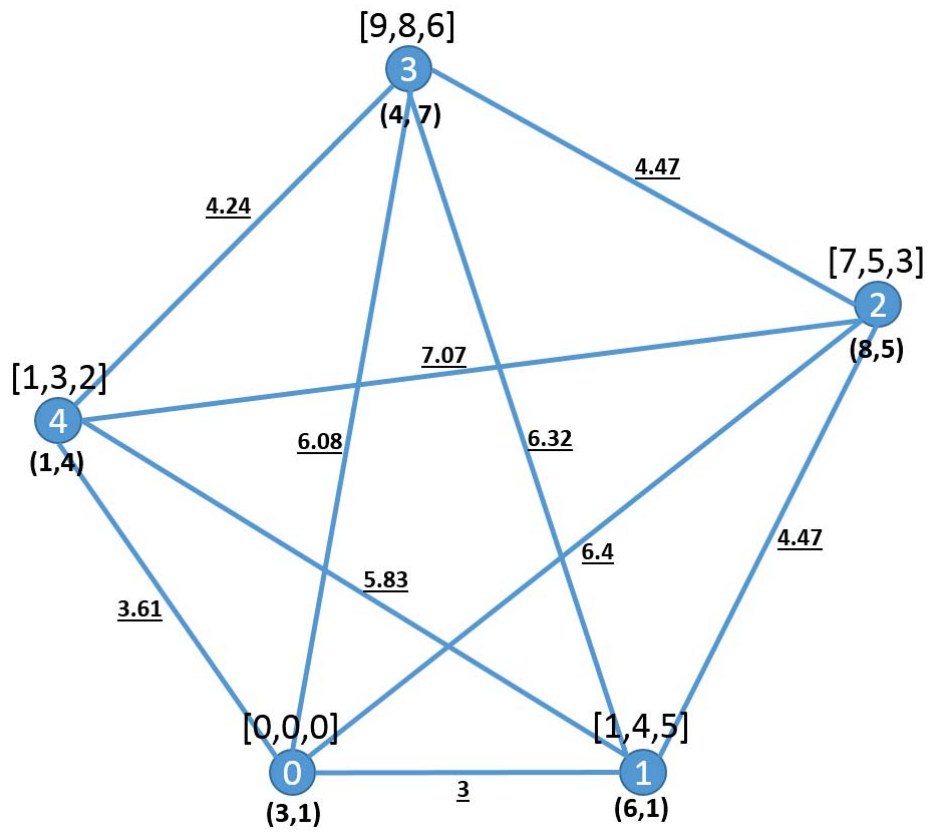


Figure 12. Example Graph of Multi-Objective Orienteering Problem with five nodes.

Objective Function 1 - Maximize Resources.

As a scout unit moves across the map, it looks for areas to harvest resources, thus maximizing the potential for the friendly forces to expand their economy and build a stronger army. Equation 1 represents the sum of the maximum benefit collected in the path taken by the unit, where s_{ir} is the benefit value of the resource at vertex i and y_i is the vertex included in the path.

$$\max f_r(y) = \sum_{v_i \in V \setminus \{v_0, v_{n+1}\}} s_{ir} y_i \quad (1)$$

Objective Function 2 - Maximize Enemy Buildings.

Along the same lines as the previous objective function, a scout unit looks for areas where the enemy has their buildings, thus maximizing the potential of finding their base, which is one of its main goals. Once the enemy base is found the commander can direct his or her army to destroy and thus winning the game. This objective goes in hand with the scouting strategy of finding the opponent's base and collecting information about their technologies or units. Equation 2 represents the sum of the maximum benefit collected in the path taken by the unit, where s_{ib} is the benefit value of the enemy building at vertex i and y_i is the vertex included in the path.

$$\max f_b(y) = \sum_{v_i \in V \setminus \{v_0, v_{n+1}\}} s_{ib} y_i \quad (2)$$

Objective Function 3 - Minimize Enemy Risk.

While moving across the map, the scout unit must avoid encountering the enemy in order to survive long enough to discover resources or enemy buildings (enemy base), which is why it is important to minimize enemy risk. This objective goes in hand with the scouting strategy of avoiding taking damage or being killed. Equation 3

represents the sum of the minimum benefit or risk in the path taken by the unit, where s_{ir} is the value of the resource at vertex i and y_i is the vertex included in the path, where s_{ie} is the risk value of the enemy at vertex i and y_i is the vertex included in the path.

$$\min f_e(y) = \sum_{v_i \in V \setminus \{v_0, v_{n+1}\}} s_{ie} y_i \quad (3)$$

Conditions and Constraints.

The previous objective functions are subject to some conditions and constraints.

Subject to

$$y_0 = y_{n+1} = 1 \quad (4)$$

$$\sum_{\{v_i, v_j\} \in A} c_{ij} x_{ij} \leq T_{max} \quad (5)$$

$$x_{ij} \in \{0, 1\} \quad ((v_i, v_j) \in A) \quad (6)$$

$$y_i \in \{0, 1\} \quad (v_i \in V) \quad (7)$$

Schilde et al.'s MOOP function also had some constraints, the are adapted to the MOSOP. Constraint (4) makes the tour start and end at the correct points; and constraint (5) limits the total tour duration. (6) associates one binary variable equal to 1 if and only if the corresponding arc is used in the solution, 0 otherwise; and one binary variable equal to 1 if and only if the corresponding vertex is visited, 0 otherwise in (7) [53].

The aim of Schilde’s MOOP [53] is to find all Pareto efficient tours that start at vertex v_0 , end at vertex v_{n+1} and do not violate the maximum tour length restriction T_{max} . The MOSOP is modeled using this restriction as well.

3.2 MOEA Design Domain

This section provides an overall, top-down algorithm design for how a multi-objective problem is solved from the problem domain. When designing an algorithm, the first thing we accomplish is an outline of the required steps to effectively complete the algorithmic process. Algorithm 1 represents a general, high-level outline of this process for a multi-objective problem [7]. The previous section covered the first step; declared the problem definition, to include the objective function and how it would be evaluated - conditions and constraints. The next section looks at steps 2 through 5 from Algorithm 1 for MOEA implementation.

Algorithm 1 High-level Pseudocode of a Multi-Objective Problem

- 1: **Step 1:** Declare problem definition
 - 2: **A:** Receive inputs
 - 3: **B:** Define evaluators for each function
 - 4: **C:** Return values of objective functions
 - 5: **Step 2:** Generate a population for the problem
 - 6: **A:** Create a population P of desired size
 - 7: **B:** Randomly generate decision variables within range
 - 8: **Step 3:** Choose an MOEA to evaluate the problem
 - 9: → Set # of Generations
 - 10: → Evaluate against constraints
 - 11: → Determine crossover and mutation rates
 - 12: → Declare any MOEA specific parameters
 - 13: **Step 4:** Apply MOEA to evolve the population
 - 14: **Step 5:** Return the final population as a solution
-

3.3 MOEA Implementation

Chapter 2 provided a background on the different frameworks and MOEAs studied or used for this research effort. This section details the process of selecting and implementing an MOEA software as well as the MOEAs themselves. Additionally, it explores Step 3 in Algorithm 1, which states that an MOEA must be chosen to evaluate the problem.

MOEA Software.

There exist various MOEA software, or frameworks, that are capable of evaluating Multi-objective problems. Programs such as *jMetal*, *MOEA Framework* and *PyGMO* allow for customization of the problem space to apply well known MOEAs and access a solution [41, 42, 45]. For this effort, PyGMO is selected because it can be integrated using Python into the AFIT RTS framework. jMetal is valuable software due to its ease of use and capability to run a study of different problems and MOEAs simultaneously. However, it is written in Java and does not provide easy integration with the Spring Engine. Similarly, MOEA Framework showed great potential, but is also written in Java.

PyGMO is used as follows: First, define 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 14 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()           #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 14. An example of solving a problem through *PyGMO* [8]

Selecting an MOEA.

With the selection of PyGMO as the MOEA framework of choice, a number of algorithms can be used to evaluate multi-objective problems. These algorithms are briefly explained in Chapter 2. To evaluate the problem domain, the following are used: Non-dominated Sorting Genetic Algorithm II (NSGA II), Strength Pareto Evolutionary Algorithm 2 (SPEA 2), and Non-dominated Sorting Particle Swarm Optimization (NSPSO).

Algorithm 2 NSGA II - Pseudocode [46]

```

1: draw multiset  $P$  with  $\mu$  elements  $\in X$  at random
2: repeat
3:   generate  $\mu$  offspring  $x_1, \dots, x_\mu \in X$  from  $P$  by variation
4:    $P = P \cup \{x_1, \dots, x_\mu\}$ 
5:   build ranking  $R_1, \dots, R_h$  from  $P$ 
6:   set  $P = \emptyset$  and  $i = 1$ 
7:   while  $|P| + |R_i| \leq \mu$  do
8:      $P = P \cup R_i$ 
9:      $i = i + 1$ 
10:  end while
11:   $k = \mu - |P|$ 
12:  if  $k > 0$  then
13:    remove those  $|R_i| - k$  elements from  $R_i$  that have smallest crowding distance
14:  end if
15:   $P = P \cup R_i$ 
16: until stopping criterion fulfilled

```

1. **NSGA-II** sorts the population by non-domination ranking before creating new children. (See Algorithm 2)

2. **SPEA2** maintains a record of the best combinations found during execution of the algorithm.
3. **NSPSO** adapts the same method as NSGA-II, however, it merges the new elite set with previous non-dominated members for the next generation.

3.4 Integration into the AFIT RTS AI Platform

After developing a mathematical model of the problem and selecting an MOEA during the offline simulation, the next step is to put the scout into action and implement it into the AFIT RTS AI agent. The next few sections discuss various modules within the AFIT RTS AI agent, as well as the implementation of the algorithm into the proper module. Additionally, it discusses how to issue commands to units within the Spring Engine such as moving and patrolling, and how these are used to move the scout unit.

AFIT RTS AI Platform.

The AFIT RTS AI platform provides a working framework for interfacing with the Spring Engine [1, 4, 9, 8, 10]. A simple modification is the Balanced Annihilation game is implemented within this investigation. This mod is all that is required to provide adequate testing on how the algorithm designed in this effort interfaces with the platform.

Structure/Design.

