6-1-2019

Methodology for Comparison of Algorithms for Real-World Multi-objective Optimization Problems: Space Surveillance Network Design

Troy B. Dontigney

METHODOLOGY FOR COMPARISON OF ALGORITHMS FOR REAL-WORLD MULTI-OBJECTIVE OPTIMIZATION PROBLEMS: SPACE SURVEILLANCE NETWORK DESIGN

THESIS

Troy B. Dontigney, MSgt, USAF

AFIT-ENG-MS-19-J-003

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-19-J-003

METHODOLOGY FOR COMPARISON OF ALGORITHMS FOR REAL-WORLD MULTI-OBJECTIVE OPTIMIZATION PROBLEMS: SPACE SURVEILLANCE NETWORK DESIGN

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 Electrical Engineering

Troy B. Dontigney, BS

MSgt, USAF

June 2019

AFIT-ENG-MS-19-J-003

METHODOLOGY FOR COMPARISON OF ALGORITHMS FOR REAL-WORLD MULTI-OBJECTIVE OPTIMIZATION PROBLEMS: SPACE SURVEILLANCE NETWORK DESIGN

THESIS

Troy B. Dontigney, BS
MSgt, USAF

Committee Membership:

Dr. Laurence D. Merkle
Chair

Dr. Richard G. Cobb
Member

Dr. John M. Colombi
Member

Dr. Gary B. Lamont
Member

AFIT-ENG-MS-19-J-003

Abstract

Space Situational Awareness (SSA) is an activity vital to protecting national and commercial satellites from damage or destruction due to collisions. Recent research has demonstrated a methodology using evolutionary algorithms (EAs) which is intended to develop near-optimal Space Surveillance Network (SSN) architectures in the sense of low cost, low latency, and high resolution. That research is extended here by (1) developing and applying a methodology to compare the performance of two ore more algorithms against this problem, and (2) analyzing the effects of using reduced data sets in those searches. Computational experiments are presented in which the performance of five multi-objective search algorithms are compared to one another using four binary comparison methods, each quantifying the relationship between two solution sets in different ways. Relative rankings reveal strengths and weaknesses of evaluated algorithms empowering researchers to select the best algorithm for their specific needs. The use of reduced data sets is shown to be useful for producing relative rankings of algorithms that are representative of rankings produced using the full set.

Acknowledgements

The beginning of wisdom is this: Get wisdom. – Proverbs 4:7

I would like to thank my advisor and the members of my committee for their help and support throughout this process. They each showed a real knack for asking the right questions and producing the right nuggets of wisdom to keep me moving in the right direction or, as was often the case, to get me off of whatever rabbit trail I'd wandered down. I'd also like to thank Mr. David Meyer for his invaluable assistance with STK. His guidance and insight were instrumental in making the hardest part of my research into a real success.

Above all, I would like to thank my wife and daughter for putting life on hold to support me through my time at AFIT. From patiently listening to my rants to understanding all the times I couldn't be there, even though I wanted to be, they were real troopers. They are the reason I could do what I did.

Troy B. Dontigney

Table of Contents

Page

Page

List of Figures

List of Tables

List of Abbreviations

ϵ-NSGAII	Epsilon Non-Dominated Sorted Genetic Algorithm II
ACT	Advanced Concepts Team
AER	Azimuth, Elevation, and Range
AFRL	Air Force Research Laboratory
API	Application Programming Interface
COST	Common Open Source Tools
DM	Decision Maker
DTID	Detect, Track, Identify
ES	Evolution Strategies
ESA	European Space Agency
GB	Gigabyte
GBT	Ground-Based Telescope
GEO	Geostationary Earth Orbit
GPU	Graphical Processing Unit
HDD	Hard Disk Drive
HPC	High-Performance Computer
HV	Hypervolume
IBEA	Indicator-Based Evolutionary Algorithm
LEO	Low Earth Orbit
MMOTG	Mean Maximum Observation Time Gap

MOEA Multi-Objective Evolutationary Algorithm

MOEA/D Multi-Objective Evolutionary Algorithm based on Decomposition

MOGA Mutation Only Genetic Algorithm

MOO Multi-objective Optimization

MOP Multi-objective Problem

MPI Message Passing Interface

NSGA-II Non-dominated Sorted Genetic Algorithm II

NSGA-III Non-dominated Sorted Genetic Algorithm III

OM Object Model

PAES Pareto Archived Evolution Strategy

PBS Portable Batch System

PFLOPS Petaflops

RAM Random-Access Memory

RNG Random Number Generation

RRHC Random Restart Hill Climber

RS Random Search

RSO Resident Space Object

SPEA2 Strength Pareto Evolutionary Algorithm

SSA Space Situational Awareness

SSD Solid State Drive

SSN Space Surveillance Network

STK Systems Toolkit

TB Terabyte

TW&A Threat Warning and Assessment

USG United States Government

METHODOLOGY FOR COMPARISON OF ALGORITHMS FOR REAL-WORLD MULTI-OBJECTIVE OPTIMIZATION PROBLEMS: SPACE SURVEILLANCE NETWORK DESIGN