The AFIT RTS AI agent was originally developed by DiTrapani [9]. This agent is composed of several smaller modules that communicate with one another as it progresses through the game. Even though DiTrapani's effort focused on strategic level, it also allows build for tactical level game execution. The various different structures,

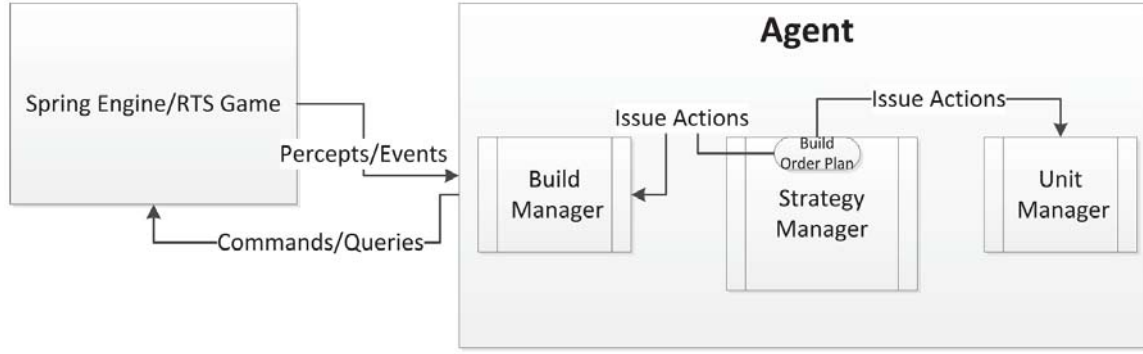


Figure 15. AFIT Multi-Scale Agent Architecture [1]

which are still being used have a unique role within process of execution. 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 [1] (Figure 15).

Gruber and Ball’s research focused on development of the tactical and positional aspect of the agent [4, 8]. There is a python file titled *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 ones like Gruber’s targeting MOEA and Ball’s positional MOEA. The current effort adds scouting, an essential tactical task, to these tactical options as the MOSOP.

Currently, the AFIT RTS AI framework is used with perfect information, therefore the agent knows exactly where the enemy units are at all times and can make decisions. The platform uses simple functions that are helpful, provided by the agent’s structure such as: *assign move* or *assign patrol* as well as qualifiers to know if a unit is within Line of Sight of an enemy. Future work can add functionality in an environment with the FoW on (imperfect information).

Scout Integration.

After optimizing the scouting problem within PyGMO, the results, determined by the optimization software choosing the “champion” member of the population from

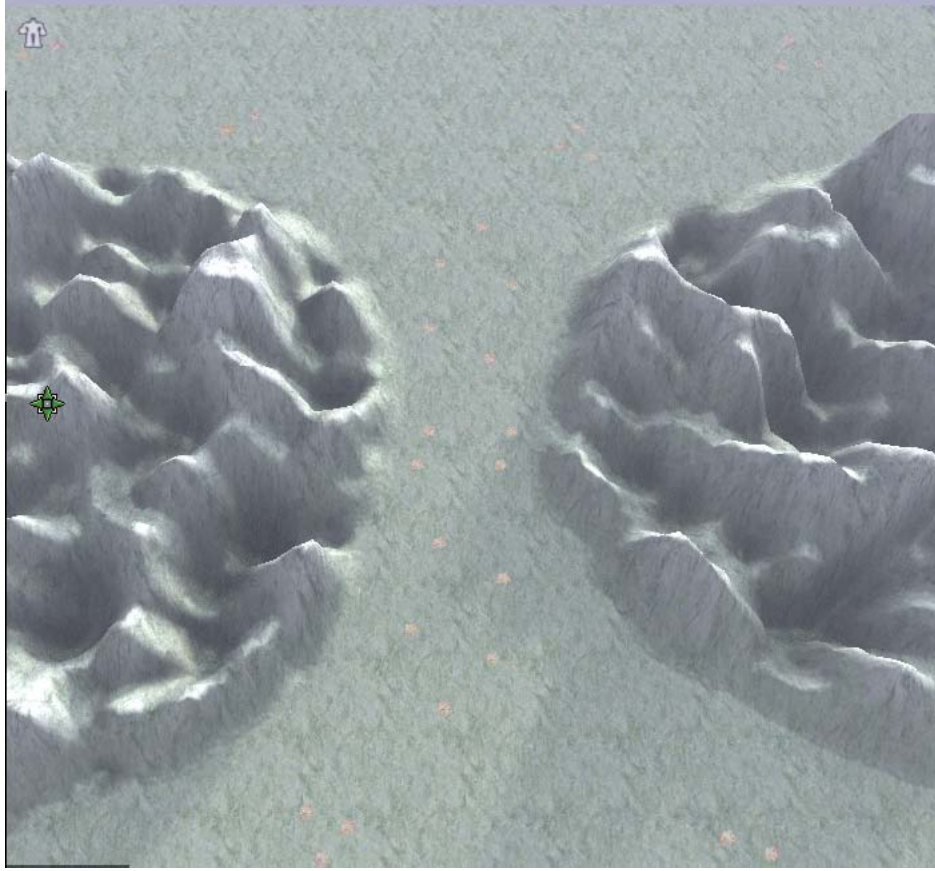


Figure 16. The testing map “The Pass”

the solution space, are taken and applied to the scout unit module using different scenarios [41]. These scenarios are run with scripted commands first to determine which of the two scouting units available in the Spring Engine performs better, and then that select unit is further integrated using the MOSOP with the AFIT RTS AI framework. The scout units are run functional testing trials, factors such as movement range, actions inside and outside of line of sight, and movement across the map determined how to properly configure the agent and prepare it for the MOSOP experimentation.

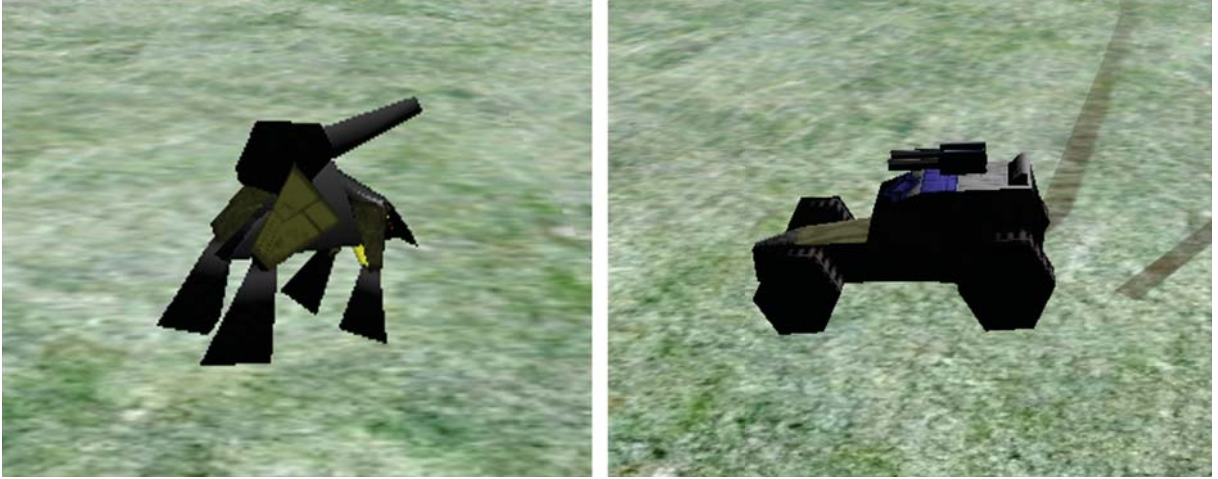


Figure 17. Fast Scout Kbot - Flea (left); Light Scout Vehicle - Jeffy (right)

Table 2. Characteristics comparison of Jeffy and Flea

Characteristics	Flea	Jeffy
Max Velocity	4.4	6.4
Sight distance	550	585
Acceleration	0.5	0.12
Breakrate	1.5	0.495
Turnrate	1672	1144

Map Choice.

The map chosen for testing is the same map used in all previous AFIT Agent tests, *The Pass*, because it has a simple design. It provides a perfect combat scenario for a scout unit to explore the environment, especially the choke point in the valley created by the mountains (Figure 16). While many other maps exist within the Balanced Annihilation mod of the Spring Engine, this map is among the least complex [54].

Unit Selection.

In Balanced Annihilation (BA) there are two units that can be used for scouting purposes. There is the *Fast Scout Kbot*, also known as Flea, and the *Light Scout Vehicle*, or Jeffy (Figure 17). This investigation evaluates both of these BA scout units. Table 2 provides some of their characteristics for comparison purposes.

Establishing a Scenario.

Both units, Flea and Jeffy are compared and subject to different scenarios to test the unit's functionality and effectiveness conducting the three scouting strategies with perfect information using scripted commands. After testing their functionality to see which unit performs better, thus reducing the amounts of trials in the next part, the selected scout unit is run through one scenario from each of the three scouting strategies using the MOSOP tactical option with the selected MOEA to evaluate the complete integration.

1. Find the opponent's base as soon as possible:

- (a) The scout unit is sent to the known enemy position, in order to test the effectiveness of the unit patrolling there and returning to friendly base when in the line of sight of the enemy base.
- (b) The scout unit is sent to random coordinates of the map where the opponent's base is typically placed, in order to test the effectiveness of the unit finding the opponent's base.

2. Collect information about the opponent's technologies and units:

- (a) Send the scout through "The Pass" to check if it can detect the presence of the enemy units.
- (b) Send the scout through "The Pass" to check if it can detect the presence of the enemy buildings.

3. Avoid being killed or damaged:

- (a) Evaluate the scout's survivability by sending the unit near the enemy on a random path.

- (b) Evaluate the scout's survivability by sending the unit near the enemy, knowing is a high risk to see if it can survive.

3.5 Chapter Summary

This chapter presents the high- and low-level designs of the problem space, more specifically, the multi-objective scout orienteering problem (MOSOP). It defines the problem and describes various aspects of the problem space mathematically. It provides a description of the MOEA software and MOEAs used for the MOSOP. Finally, it discusses the AFIT RTS AI platform and how to integrate the scout unit and MOEA in an online simulation using the Spring Engine.

IV. Design of Experiments and Testing of Scouting

4.1 Introduction

This chapter discusses the design of the experiments and tests performed to evaluate the effectiveness of the scout orienteering problem detailed in Chapter 3. It describes the testing environment for the evaluation of the different MOEAs. It details the number of tests conducted and metrics used to measure their effectiveness. It discusses the experiments to evaluate two scout units within the AFIT RTS AI platform and the integration of a selected scout unit using the chosen MOEA from previous experiment. Several metrics are use to evaluate this experimental objective: To integrate the “best” scouting path into the Spring Engine - Balanced Annihilation game.

4.2 Experiment Equipment

This investigation uses similar experimental equipment as those conducted by Gruber and Ball [4, 8]. The different MOEAs are tested using PyGMO version 1.1.5 on a 64-bit computer running Ubuntu 14.04. Similarly, the MOEA is integrated in to the AFIT RTS AI platform using Spring Engine version 98.0.1 and the Balanced Annihilation RTS game. Table 3 provides an overview of the hardware and software used in the experiments.