I. Introduction

1.1 Background

When humanity first developed spaceflight capabilities, little attention was given to the issue of congestion. Space was a seemingly infinite frontier containing a mere handful of artificial objects. In the intervening decades, much has changed. Technology has progressed at an astonishing rate, and the domain that was once in reach only for global superpowers is now more accessible than ever before. In particular, with the rise of miniaturization, technologies such as CubeSats [1] have opened space to much wider use. What once required tremendous government resources is now achievable by commercial entities and even high schools [2]. Concurrently, space has become an indispensable domain for the United States Government (USG) and military, enabling a myriad of capabilities such as communication, surveillance, reconnaissance, navigation, and weather forecasting at levels that would not be otherwise possible [3]. In particular, with its 24-hour orbital period, the Geostationary Earth Orbit (GEO) regime allows satellites to loiter over particular points on the surface of the Earth [4], and has proven to be a particularly valuable orbital regime for the USG. However, as the number of nations and industries dependent on space increase, the once untapped domain of space is becoming ever-more congested, competitive, and contested [3].

1.2 Problem Statement

Space Situational Awareness (SSA).

Congestion in the exosphere is a growing problem. With the total number of Resident Space Object (RSO) orbiting the Earth projected to multiply several times over in the near future [5, 6, 7], including the associated debris from each launch, the risk of accidental conjunctions and, therefore, the need for effective Space Situational Awareness (SSA) will continue to increase. Current SSA capabilities are already overburdened [8]. RSOs in GEO are not observed often enough nor with sufficient resolution to detect hazardous conditions, accidental or hostile, in a reliably timely manner.

In response to this problem, Stern and Wachtel [9] developed a minimization approach intended to find a selection of Space Surveillance Network (SSN) architectures that explore the trade space of low cost, low latency, and small detection size (high sensitivity). In both their research and the present effort, an SSN is defined to be a collection of optical telescopes each of which can be either ground-based at one of nine locations or in orbit around the Earth in one of three orbital regimes.

In their work, the SSN architecture design problem is framed as a multi-objective minimization problem with three objectives: minimize cost, latency, and detection size of candidate SSN architectures [9]. The cost is the approximate expense to build, deploy, and operate the collected telescopes within the architecture, and is primarily driven by the numbers, sizes, and deployment domains of telescopes. Latency describes the average elapsed time between observations an architecture can be expected to provide, which mainly depends on the number of telescopes and the number of RSOs to be observed. Detection size is the smallest object the architecture can be expected to be able to detect which, though dependent on many factors, is most impacted by the aperture size of the telescopes in the architecture and, to a lesser

degree, the number and distribution of RSOs across the sky.

These objectives are not independent, meaning that changes made to improve one objective will worsen another. Consider an effort to optimize arbitrary architecture. If one attempts to improve (reduce) overall cost by reducing the number of telescopes, latency will suffer as there are fewer telescopes to observe the same number of RSOs. Improving cost by reducing the aperture size of all telescopes results in a worsening of detection size. Conversely, if one increases the size of apertures to improve detection size, or increases the number of telescopes to improve latency, cost increases. This phenomenon is demonstrated using specific architectures in Appendix A.

Computational Cost.

The search space considered for this problem is enormous, with 2.428×10^{21} possible architectures in Stern and Wachtel's underlying model [9]. Compounding the problem, Stern and Wachtel's objective functions, as implemented in this research, result in an average evaluation time of 303.2 seconds for a single architecture on the Mustang High-Performance Computer (HPC) at the Air Force Research Laboratory (AFRL) (Section 3.6 summarizes Mustang's technical specifications).[1] This means that an exhaustive search would require approximately 23.3×10^{15} years of CPU time using technology available today. In Stern and Wachtel's work, the optimization method relied on an evolutionary algorithm (EA) to perform a non-exhaustive search, producing a set of near-optimal solutions in a more reasonable time. *Search algorithms are not all created equally*, however, and each will perform better on some problems and worse on others [11]. A problem of this importance should be evaluated with a number of appropriate algorithms to determine which algorithm(s) are most effective and efficient.

[1]Stern and Wachtel reported an average evaluation time of "30 to 40 minutes" per generation on AFRL's decommissioned Spirit HPC [10]. Each individual in a generation is evaluated in parallel under their methodology.

The high computational cost is one barrier to performing such a comparison among algorithms. Furthermore, the computational cost of this problem grows with the number of RSOs simulated. In contrast with the 303.2 second average evaluation for the current implementation of Stern and Wachtel's model, which simulates 813 RSOs, preliminary computational experiments in this research effort simulating only 20 RSOs reduced that to an average of 8.3 seconds. Overall, those early trials demonstrated that computational costs increase superlinearly as the number of RSOs increases. This was not surprising because each evaluation of a candidate architecture must process each 30 second observation interval in the 24-hour simulation period for each sensor/target pair, accumulating results as it goes, before determining final values for the entire architecture. Any inefficiency in computations, such as delasy related to managing the larger memory footprint, would push this otherwise linear growth into the superlinear range. Reducing the number of RSOs is a simple way to reduce the computational cost of evaluations, i.e. to improve algorithmic efficiency, but the impact of this reduction on algorithmic effectiveness is not obvious *a priori*.