Table 3. Overview of hardware and software used in the experiment

Hardware	Model	Software	Version
Processor	Athlon II Dual Core @2GHz	Ubuntu	14.04 LTS-64bit
Memory	2GB DDR3	PyGMO	1.1.5
Graphics	Gallium 0.4, AMD RS880	Spring RTS	98.0.1-516-ga626219

4.3 Finding the Best MOEA

The first objective in this investigation is to design a mathematical model, the second is to generate an algorithmic solution to the problem using MOEAs. This is done by integrating the Multi-objective Scout Orienteering Problem model, using the five vertex graph in Figure 12 into a multi-objective framework, and selecting the best MOEA. The expected outcome of this objective is a MOSOP solvable through the use of MOEAs.

Testing MOEAs.

As discussed in Chapter 3, the multi-objective framework of choice for the experimentations is PyGMO. This framework provides various different algorithms for evaluating single and multi-objective problems [41]. The experiments conducted compares the effectiveness of the MOSOP when tested with NSGA-II, SPEA2 and NSPSO, thus to helping in the selection process of the best MOEA to integrate the MOSOP in the AFIT RTS AI platform. These three MOEAs are selected because they cover a variety of generic search procedures and have worked well in previous work [4, 8]. These three MOEAs, explained in detail in Chapter 2, cover the random, spread, and focused search processes. The parameters to test an MOEA optimization problem are population size and numbers of generations. To evaluate the MOSOP, this experiment used the values in Figure12, population sizes of 20, 40, 60, 80, and 100, and generation sizes of 2, 3, 4, and 5, since it is a small graph and does not require to explore large number of solutions. Altogether this gives 20 different experiments per MOEA, or 60 different experiments total. Furthermore, each experiment is run through 30 iterations in order to create a statistical baseline for analysis. Additionally, the PyGMO framework provides a function that selects a “best” solution to apply in the integration with the AFIT RTS AI platform.

Metrics for Analysis.

This investigation uses two quality indicators, a metric that evaluates the quality of the solutions generated by a MOEA based on their proximity to the Pareto front, hypervolume and spacing [7]. Other metrics used are Pareto front structure and execution time.

- **Hypervolume:** Used for Pareto fronts involving more than two objectives, attempts to equate the multidimensional area or volume covered by a given population of potential solutions. A solution with a higher area or volume is often preferable to one with smaller value [7, 55].
- **Spacing:** A measure of the average distance between population members on the Pareto Front, which determines if clustering is occurring amongst the population. A larger spacing ensures a wider diversity amongst the population [56].
- **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.
- **Execution Time:** Measures the running time for each MOEA, which allows for selection of the fastest running algorithm even though it is not guaranteed that is the best performing one.

4.4 Integrating the Selected MOEA into the AFIT RTS AI Platform

The third objective in this investigation is evaluate the scouting unit in an online experiment with the Spring Engine. Once the best MOEA for the MOSOP is selected, it is integrated into the engine to run with AFIT RTS AI platform. However, two

scout unit are first examine to evaluate their effectiveness and select the best unit to continue with the integration of the MOSOP problem using the selected MOEA. These scouts are integrated at the tactical level, using Gruber’s code, version 0.4.3b, and Ball’s version 0.4.3c, to create this version, 0.4.3d . The expected outcome of this objective is the selecting a of best performing scout unit and the effectiveness of the MOSOP tactical option integration to find the “best” path.

Maneuvering a Scout Unit.

As described in Chapter 3, the AFIT RTS AI platform makes use of two primary files - *agent.py* and *group.py*. These files are use to add functionality to units being created. The MOSOP is particularly placed in the *group.py* file, which adds control to group of units either *moving* or *attacking*. To these control capabilities, the *patrol* command is added. The patrol command allows the scout unit to go from an initial point on the map to one designated either by scripting it or at random and back to initial point. The primary functionality of the selected scout is controlled by the *move* command through the optimization algorithm, which selects the points in the map produced by the PyGMO optimization framework. For testing purpose, this investigation uses a “cheat” script to spawn up the scouting units at the beginning of each run.

Testing the Scout Units.

Two scouting units are evaluated using scripted commands to select the best performing scout, since they have different characteristics. The selected scout is then examined with the MOEA of choice to find the “best” path. This is done by running several scenarios using the three scouting strategies previously discussed: (1) Find the opponent’s base as soon as possible, (2) Collect information about the opponent’s

technologies and units, and (3) Avoid being killed or damaged. These scenarios are run against three other tactical options from previous version - *default*, *moea*, and *moea2* and measured by either success or failure after running 10 iterations of each, which establishes a statistical basis. Overall, there are two sets of evaluations, one set with scripted, or assigned, points of interest and one with points given by the integrated MOEA.

1. Find the opponent's base as soon as possible:

- (a) The scout unit is sent to the known enemy position, in order to test the effectiveness of the unit patrolling there and returning to friendly base when in the line of sight of the enemy base (Figure 18).
- (b) The scout unit is sent to random coordinates of the map where the opponent's base is typically placed, in order to test the effectiveness of the unit finding the opponent's base.

2. Collect information about the opponent's technologies and units:

- (a) Send the scout through "The Pass" to check if it can detect the presence of the enemy units. (Figure 19).
- (b) Send the scout through "The Pass" to check if it can detect the presence of the enemy buildings.

3. Avoid being killed or damaged:

- (a) Evaluate the scout's survivability by sending the unit near the enemy on a random path.
- (b) Evaluate the scout's survivability by sending the unit near the enemy, knowing is a high risk to see if it can survive (Figure 20).



Figure 18. Example of Scout Finding the Enemy's Base

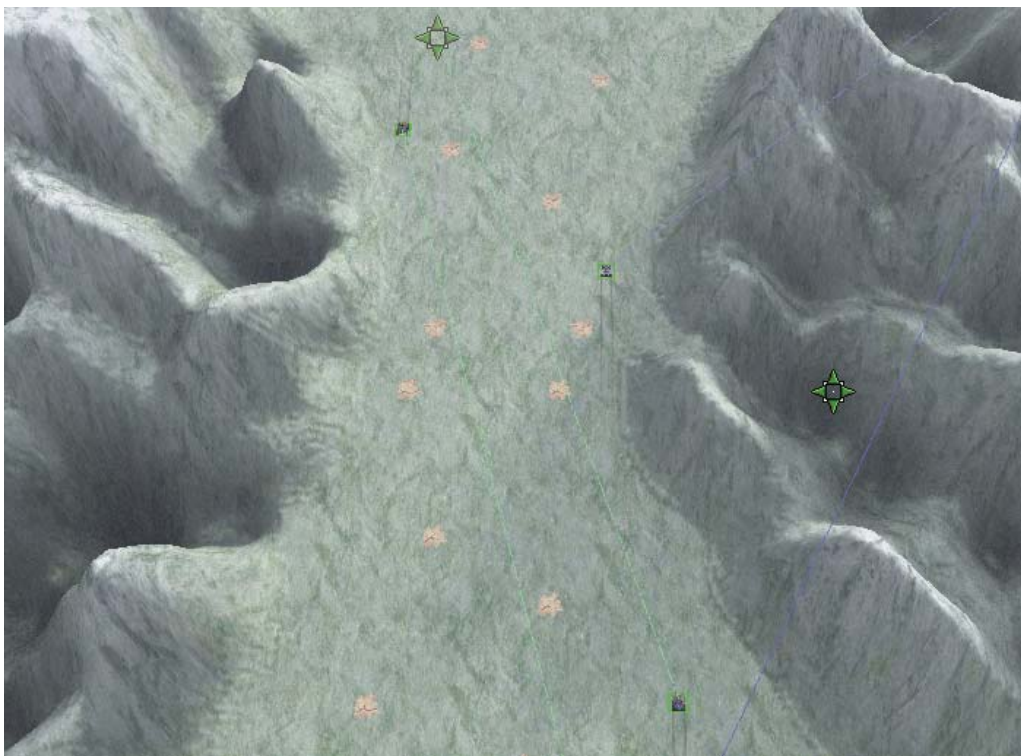


Figure 19. Example of Scout Avoiding Contact



Figure 20. Example of Scout Outside LOS of enemy

4.5 Chapter Summary

This chapter discusses the design of the experiments and tests performed to evaluate the effectiveness of the scout orienteering problem detailed in Chapter 3. It describes the testing environment for the evaluation of the different MOEAs. It details the number of tests conducted and metrics used to measure their effectiveness. It discusses the experiments performed to evaluate two scout units within the AFIT RTS AI platform and the integration of a selected scout unit using the chosen MOEA from previous experiment through several scenarios. It also details how these scenarios measure the experimental objective of integrating the “best” scouting path into the Spring Engine - Balanced Annihilation game.

V. Analysis of Results

5.1 Introduction

This chapter presents the results from the experiments detailed in Chapter 4 and discusses their analysis. It begins with comparing and contrasting the effectiveness and efficiency of using the different MOEAs (NSGA-II, SPEA2 and NSPSO) to solve the MOSOP in the offline simulation. It also details the results of performance between the two scouting units being tested and the integration of this scout unit using the MOSOP into the AFIT AI platform against other tactical options.

5.2 Results from MOEA Evaluation

This section covers the analysis of the MOEAs used within the PyGMO framework. It compares the effectiveness and efficiency of the NSGA-II, SPEA2 and NSPSO algorithms to solve the MOSOP. Each algorithm is analyzed by the metrics established in the previous chapter - hypervolume, spacing, Pareto front structure, and execution time. The analysis is conducted using all the results generated during testing, however, only a few sample graphs are depicted (results for each variations presented in Chapter 4 are in Appendix A).

Analyzing the Hypervolume Metric.