> *Problem Statement - Determine the relative ranking of search algorithms, with respect to solution quality, when evaluating a high-dimensional SSN model.*

1.3 Research Objectives, Questions, and Hypotheses

As discussed above, Stern and Wachtel's model has not been rigorously tested to determine the effectiveness of alternate optimization algorithms[2]. This is presumably due in part to the high computational cost of their implementation precluding the possibility of performing the necessary computational experiments.

[2]Effectiveness can have a number of meanings. In a problem such as this, where the optimization is not done in real-time, it refers mainly to the ability to obtain the best solutions with respect to the objective functions across a large portion of the search space (quality and diversity). However, in other situations it could also consider computational cost and other time/speed measures.

These circumstances motivate the following research objectives.

RO1 The main objective of this research is to develop a methodology to efficiently determine the most effective among a set of candidate search algorithms for the SSN architecture design problem.

RO2 Since this necessarily involves comparing the results produced by the various algorithms, a supporting objective is to assess the value of various methods of performing those comparisons.

RO3 Finally, since the first objective seeks an efficient methodology, another supporting objective is to assess the viability of performing the algorithm comparisons at reduced computational cost by reducing the number of RSOs simulated.

The steps taken in this effort toward satisfying these research objectives are guided by three research questions that lead to four testable hypotheses.

Research Questions.

By applying a representative selection of classical and evolutionary search algorithms to the SSN optimization problem, and employing four binary comparison methods to compare the output of those algorithms, this research addresses the following questions:

Q1 Which of the representative algorithms is (are) most effective?

Q2 What useful insights are provided by various means of comparing the results of the algorithms?

Q3 What is the impact of using fewer simulated RSOs on the quality of solutions produced by these algorithms?

There are, of course, myriad classical search algorithms and evolution algorithm variants. The scope of this research is limited to random search, random restart hill climbing, and three multi-objective evolutionary algorithm variants, which are discussed in greater detail in Section 3.4.

Hypotheses.

These questions suggest some testable hypotheses, several of which have multiple implicit hypotheses due to the use of multiple comparison methods. Where the terms better or worse are used, they refer to the numeric results of those binary comparisons.

H1 For each pair of algorithms, one will tend to produce better Pareto fronts than the other.

H2 The Pareto fronts produced by random search tend to be worse than those of all remaining algorithms.

H3 Each evolutionary algorithm tends to produce better Pareto fronts than a random-restart hill climber.

H4 Simulating fewer randomly selected RSOs does not tend to change the relative effectiveness of the algorithms.

1.4 Methodology

These research questions and hypotheses are addressed through two series of computational experiments. The first series, which addresses research questions Q1 and Q2 and hypotheses H1 through H3, compares the solution sets produced by various multi-objective search algorithms when applied to the problem of optimizing SSN architectures. Because this is a real world problem, there is no known Pareto Optimal Set against which to compare results. Therefore, the results of the search algorithms

being evaluated can only be compared to one another to produce an overall ranking. Several binary[3] comparison methods can be employed to quantify the differences between two solution sets. Four such binary comparison methods are used in this research: binary hypervolume, coverage, ϵ-indicator, and additive ϵ-indicator (see Section 2.5).

The second series of experiments, which addresses research questions Q2 and Q3 and hypothesis H4, examines the impact on algorithmic effectiveness of reducing the computational cost of the search algorithms by simulating fewer randomly selected RSOs. The direct relationship between computational cost and the number of RSOs simulated strongly motivates the desire to understand the impact of using fewer RSOs on the quality of solutions produced by search algorithms, relative to the results produced using a full complement of RSOs. If the relationship is predictable, many algorithms may be evaluated at a much lower computational cost than is currently possible.

This research combines three major concepts: the Stern and Wachtel space surveillance network architecture model [9], search algorithms, and Pareto front comparison methods. The model produces executable architectures[4] defined by 28 dimensions that define quantity, size, and location of telescopes on the ground or in one of three orbits. It evaluates them using pre-simulated data and three objectives: cost, Mean Maximum Observation Time Gap (MMOTG) (also known as latency), and minimum detectable object size (detection size).

Each of the various search algorithms employs a different approach to guide its exploration of the search space through several thousand evaluated architectures.

[3]Binary, in this usage, refers to the comparison of two items, not the numbering system. This is in contrast to unary indicators, which characterize a single solution set in isolation (e.g. the hypervolume indicator) [12].

[4]An executable architecture is an architecture that includes enough detailed information that it can be implemented automatically or semi-automatically [13]. They are useful when using simulations to analyze systems for emergent properties.