A sample graph is shown for the analysis of hypervolume (HV), with HV values on the vertical axis and the different MOEAs on the horizontal axis - Figure 21. The result of the experiments show that after population size 60 there was no significant variation between NSGA-II and SPEA 2. Having used a five vertex graph for this MOSOP, the Pareto fronts converged rather quick since it is not that exhaustive to come up with solutions on a small graph. The purpose of the size of this graph was

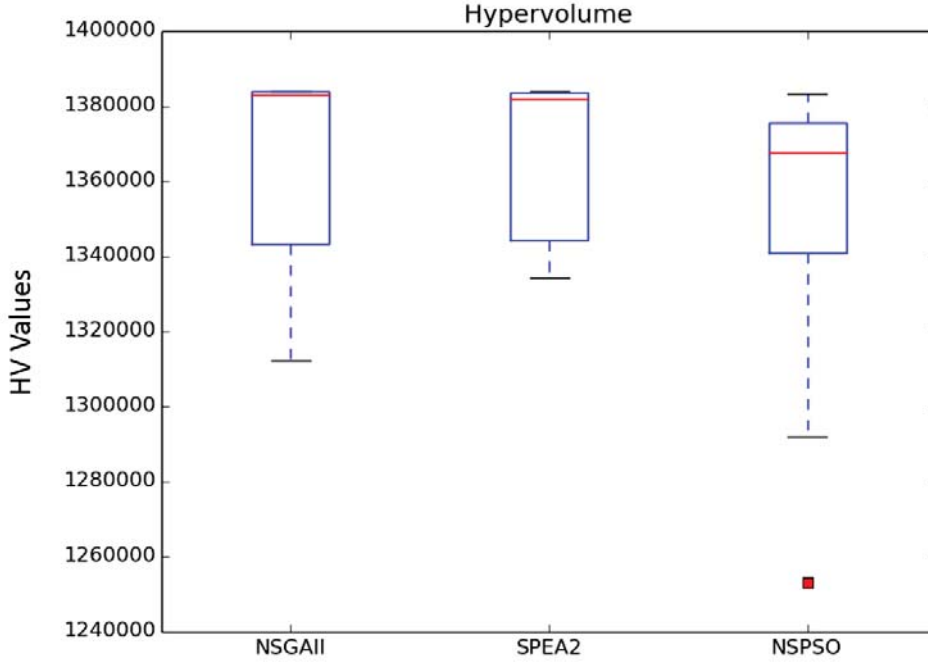


Figure 21. MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 5

Table 4. Hypervolume results - Pop. 20 Gen. 5

MOEA	Mean	Median	Variance
NSGA-II	1.3660e+06	1.3831e+06	5.3861e+08
SPEA2	1.3659e+06	1.3821e+06	4.5354e+08
NSPSO	1.3508e+06	1.3675e+06	1.2013e+09

to test if the MOSOP was solvable and the results show that it is. Table 4 provides another visual representation providing the mean, median, and variance.

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 solution space, allowing it to be guided towards a more desirable solution much easier. Figure 22 depicts the spacing results of the experiment for each of the three MOEAs with a population of 20 and a generation

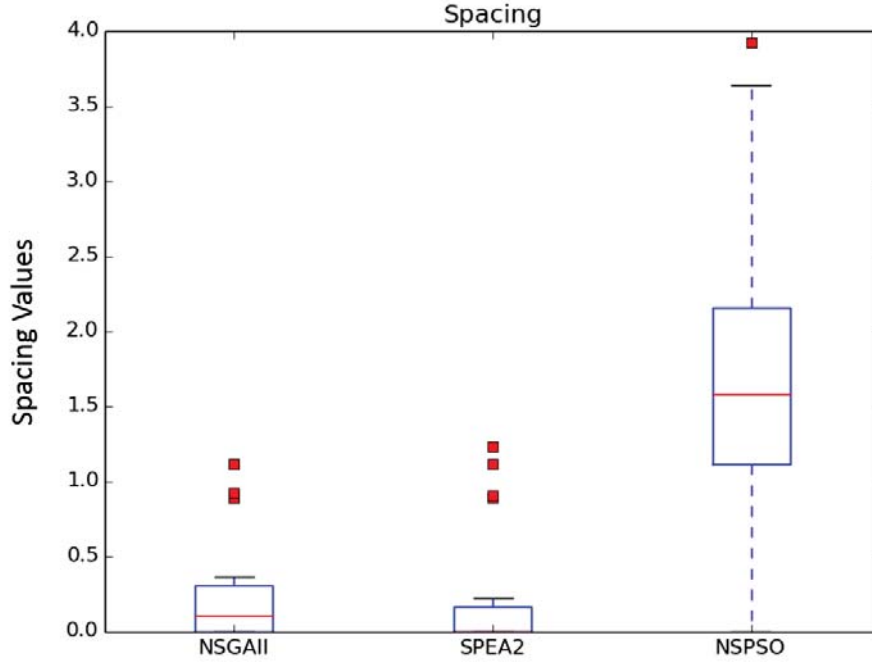


Figure 22. MOEAs Spacing Comparison Sample - Pop. 20 Gen. 5

Table 5. Spacing results - Pop. 20 Gen. 5

MOEA	Mean	Median	Variance
NSGA-II	2.8116e-01	1.1180e-01	1.3893e-01
SPEA2	1.6828e-01	0.0000e+00	1.2484e-01
NSPSO	1.6663e+00	1.5850e+00	9.7156e-01

of 5. Overall, SPEA 2 shows have better spread of the vectors from PF_{known} , since SPEA 2 uses archive to store nondominated solutions found through the generational process. Table 5 provides another visual representation providing the mean, median, and variance.

Pareto Front Structure.

Overall, the Pareto front of the three MOEAs are very similar in structure. The values shown on the graph represent the three objective functions. Negative values are the representation by the optimization framework of the two maximization objectives,

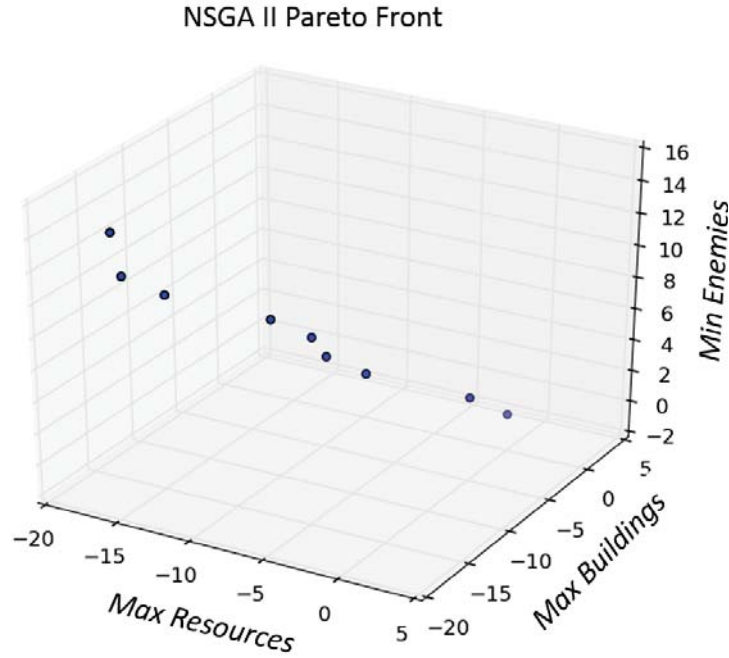


Figure 23. NSGA-II Sample Pareto Front - Pop. 20 Gen. 4

but can be think of as positive. The NSGA-II Pareto front, as shown in Figure 23, illustrates a disconnected front consisting of several noncontiguous points. Similarly, the sample SPEA2 Pareto front (Figure 24) shows a disconnected structure and one less point being mapped from the decision space. The NSPSO Sample, illustrated in Figure 25 shows a much looser Pareto front. This is not surprising given the nature of NSPSO, as it is a less connected and faster algorithm.

Analyzing the Execution Time.

The execution time metric measures how much time an MOEA is spending on calculations. Figures 26 and 27 show boxplots of the execution time for all three MOEAs with two different parameters. As for this metric, NSPSO outperformed both NSGA-II and SPEA 2. SPEA 2 was faster than NSGA-II when testing a lower population size. The data is gathered using Python's *time.time()* function, which is