These algorithms are run several times each, accelerated by running them in parallel using AFRL's HPC facilities [14]. In addition to running the algorithms using the full simulation data, they are also run repeatedly using smaller sets of simulation data, which simulate several reduced sets of RSOs. There are five sets of simulation data used with 813, 407, 203, 81, and 20 RSOs (100%, 50%, 25%, 10%, and 2.5%, respectively). Search does not alter the simulation data data, so the five data sets are only simulated once and reused with each of the search algorithms. Regardless of which algorithm is used, the concept of Pareto domination is used to select a final set of architectures that present optimal compromises among the three objectives. Thus, each algorithm run produces a set of solutions.

The repeated runs of each algorithm are compared pairwise using four different comparison methods to assess the relative reliability of the algorithms as well as the effects of using smaller data sets. The results of the runs are also aggregated for each algorithm and compared with those of the other algorithms to assess the relative effectiveness of the algorithms against this specific problem.

1.5 Assumptions and Limitations

This research assumes that the Stern and Wachtel model [9] accurately predicts the cost, latency, and detection size of SSN architectures; no effort is made here to modify or improve its function beyond those changes necessary to facilitate the execution of the computational experiments. All assumptions of the model and any inaccuracies in its predictions remain. Assessment of the accuracy of the model and enhancements to address any inaccuracies remain areas for future research.

It is also assumed that there is an external Decision Maker (DM) whose preferences are not known, meaning that no weighting scheme is applied when evaluating candidate solutions. This rules out the use of single-objective optimization algorithms

applied to a linear combination of the multiple objectives. Identification of candidate preference schemes and incorporation into the search process could result in software tools with greater real-world utility, but this is also left as an area for future research.

Finally, it is assumed that this is a real-world problem, and that there is no prior knowledge of the search landscape to guide algorithm selection. Relaxation of this assumption and incorporation of domain knowledge in the optimization algorithm could lead to improved effectiveness, but this is left as yet another area for future research.

This research is primarily limited by available computing power. Even using HPCs, the combination of computational cost, the number of runs required of each algorithm, and the use of multiple data sets limits the total number of algorithms to five. Also, there is a self-imposed limitation to working with a single library of algorithms. Doing so eliminates concerns about varying skill levels among different programmers, and it also enables the implementation of supporting software tools that will be easier for future researchers to understand and modify than would be possible if multiple libraries were used.

1.6 Implications

This research represents a first step in, and a methodology for, identifying algorithms that are more effective against the SSN architecture search problem. Stern and Wachtel's work [9] makes it clear that building a SSN that will meet our needs is likely to cost billions of dollars and require years of planning. Given that the most appropriate algorithms for this problem do not guarantee truly optimal solutions, it is important to ensure that the most effective algorithm is being applied before choosing an architecture for such a large undertaking.

This research also provides insight into the uses and limitations of using reduced

data sets with problems relating to GEO and related large-scale simulation-based optimizations. In the search for the most effective algorithm, many algorithms will need to be applied to this problem. Given that the search space considered is so large that a modern computer could not search even 1% of the search space in the Sun's remaining lifetime, finding effective ways to reduce the computational load of this problem is extremely important.

1.7 Organization

Chapter II discusses the basics of SSA, EAs, and binary comparison methods. Brief descriptions of common multi-objective algorithms are included, as is a summary of Stern and Wachtel's work.

In Chapter III, the methodology used to answer the research questions is described. This includes descriptions of computational experiments to test the hypotheses and some discussion of obstacles that affect the methodology.

Chapter IV presents results obtained from the comparisons and the analysis of the data. These results include how each algorithm performs relative to the others, as well as the impact of using reduced data sets on the results of each algorithm. Optimal SSN architectures are not presented, except where appropriate to elaborate on the results.

Finally, Chapter V presents conclusions and makes recommendations for future, related work.

II. Literature Review

2.1 Chapter Overview

The purpose of this chapter is to define concepts and terminology relevant to this research, describe the current state of SSA and recent research in the field, and to describe the tools that are used to perform the computational experiments described in Chapter III. The goal is to highlight areas where efficiencies are likely to be obtainable and where further study is required. The chapter begins with definitions of terminology, algorithms, and tools that relate to the computational aspect of the problem. Next, terminology and concepts relating to SSA are discussed. Finally, the specific SSN model to be used in this research is introduced.

2.2 General Definitions

Before delving into detailed discussions of the major areas of this research, brief summaries are offered for easy reference throughout the thesis. It is not a substitute for the more thorough discussions immediately following this section, but serves to be more descriptive than the list of acronyms at the beginning of the document.

High-Performance Computing (HPC), refers to the general concept of using one or more nodes within a cluster of high-end computers that is purpose-built to handle heavy computational loads. The use of HPC is not significant to the outcomes of this research, and is simply a tool to offload the most computationally expensive portions of the work in order to receive the highest-quality results possible within the available time.

Evolutionary Algorithms (EAs) can refer to a broad category of algorithms, encompassing subcategories such as Genetic Algorithms (GA) [15, 16], Evolutionary Programming (EP) [17], and Evolution Strategies (ES) [18], in which evolution-

ary concepts are used to guide the "evolution" of progressively better sets of solutions [12, 19]. As the specific classes of EA are not a central topic to the research, "EA" is used in a general sense to indicate any such algorithm except in places where a more specific definition is appropriate.

Multi-objective Optimization (MOO) is a special type of optimization in which a problem has multiple objectives and, in most cases, some of the objectives are in direct opposition to one another. That is, when variables are adjusted to improve one objective value, it comes at the cost of another objective value becoming worse. Multi-objective algorithms normally return a set of possible solutions, each representing some optimal compromise between objectives, referred to as a Pareto Front [12].

The term Space Surveillance Network (SSN) typically refers to the collection of radar and optical telescope sites maintained by the US Air Force for the purpose of monitoring objects in orbit around the Earth [20]. For this research, SSN is used more genearlly to describe a collection of ground- or space-based optical telescopes tasked to monitor objects in GEO. Other types of sensors or orbital regimes are not considered here.

2.3 High Performance Computing (HPC)

HPC is a tool that is used heavily to accommodate the significant computational burden of this research effort. Note that the acronym HPC may refer to either "high performance computing" or "high performance computer," and is used in both ways throughout this document. HPC platforms are clusters of server-grade computers that can be used to tackle computational problems that are too large for a traditional workstation. While they can be small clusters of just a few computers used for business purposes, clusters with thousands of machines and tens of thousands of cores are the norm for scientific and government purposes. Unlike traditional computing,

focusing on performing small tasks in a generally serial manner, HPCs are designed with highly parallel performance in mind. Use cases for an HPC typically fall into two types of jobs: running many instances of a serial task simultaneously, or distributing one or more highly-parallelizable task across many nodes.

HPCs consist of a collection of servers interconnected with a specialized, high-speed network. Individual nodes may have dozens of cores, and hundreds of gigabytes of system memory [21]. There may also be multiple classes of machine available on a cluster, with most falling into a standard category, a handful offering much higher system memory, and another small collection of nodes with specialized coprocessors, such as a Graphical Processing Unit (GPU).

While the allure of an HPC can be great, there are distinct pros and cons associated with using HPCs which must be considered before electing to make use of them in any research effort. Advantages mainly center on the enormous pool of computational resources and the careful tuning of the architecture for parallel computation. These advantages offer the promise of faster execution for certain tasks, and the possibility of tackling problems that would be otherwise intractable. Many of the disadvantages also center around the parallel nature of the system: writing programs to properly capitalize on the architecture can be more difficult than with traditional computers, memory management is more complicated, problems that cannot be decomposed into enough sub-tasks will not enjoy meaningful speedup, and debugging can be slower and more difficult than with dedicated, traditional hardware. Additionally, HPCs are almost always shared resources, with many users competing for processing time, so delays while a job waits in the queue can be both significant and unpredictable as they are influenced by many dynamic factors. Finally, virtually all HPCs are somewhat custom orders, and there can be significant variation in software and hardware employed from one HPC to the next, making portability a challenge if one wishes to

migrate their work, or to extend previous work on a different platform.

2.4 Evolutionary Algorithms (EAs)

Evolutionary algorithms are a broad class of algorithm that draw some inspiration from the theory of evolution. These algorithms operate on populations of candidate solutions, and apply some combination of evolutionary operators (typically categorized as either selection or variation operators) to evolve progressively more fit generations of solutions [19]. In terms of an EA, fitness is defined in terms of the objectives of the optimization problem, so a minimization problem's population becomes more fit as the overall objective values trend downward, and less fit as they trend upward. An individual is simply one possible solution to the problem, represented as a combination of valid characteristics, often called a decision variable vector, that can be mapped to specific values for each of the objectives, called an objective vector [22]. The population is a collection of individuals against which evolutionary processes can be imposed. Every individual is evaluated for fitness using a fitness function, which mathematically evaluates the individual against the objective(s) of the problem. In the case of a minimization problem, individuals that have a lower value for some objective function are deemed more fit than those with higher values. The search ends based on some termination criteria, typically either based on the number of candidates evaluated or in the overall fitness of the population, and the best solution(s) found are output to the user.

Bio-inspired Terminology.

As EAs borrow heavily from biology, some terminology is also borrowed. In biology, two ways to describe an organism are genotypically or pheontypically. "Genotypic" refers to the genetic makeup of an organism, while "phenotypic" refers to the

outward appearance or traits of the organism. Roughly speaking, genetic data is encoded in chromosomes and decomposed into specific factors, genes, which can take on any one of several possible values, known as alleles. Depending on the value of one or more specific genes, different phenotypes (observable traits) may result.

Similarly, EAs are discussed in terms of genes and alleles in both genotye and phenotype, with the addition of objective values [19]. Genotypic space, also called decision space, is a k-dimensional space, where k is the number of factors that can be controlled (i.e. genes), representing all of the feasible combinations of factors relevant to the problem. Again, phenotype is the outward expression of the genotype, which is specific to the problem. Objective space is an n-dimensional space, where n is the number of objectives for the problem. It is important to note that there is no requirement for k and n to be equal, and a typical problem difficult enough to warrant an EA will have complex interactions between multiple genes for some (or all) of the objectives. It is common to see problems with more genes than objectives.

Selection Operators.