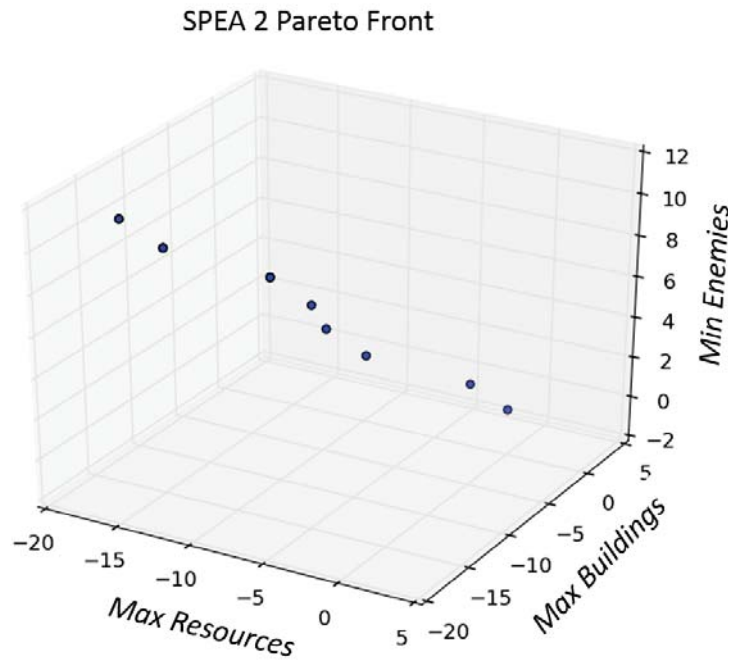


Figure 24. SPEA2 Sample Pareto Front - Pop. 20 Gen. 4

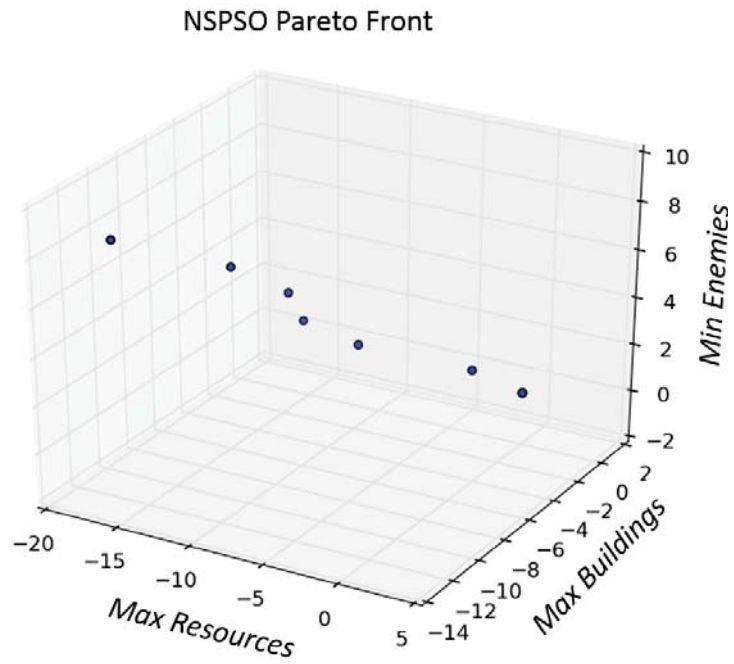


Figure 25. NSPSO Sample Pareto Front - Pop. 20 Gen. 4

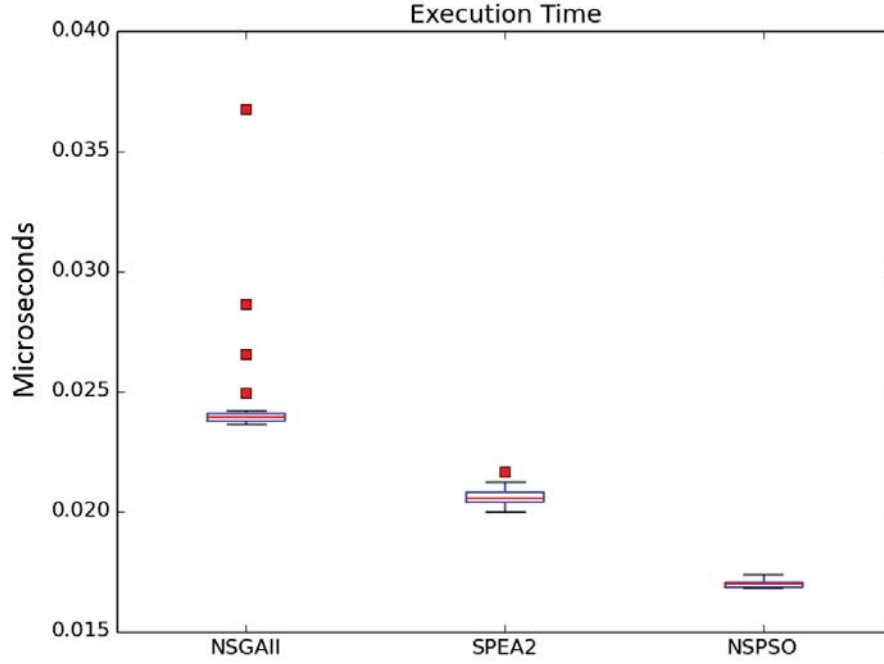


Figure 26. MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 5

Table 6. Execution time results - Pop. 20 Gen. 5

MOEA	Mean	Median	Variance	Shortest	Longest
NSGA-II	$24.6\mu s$	$23.9\mu s$	$60.1\mu s$	$23.6\mu s$	$36.7\mu s$
SPEA2	$20.6\mu s$	$20.5\mu s$	$14.1\mu s$	$20\mu s$	$21.6\mu s$
NSPSO	$17\mu s$	$17\mu s$	$19.1\mu s$	$16.8\mu s$	$17.3\mu s$

a floating point number of the seconds since 1 January 1970 [57]. The results can be best represented in microseconds, where the spread can be seen in Table 6.

Overall Analysis of MOEA Testing.

Overall, the NSGA-II shows to be the better of the three when running and solving the MOSOP. This means there is a balance between a quality of solution and speed at which a solution for the MOSOP is acquired. It is assume that the NSGA-II would also be the MOEA of choice even for a problem with larger number of vertices. The PyGMO framework generated a “best” of the path, for example, [0, 2, 3, 4, 0],

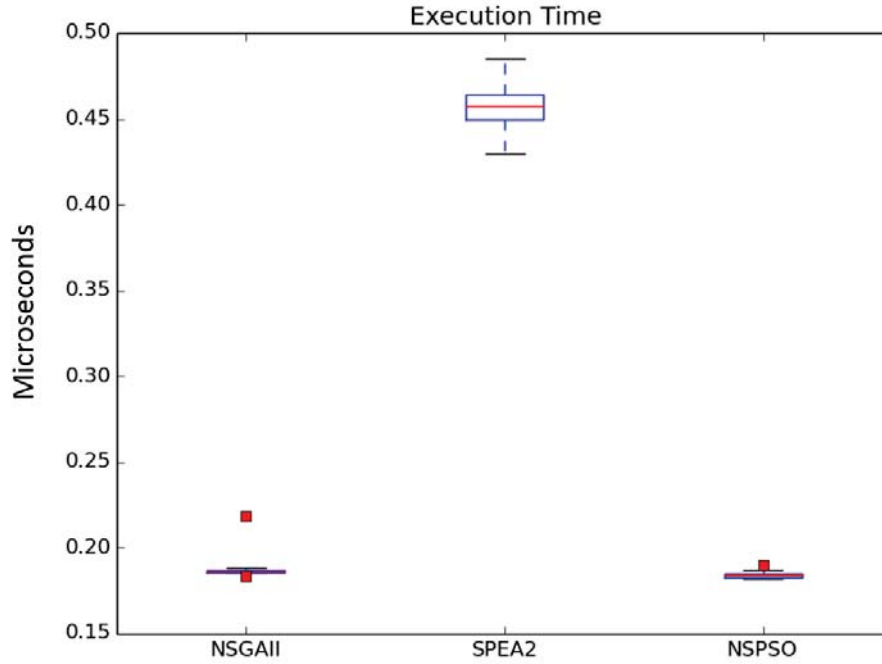


Figure 27. MOEAs Execution Time Comparison Sample - Pop. 100 Gen. 5

which is used with the selected scout unit in the AFIT RTS AI platform. The overall results show that the multi-objective scouting problem is solvable through the use of MOEAs.

5.3 Results from Testing the Scouting Units

This section covers the analysis of various scenarios and tactical options using NSGA-II to integrate the MOSOP into the AFIT RTS AI using the Spring Engine. It compares the performance of the scout units, Jeffy and Flea, functionality using the *scout* tactical option against the *default*, *moea*, and *moea2* tactical options. Once the best performing unit is selected, it is then run through the first scenario in each scouting strategy using the MOSOP. The results of these experiments are measured against the scenarios and metrics established in Chapter 4.

Evaluating the Scenarios.

Each scenario was run with each of the two scout units discussed in Chapter 3 and testing the *scout* tactical option against the *default*, *moea*, and *moea2* tactical options for 10 iterations to form a statistical basis. Success for each scenario is define as the scout unit accomplishing each scenario, surviving at least 7 out of the 10 iterations or more than 70

1. Find the opponents base as soon as possible:

- (a) **The scout unit is sent to the known enemy position, in order to test the effectiveness of the unit patrolling there and returning to friendly base when in the line of sight of the enemy base.** Both units are successful in patrolling towards and finding the opponent's base and once in line of sight of the enemy, returning to their own base. This validates the patrol command and the line of sight restriction.
- (b) **The scout unit is sent to random coordinates of the map where the opponent's base is typically placed, in order to test the effectiveness of the unit finding the opponent's base.** Both units are successful in patrolling towards the randomly assigned points. This validates the patrol command and the line of sight restriction.

2. Collect information about the opponent's technologies and units:

- (a) **Send the scout through "The Pass" to check if it can detect the presence of the enemy units.** Both units are successful at detecting the presence of enemy units while traversing through "The Pass."
- (b) **Send the scout through "The Pass" to check if it can detect the presence of the enemy buildings.** Both units are successful at



Figure 28. Survivability Example of Flea Scout

Table 7. Summary of Scripted Scenarios - default vs. scout

	Scenarios					
scout	1.a	1.b	2.a	2.b	3.a	3.b
Jeffy	80%	80%	90%	90%	70%	70%
Flea	70%	80%	80%	90%	40%	30%

detecting the presence of enemy buildings while traversing through “The Pass.”

3. Avoid being killed or damaged:

- (a) **Evaluate the scout’s survivability by sending the unit near the enemy on a random path.** Jeffy scout is successful 80% of the time at being able to detect the enemy and avoid contact or being killed. Flea scout fails at avoiding contact, only surviving 40
- (b) **Evaluate the scout’s survivability by sending the unit near the enemy, knowing is a high risk to see if it can survive.** Jeffy scout is successful 90% of the time at being able to detect the enemy and avoid contact or being killed. Flea scout fails at avoiding contact, only surviving 30

Table 8. Summary of Scripted Scenarios - moea vs. scout

	Scenarios					
scout	1.a	1.b	2.a	2.b	3.a	3.b
Jeffy	80%	80%	90%	90%	70%	70%
Flea	70%	80%	70%	80%	40%	30%

Table 9. Summary of Scripted Scenarios - moea2 vs. scout

	Scenarios					
scout	1.a	1.b	2.a	2.b	3.a	3.b
Jeffy	80%	80%	80%	80%	70%	70%
Flea	70%	80%	70%	80%	40%	30%

Summary of the Scenarios.

Overall, the Jeffy scout is able to survive at a higher rate than the Flea scout while performing the three scouting strategies. Having identified which scout unit to utilize, it is then implemented with the MOSOP in the AFIT RTS AI platform.

Implementation of MOSOP in AFIT RTS AI platform.

The implementation of the MOSOP into the AFIT RTS AI platform resulted inconclusive. The unit is able to go to the first point in the selected “best,” but is currently unable to move to the next point. Furthermore, the scout unit is unable to run separate functionality when other units are in the line-of-sight of the enemy, thus generating additional computation of the MOSOP and assigning other points at a rapid change rate. Future work can add a separate unit manager for just the scout unit and move the MOSOP tactical option there so there is no conflict when other friendly units are in the line-of-sight with the enemy.

5.4 Chapter Summary

This chapter presents the results from the experiments detailed in Chapter 4 and discusses their analysis. It compares and contrasts the effectiveness and efficiency

of using the different MOEAs (NSGA-II, SPEA2 and NSPSO) to solve the MOSOP in the offline simulation. It shows that the NSGA-II was able to solve the MOSOP more effectively. This chapter also details the results of performance between the two scouting units being tested, concluding that the Jeffy scout unit had much greater survivability. This unit was then used for the integration of the MOSOP into the AFIT AI platform against other tactical options. However, this evaluation was inconclusive and showed that more work is required to effectively integrate the MOSOP.

VI. Conclusion and Final Remarks

6.1 Conclusion

Chapter 1 introduced the overall goal of this investigation and three objectives. The goal in this investigation is to study different scouting theories and methods used in RTS research, as well as develop and implement a new approach to scouting for the AFIT RTS AI.

The first objective is to design a multi-objective mathematical model representation of the scouting problem space. This is done by abstracting the scouting problem into three objective functions found in Chapter 3, thus creating the multi-objective scout orienteering problem. This completes this objective.

The second objective is to take the multi-objective scout orienteering problem model and evaluate it using multi-objective evolutionary algorithms. The results of this objective (Chapter 5) found that overall, NSGA-II proved to be a better algorithm to execute the problem domain. The offline simulation using PyGMO, then concluded that the multi-objective scouting problem is solvable through the use of MOEAs, thus completing this objective.

The third objective is to evaluate the online performance of two scripted scout units against other scripted attack and defense agents found within the AFIT RTS AI platform, and then use the selected scout unit to evaluate the integration of the MOSOP into the AFIT RTS AI platform. This online simulation concluded that the *Jeffy* scout is able to survive at a higher rate than the *Flea* scout while performing the three scripted scouting strategies and is a much better unit to use in the AFIT RTS AI platform, thus completing the first part of the objective. However, the second part of this objective is inconclusive as the MOSOP did not perform as intended. The scout unit is unable to run a separate functionality when other units are in the

enemy's line-of-sight, thus generating additional computation of the MOSOP and assigning other points at a rapid change rate. Future work may add a separate unit manager for just the scout unit, call it a scouting manager, and move the MOSOP tactical option there so there is no conflict when other friendly units are in the enemy's line-of-sight.

Additionally, the scope of this investigation did not allow for implementation of other approaches described in Chapter 2. Adding techniques such as potential flow [5] or next best view [19] to the scout unit and even the other combat units may increase the agent's survivability rate.

6.2 Future Work

The large complexity of RTS games allows for many other vectors future researchers can pursue:

- The scope and utility of a scouting unit is much greater than what this investigation covered. Future research can develop a new scouting manager, for full control the scout unit with better functionality.
- Adding techniques such as potential flow [5] or next best view [19] within the unit manager can increase the agent's survivability rate.
- Integrate the capability to change strategies to counter opponents, perhaps a separate strategy manager.
- Reduce calculation time of the tactical manager to reduce lag impact to human players

Each of these areas are important upgrades to the current performance of the agent, thus improving the platform.

6.3 Final Remarks

The experiments performed in this investigation expands the development of the Air Force Institute of Technology Real Time Strategy Artificial Intelligence platform. It contributes to the advancement of the overall goal of developing an artificial intelligence RTS planning platform for military critical decision-making education.

Appendix A. Results of MOEA Experiments

1.1 MOEA Experiment Results

This appendix contains graphical results from the MOEA experiments using the metrics established in Chapter 4. It is organized in order from lower parameter size to greater based on population size and generations.

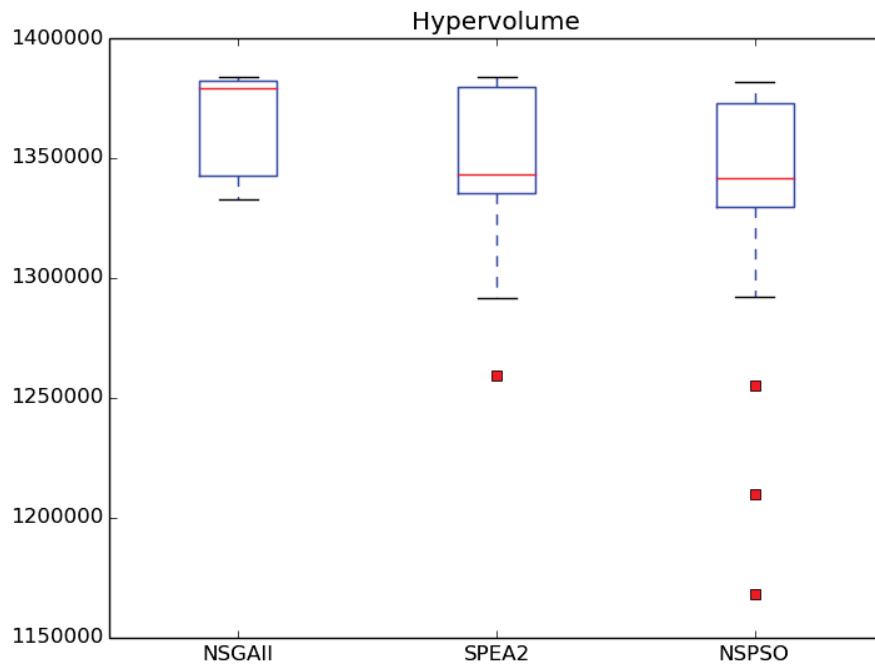


Figure 29. MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 2

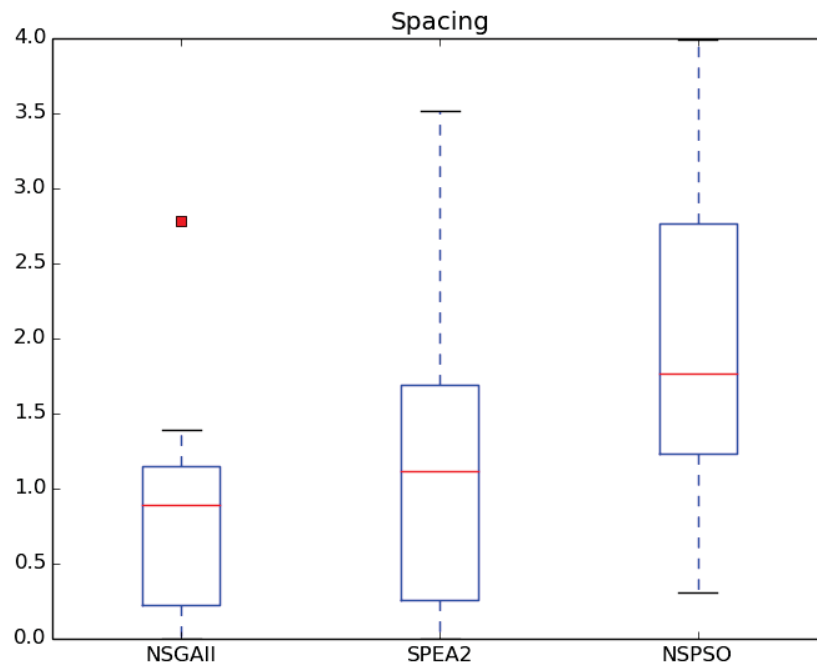


Figure 30. MOEAs Spacing Comparison Sample - Pop. 20 Gen. 2

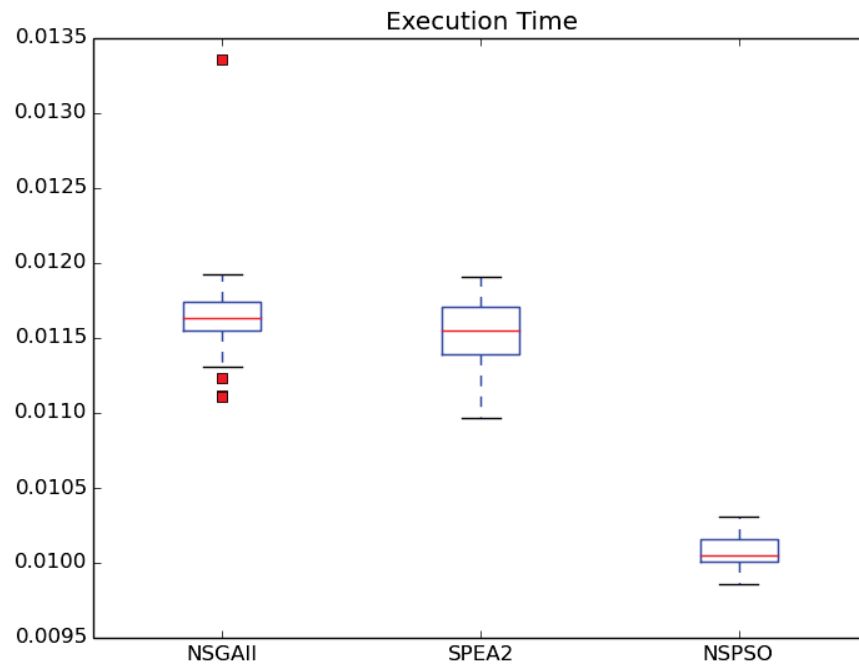


Figure 31. MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 2

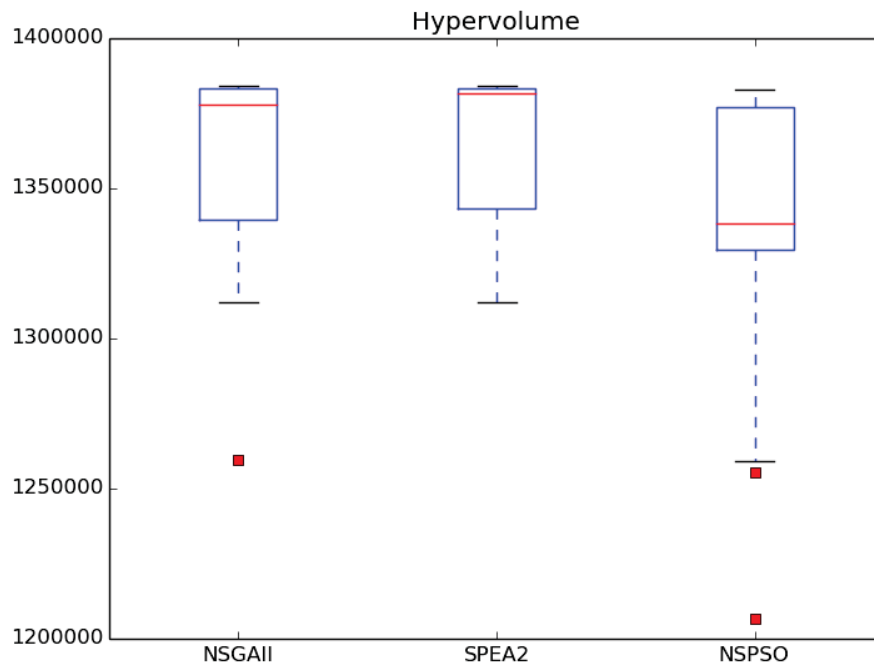


Figure 32. MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 3

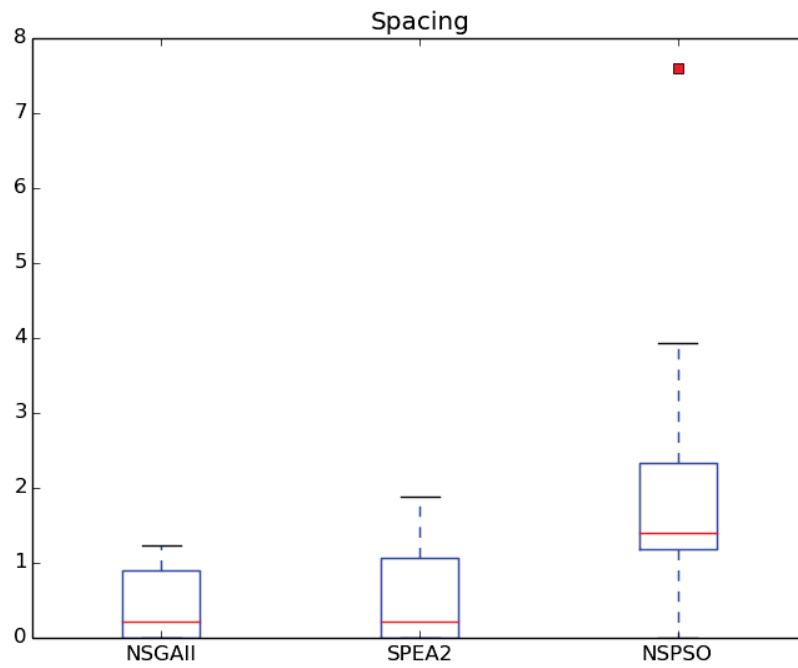


Figure 33. MOEAs Spacing Comparison Sample - Pop. 20 Gen. 3

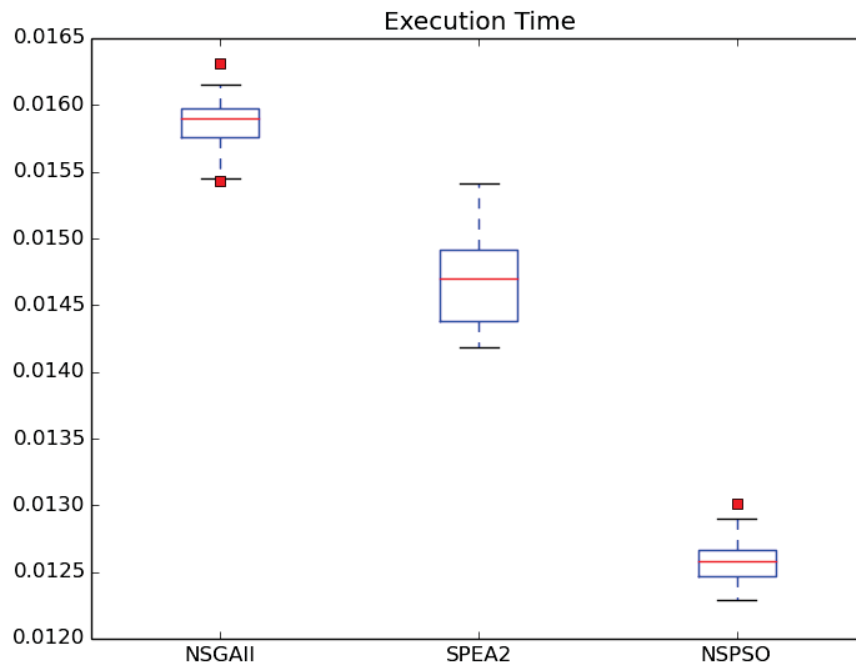


Figure 34. MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 3

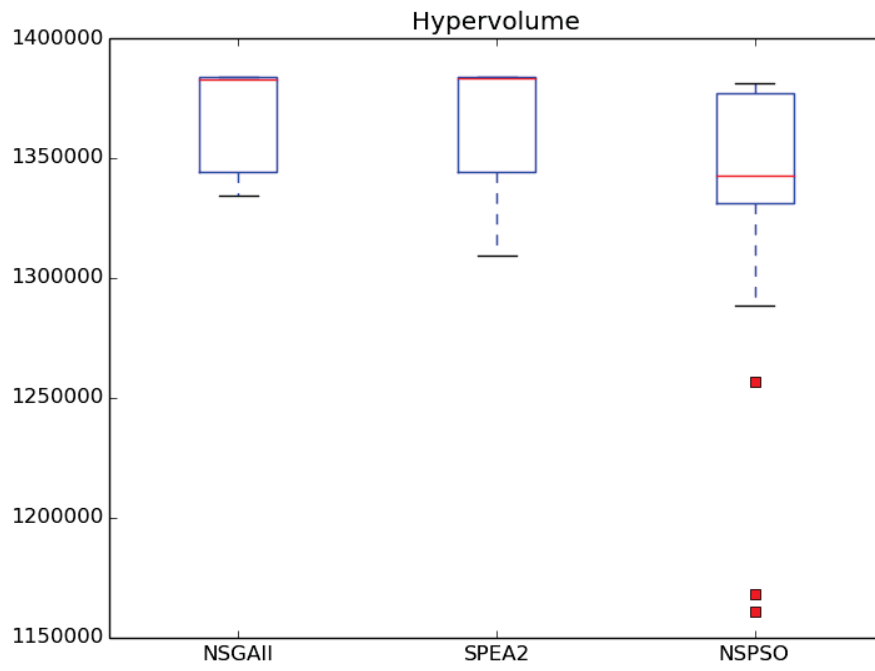


Figure 35. MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 4

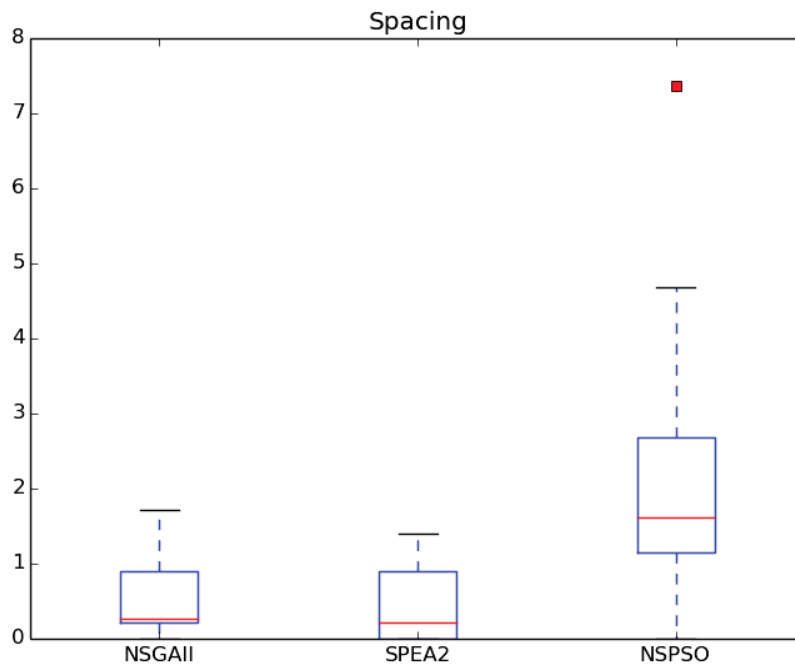


Figure 36. MOEAs Spacing Comparison Sample - Pop. 20 Gen. 4

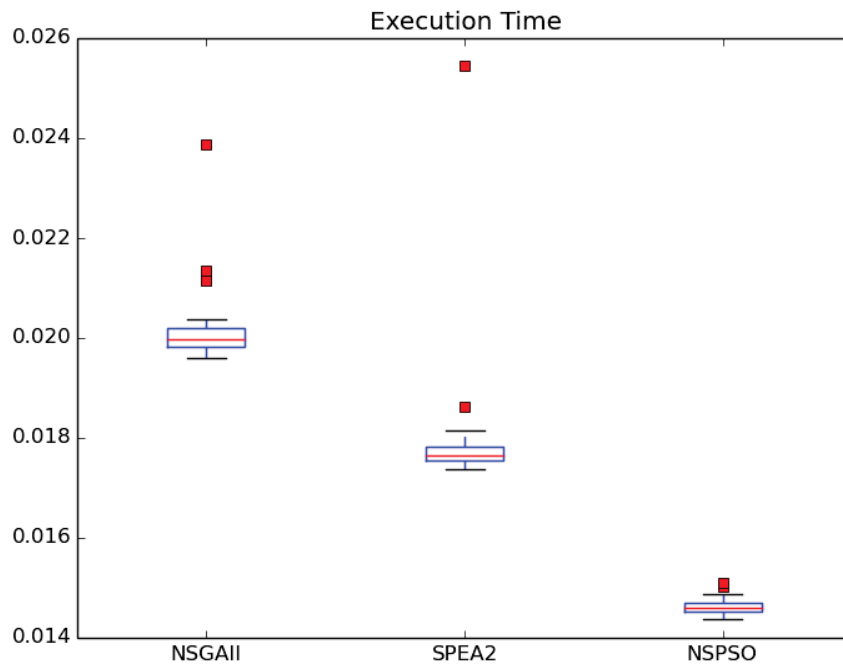


Figure 37. MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 4

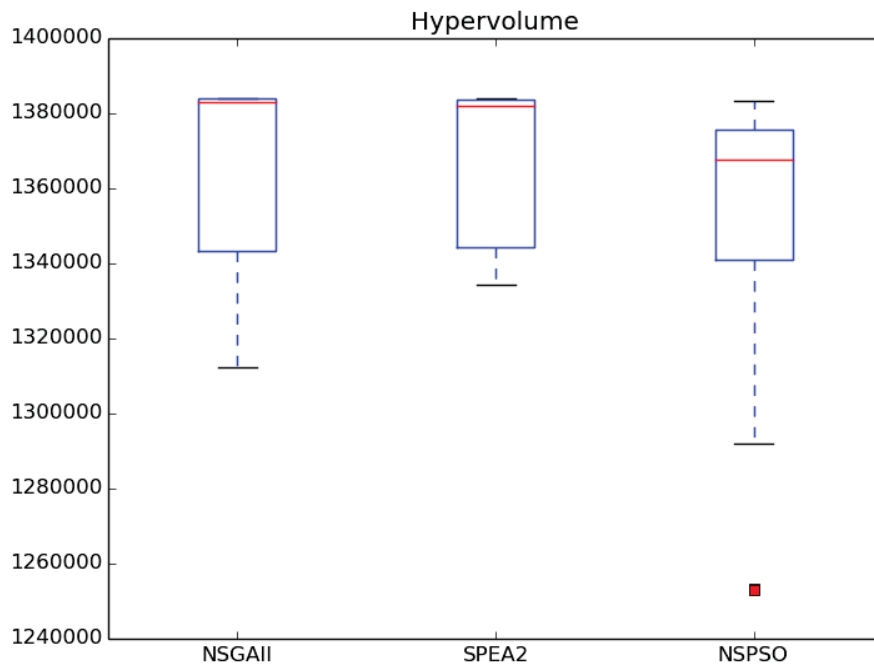


Figure 38. MOEAs Hypervolume Comparison Sample - Pop. 20 Gen. 5

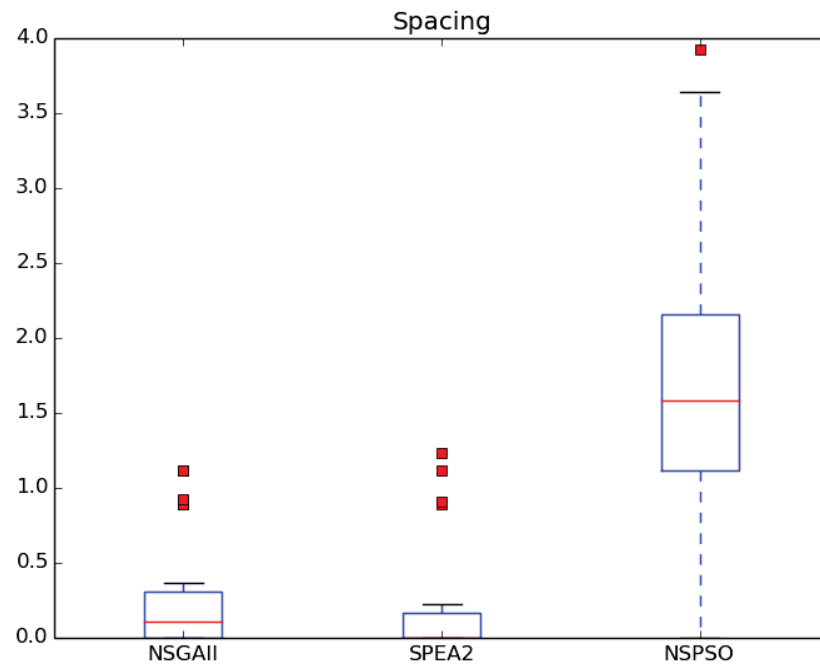


Figure 39. MOEAs Spacing Comparison Sample - Pop. 20 Gen. 5

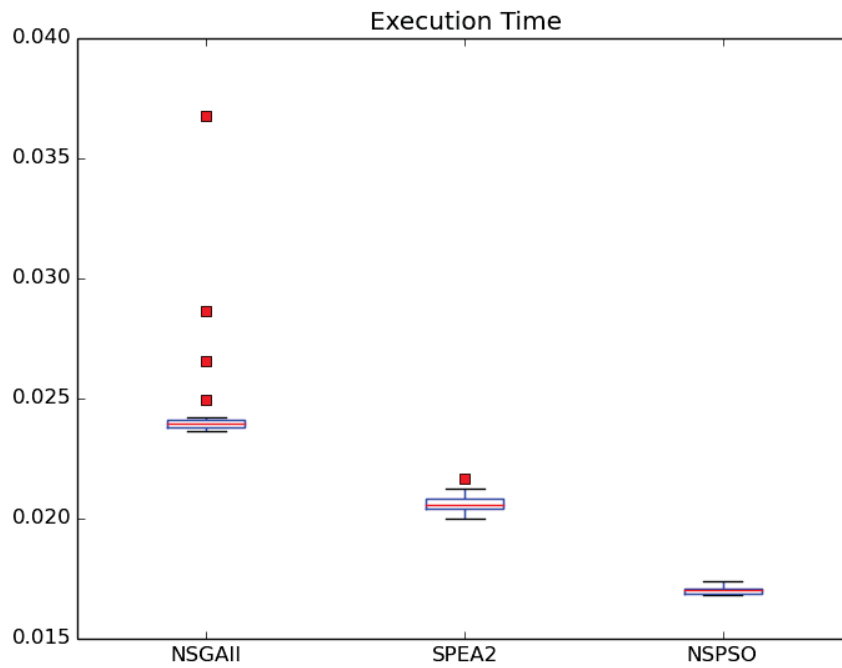


Figure 40. MOEAs Execution Time Comparison Sample - Pop. 20 Gen. 5

Bibliography

1. 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.
2. “Dune II - wikipedia, the free encyclopedia,” http://en.wikipedia.org/wiki/Dune_II, Accessed: December 29, 2016.
3. Cavedog Entertainment, “Total annihilation,” https://en.wikipedia.org/wiki/Total_Annihilation, Accessed: December 29, 2016.
4. Donald A Gruber, “Tactical ai in real time strategy games,” M.S. thesis, Air Force Institute of Technology, 2015.
5. Kien Quang Nguyen, Zhe Wang, and Ruck Thawonmas, “Potential flows for controlling scout units in starcraft,” *Proceedings of the 2012 IEEE International Conference on Computational Intelligence and Games*, 2012.
6. Simon Butler and Yiannis Demiris, “Partial observability during predictions of the opponent’s movements in an rts game,” *Proceedings of the 2010 IEEE International Conference on Computational Intelligence and Games*, 2010.