Selection operators are routinely used in two ways. First, they can be used to select parents from a population from which to create offspring (selecting individuals to mate). Second, they can be used to select replacements or successors to make up the next generation (selecting individuals to survive). Selection operators can be implemented in a number of ways, but always use fitness as a driving factor, selecting the more fit individuals more often than the less fit ones. This is normally accomplished using a probabilistic approach which assigns a higher probability of selection to more fit individuals without completely precluding the possibility of a less fit individual's selection [19]. The idea is that there may be some good genes hidden in less fit individuals that could be beneficial to subsequent generations when

combined with good genes from other individuals. Though these operators would seem to work hand in hand, algorithms do not always implement both forms of selection.

Variation operators.

Variation operators are used to introduce and maintain diversity in the population [19]. The two main variation operators are recombination and mutation. Mutation operates on an individual, making random changes to its genotype, while recombination, sometimes called crossover, operates on two or more "parents" combining their genotypes to make one or more "children" [12]. These operators work to introduce alleles that were not present in the population and to produce new combinations of existing genes, respectively, aiding in the exploration of the decision space.

It is most common to see both operators used, but, as with selection operators, it is possible to implement algorithms which do not use both. For example, the Mutation Only Genetic Algorithm (MOGA) [23] and Evolution Strategies (ES) [18] rely solely on mutation to introduce variation, and therefore do not implement parent selection or recombination operators. It is important to note that the method of recombination or mutation used depends on the genotypic data representation. For example, changing a binary bit from a 0 to a 1 or from a 1 to a 0, called bit flipping, should normally only be used if the genotype is represented as a string of binary bits. If the data is represented as real numbers, then some form of random number generation (RNG) would be required, instead. Likewise, if allele values are pulled from a discrete list of values, then neither bit flipping nor RNG would directly apply. The effectiveness of specific variation operators is not a focus of this research effort, so the discussions of relevant algorithms later in this chapter do not describe the specific types of variation operators. Rather, they state only whether they are used or not.

Dominance and Pareto Optimality.

Dominance, a relation on the set of solutions and denoted by $A \prec B$ for a minimization problem, is an important concept for MOO used to determine which solutions are "better" than others. Simply put, given two solutions to a problem, A and B, A dominates B if all of its objective values are at least as good as B's corresponding objective values, and at least one of its objective values is better than B's corresponding value [19]. In the case of minimization, all objective values in A must be less than or equal to the corresponding objective values in B, and at least one value must be strictly lesser than B's. It is formally expressed as follows:

$$A \prec B \iff \forall i \in \{1, ..., n\} \, | a_i \leq b_i \quad \cap \quad \exists i \in \{1, ..., n\} \, | a_i < b_i \tag{1}$$

Additionally, there are different "strengths" of dominance which are summarized in [24]. Weak dominance, denoted by $A \preceq B$, is a relaxed form of dominance in which we only say that A is equal to or better than B in all objectives. This is simply dominance which allows for the special case where A is equal to B in all objectives. Strong or strict dominance, denoted by $A \lessdot B$, indicates a case where A is better than B in all objectives. Strong (or strict) dominance implies dominance, and dominance implies weak dominance. Finally, A and B are said to be incomparable ($A||B$) when neither A weakly dominates B, nor B weakly dominates A. Figure 1 illustrates these concepts in a 2-objective minimization problem.

Pareto optimality, which is an idea borrowed from economics [25], describes a distribution of resources in which no reallocation can be accomplished except at the detriment of one of the individuals to whom resources are distributed. In MOO, the concept is adapted to describe a solution A for which there exists no solution B that improves one or more objective values without worsening another, with respect