7. 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.
8. Christopher Ball, “A multi-objective approach to tactical maneuvering within real time strategy games,” M.S. thesis, Air Force Institute of Technology, 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. 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.
11. 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.
12. Glen Robertson and Ian Watson, “A review of real-time strategy game AI,” *AI Magazine*, vol. 35, no. 4, pp. 75–104, 2014.

13. “Real-time strategy - wikipedia, the free encyclopedia,” http://en.wikipedia.org/wiki/Real-time_strategy, Accessed: December 29, 2016.
14. Raúl Lara-Cabrera, Carlos Cotta, and Antonio J Fernández-Leiva, “A review of computational intelligence in RTS games,” in *Foundations of Computational Intelligence (FOCI), 2013 IEEE Symposium on*. IEEE, 2013, pp. 114–121.
15. “Decision making - wikipedia, the free encyclopedia,” <https://en.wikipedia.org/wiki/Decision-making>, Accessed: December 29, 2016.
16. Michael Buro, “Call for ai research in rts games,” *AAAI*, 2004.
17. United States Government US Army, *Field Manual 7-93: Long-Range Surveillance Unit Operations*, HQ, Department of the Army, 2009.
18. Carl Von Clausewitz, *On war*, Digireads.com Publishing, 2004.
19. C. Si, Y. Pisan, and C. T. Tan, “A scouting strategy for real-time strategy games,” *Association of Computing Machinery*, 2014.
20. Zhe Wang, Kien Q. Nguyen, Ruck Thawonmas, and Frank Rinaldo, “Adopting scouting and heuristics to improve the ai performace in starcraft,” *Innovations in Information and Communication Science and Technology*, 2013.
21. “Multi-objective optimization - wikipedia, the free encyclopedia,” http://en.wikipedia.org/wiki/Multi-objective_optimization, Accessed: December 29, 2016.
22. “Computer + video games 101 (apr 1990),” <https://wos.meulie.net/pub/sinclair/magazines/Computer-and-Video-Games/Issue101/Pages/CVG10100103.jpg>, Accessed: January 3, 2017.
23. Blizzard Entertainment, “Blizzard entertainment: Classic games,” <http://us.blizzard.com/en-us/games/legacy>, Accessed: December 29, 2016.
24. Blizzard Entertainment, “Starcraft,” <http://us.blizzard.com/en-us/games/sc/>, Accessed: December 29, 2016.
25. Hyunsoo Park and Ho-Chul Cho, “Prediction of early stage opponents strategy for starcraft ai using scouting and machine learning,” *Association of Computing Machinery*, 2012.
26. Blizzard Entertainment, “Starcraft ii world championship series,” 2016.
27. “Spring rts engine,” <http://springrts.com>, Accessed: December 29, 2016.
28. “Balanced annihilation - spring,” https://springrts.com/wiki/Balanced_Annihilation, Accessed: December 29, 2016.

29. "Wargus — home," www.wargus.sourceforge.net, Accessed: December 29, 2016.
30. "Sparcraft - starcraft combat simulation," <https://code.google.com/p/sparcraft/>, Accessed: December 29, 2016.
31. Stuart Russel, Peter Norvig, et al., "Artificial intelligence: A modern approach, 1995," *Cited on*, p. 20, 1994.
32. John E. Laird and Michael van Lent, "Human-level ais killer application interactive computer games," *AAAI*, 2001.
33. Michael Buro and Timothy Furtak, "Rts games and realtime ai research," *IJCA*, 2003.
34. Kurt Weissgerber, Brett Borghetti, and Gilbert Peterson, "An effective and efficient real time strategy agent," *Twenty-third Annual Florida Artificial Intelligence Research Society Conference*, 2010.
35. Jason Blackford and Gary Lamont, "The real-time strategy game multi-objective build order problem," *Proceedings of the Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014.
36. U.S. Army, *Army Field Manual FM (3-0) Unified Land Operations*, 2012.
37. Guillaume Chaslot, *Monte-carlo tree search*, Ph.D. thesis, Maastricht University, 2010.
38. Johan Hagelback and Stefan J. Johansson, "The rise of potential fields in real time strategy bots," *Association for the Advancement of Artificial Intelligence*, 2008.
39. "Scouting - liquidpedia, starcraft 2," <http://wiki.teamliquid.net/starcraft2/Scouting>, Accessed: December 29, 2016.
40. Ben G Weber, Michael Mateas, and Arnav Jhala, "A particle model for state estimation in real-time strategy games," *Proceedings of AIIDE*, 2011.
41. "Welcome to PyGMO," <http://esa.github.io/pygmo/>, Accessed: December 29, 2016.
42. "MOEA framework, a java library for multi-objective evolutionary algorithms," <http://www.moeaframework.org/>, Accessed: December 29, 2016.
43. "ParadisEO, paradiseo home page," <http://paradiseo.gforge.inria.fr/>, Accessed: December 29, 2016.
44. "ParadisEO, paradiseo documentation," <http://eodev.sourceforge.net/>, Accessed: December 29, 2016.

45. “jMetal web site,” <http://jmetal.sourceforge.net/>, Accessed: December 29, 2016.
46. 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.
47. Eckart Zitzler, Marco Laumanns, Lothar Thiele, Eckart Zitzler, Eckart Zitzler, Lothar Thiele, and Lothar Thiele, “SPEA2: Improving the strength pareto evolutionary algorithm,” 2001.
48. 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.
49. R. Kennedy, J.; Eberhart, “Partical swarm optimization,” in *Proceedings of IEEE International Conference on Neural Networks*. IEEE, 1995, pp. 1942–1948.
50. T. Tsilisigirides, “Heuristic methods applied to orienteering,” *Operational Research Society Ltd*, 1984.
51. Dominique Feillet, Pierre Dejax, and Michel Gendreau, “Travelling salesman problem with profits: An overview,” *Operational Research Peripatetic Postgraduate Programme*, 2001.
52. Bruce Golden, Larry Levy, and Rakesh Vohra, “The orienteering problem,” *Naval Research Logistics*, 1987.
53. Michael Schilde, Karl F. Doerner, and Richard F. Hartl, “Metaheuristics for the bi-objective orienteering problem,” *EUMEeting*, 2009.
54. “Spring engine maps,” <https://springrts.com/wiki/Maps>, Accessed: December 29, 2016.
55. Lpez-Ibez Paquete Vahrenhold Beume, Fonseca, “On the complexity of computing the hypervolume indicator,” *IEEE Transactions on Evolutionary Computation*, 2009.
56. Corne Knowels, “On metrics for comparing nondominated sets,” .
57. “15.3. time time access and conversions,” <https://docs.python.org/2/library/time.html>, Accessed: December 29, 2016.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 23-03-2017		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Sept 2015 — Mar 2017	
4. TITLE AND SUBTITLE Scouting in Real-Time Strategy Games: Theory, Methods and Implementation				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Ortiz, Rosemberg, 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-17-M-056	
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) game environment is a strong platform for simulating complex strategical and tactical problems. The overall goal of this research is to develop artificial intelligence (AI) RTS planning platform for military critical decision-making education. The agent should have the ability to perform at an expert level as well as to challenge and assess a players critical decision-making ability or skill-level. The nature of time sensitivity within the RTS environment creates very complex situations. Each situation must be analyzed, and commands must be given to each tactical unit before the scenario on the battlefield changes and makes the decisions less relevant. This particular investigation effort of Air Force Institute of Technology (AFIT) RTS AI platform, focuses on constructing a novel approach to scouting within the RTS environment using multiobjective evolutionary algorithms (MOEAs) for finding a "best" scouting solution.					
15. SUBJECT TERMS RTS, Tactics, MOEA, Scouting					
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	97	19b. TELEPHONE NUMBER (include area code) (937) 255-3636, x4718; gary.lamont@afit.edu