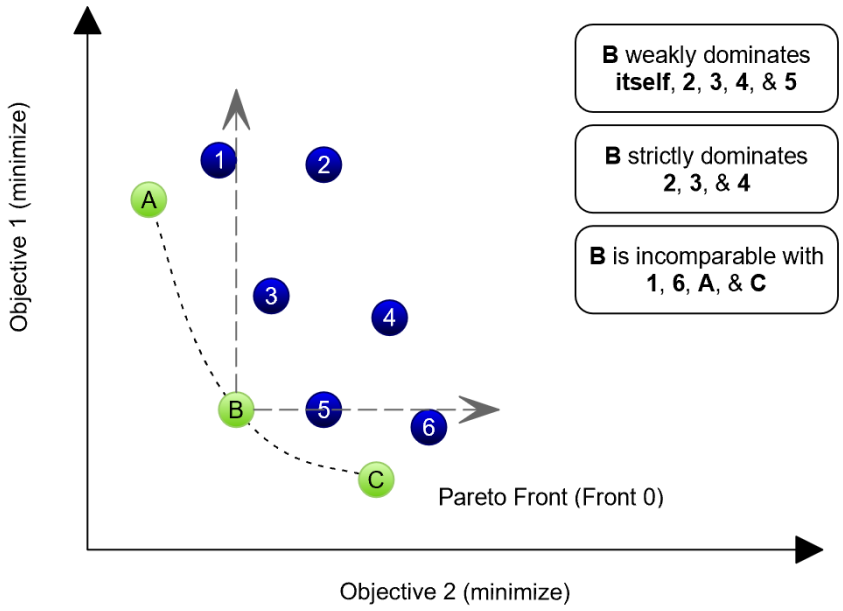

Figure 1. Dominance relationships

to A [12]. In other words, a solution is Pareto Optimal if no other solution to the problem dominates it. In Figure 1, solutions A, B, and C are Pareto Optimal. In a solution set, the subset of Pareto Optimal solutions is known as the Pareto set, or the Pareto front, or the nondominated set [19]. In the case where the Pareto front is found for the entire search space, meaning all Pareto Optimal solutions have been found, this is called the Pareto Optimal Set, sometimes denoted as PF^* [12]. This differs from a Pareto front in that the Pareto Optimal Set for a given problem is unique, but any subset of solutions to that same problem has a Pareto front, of which all, some, or none of the solutions may be members of the Pareto Optimal Set. This distinction is important when comparing algorithms because many algorithms, including EAs, make no guarantee to find the Pareto Optimal Set, but merely a good approximation of that set.

Parameter Tuning.

A typical EA has multiple parameters that dictate its behavior. Each variation and selection operator has at least one parameter associated with it, in addition to the higher level parameters like population size and maximum numbers of generations or evaluations. Adjusting any one of these parameters can affect the behavior of the algorithm, and adjusting multiple parameters together can amplify or mitigate the individual effects in interesting and surprising ways [19]. The process of determining the best combination of parameters for a given problem is known is parameter tuning, and is a field of research unto itself. Many recent research projects have been devoted solely to the determination of ideal parameters [26, 27] or automating the tuning of parameters [28, 29], which has been a staple of the Evolution Strategies branch of EA research since its inception. Parameter tuning is not the primary focus of this research. Values proposed in [9] are used for most parameters. The one notable exception to this is the case of stopping conditions. These are the parameters that determine when the algorithm should stop its search, and can have tremendous impact on the output of an algorithm. This research adheres to the recommendation of Beiranvand et al. [30] to ensure that all algorithms to be compared use the same stopping conditions.

Relevant Algorithms and Libraries.

When dealing with a real-world problem for which little is known about its objective space, selecting an algorithm can be a difficult. There are countless algorithms in existence today and many software packages available that implement some collections of those algorithms. Searching the literature and internet for suggestions regarding which algorithm to use for a given problem typically yields a daunting and generally unhelpful list of recommendations. What follows here is a series of short summaries of a few relevant software packages, as well as some of the more popular

multi-objective algorithms. The purpose is to briefly summarize the distinctive features, capabilities, and limitations found in the original literature for each. Specific operators employed by the algorithms are not discussed in depth. Software packages are limited to those available for the Python language, as the existing codebase consists entirely of Python.

inspyred.

The software package used in Stern et al.'s experiments [9], Garrett's *inspyred* [31], is a collection of optimization algorithms for Python. It consists mainly of single-objective algorithms, is somewhat limited in the types of data that can be used with the algorithms implemented, and requires the user to provide a significant amount of code to operate. It is, however, well-documented and easy to get started when applying to a custom problem. It also offers built-in capabilities for distributed processing using several different models and underlying technologies, from the use of multiple cores on a standalone machine, all the way up to distributing an algorithm on an HPC cluster.

PyGMO.

Published by the European Space Agency's (ESA) Advanced Concepts Team (ACT) [32], *PyGMO* is a Python library implementing many bio-inspired algorithms [33]. It is essentially a Python wrapper around their *PAGMO* library, which is implemented in C++. It has the obvious advantage that the C++ core is doing the heavy lifting, and therefore tends to be faster than most Python implementations of the same procedures. Documentation is extensive and, like *inspyred*, it offers strong support for distributed computing. It offers an impressive collection of algorithms, but of its 34 algorithms, only 4 support multi-objective problems.

Table 1. Summary of operators used by common algorithms. A - External Archive, BT - Binary Tournament, C - Crowding, D - Dominance, ϵD - Epsilon Dominance, I - Indicator, N - Neighborhood, R - Random

Algorithm	IBEA	MOEA/D	NSGA-II	εNSGA-II	NSGA-III	PAES	SPEA2
Parent Selection	BT	R, N	BT	BT	R	n/a	BT
Survivor Selection	I	D	D, C	ϵD	D, N	n/a	D, A
Recombination	✓	✓	✓	✓	✓		✓
Mutation	✓	✓	✓	✓	✓	✓	✓
Local Search		✓				✓	

Platypus.

Hadka's *Platypus* package [34] is a pure Python library which is exclusively devoted to multi-objective algorithms. While it does not have as extensive a collection of algorithms as PyGMO, each one is multi-objective out of the box. Compared to the others, it is rather poorly documented. Despite the limited documentation, it is relatively easy to implement a custom problem quickly. It features a well-developed type system that allows virtually any data type to be used with any algorithm (though some algorithms inherently exclude certain data types), and also eliminates the need for the user to implement a solution generator. With few exceptions, a problem needs to be defined only once and can be used with any algorithm available in the package without any changes to the code. Due to its wide offering of MO algorithms and relative ease of use, this is the library used exclusively for this research.

IBEA.

The Indicator-Based Evolutionary Algorithm (IBEA) [35], is an EA that makes use of binary quality indicators as a survivor selection tool. Binary quality indicators are comparison methods that can be used to compare two multi-objective solutions quantitatively, and are discussed later in this chapter. After randomly generating an initial population, IBEA uses binary tournament selection, recombination, and mutation to generate offspring to be added to the population. It calculates an indicator

value for each member of the population by comparing it to every other individual in the population and summing the results. The individual with the lowest (worst) value is eliminated. The compare-and-eliminate process is repeated until the population is reduced to the specified population size. This is one generation. The entire procedure is repeated with the current population, and subsequent populations, until the maximum generation is reached or some other termination criteria is met.

MOEA/D.

The multi-objective evolutionary algorithm based on decomposition (MOEA/D) [36] attempts to exploit single-objective optimization techniques by decomposing the problem into a user-defined number of single-objective sub-problems. Each sub-problem applies a unique weighting scheme to the multi-objective problem (MOP), allowing all of the objectives to be collapsed to a single value. Each weighting scheme simply specifies the relative weight of each objective to simulate a decision maker's preference. These different weightings of objectives serve to break the objective space into "neighborhoods" where individuals of similar fitness are segregated; mating is restricted to random selection within each neighborhood. The final output of MOEA/D is still an n-dimensional Pareto front, where n is the number of objectives in the MOP, but internally the algorithm splits its computation time evenly between each weighting scheme, making use of single-objective heuristics that might otherwise be unavailable to a typical MO algorithm.

NSGA-II.

The Non-dominated Sorted Genetic Algorithm II (NSGA-II) [37] is one of the most successful MOEAs developed to date, routinely appearing in literature as a comparison benchmark for newly proposed algorithms. The distinctive features of

this algorithm are the sorting of the population into multiple domination fronts and the use of a crowding calculation. Multiple domination fronts are found in the same way as finding a single front, except the procedure is repeated several times, removing the current Pareto front between each repetition. Each front is assigned a value (e.g. Front 0, Front 1). Crowding is approximated by the perimeter length of a cuboid drawn between the closest neighbors to a solution under consideration. Taken together, survivor selection is accomplished by preferring solutions in lower fronts and in less crowded regions in an attempt to maintain a population that is evenly-spread in objective space. Many variations of NSGA-II exist [12].

Epsilon NSGA-II.

Of the many variants of NSGA-II, one interesting option is the Epsilon Non-Dominated Sorted Genetic Algorithm II (ϵ-NSGAII) [38]. This variant makes use of ϵ-dominance [39] to eliminate the need for a distinct crowding function and, therefore, reduce overall computational cost. This technique requires a user to input epsilon values for each objective, which are then used to form a grid in objective space. Solutions that occur in the same grid block are compared and dominated solutions are removed. The overall result is a population that includes no more than one solution per grid block at the end of the dominance sorting step. This method can be helpful when something is known of a decision maker's preferences in advance, allowing for a coarser (larger epsilon) or finer grid size (smaller epsilon) for different objectives.

NSGA-III.

Another successor to NSGA-II is NSGA-III [40]. With the overall success of NSGA-II, one might be tempted to look at NSGA-III and assume that it must be an

upgraded, drop-in replacement for NSGA-II. This is not the case, though. NSGA-III was explicitly developed to handle many-objective problems (problems with four or more objectives). While it is capable of handling a standard MOP with two or three objectives, it builds on the framework of NSGA-II, adding five adaptations to mitigate the special considerations of many-objective problems. This can result in a modest increase in computational complexity over its predecessor. For traditional MOPs, such as the one studied in this research, NSGA-III can normally be disregarded and is not evaluated herein.

PAES.

The earliest of algorithms discussed in this review, the Pareto Archived Evolution Strategy (PAES) [41] is designed to be a computationally inexpensive baseline algorithm against which more complicated algorithms can be compared. Unlike the other algorithms discussed, it uses only local search, and is not population-based. Instead, it evaluates one solution at a time, moving to another by performing small mutations. As it traverses genotypic space, an external archive of non-dominated solutions is maintained and updated with each evaluation. At its termination, PAES outputs this archive. Though seemingly outclassed by the other algorithms on the list, it can be useful to consider less complex solvers when dealing with real-world problems for which there is no known Pareto Optimal Set.

SPEA2.

In improvement upon the original Strength Pareto Evolutionary Algorithm, SPEA2 [42] is another well-known algorithm that is used as a benchmark for newly proposed MOEAs. Its distinctive feature is an external archive of non-dominated solutions found throughout the search. The archive has a pre-defined maximum size.

Once the archive is full, a truncation operator is used to remove individuals from densely populated regions of the Pareto front to make room for individuals that belong to less-densely populated regions. Modifications to the techniques associated with the archive distinguish SPEA2 from SPEA. Reproduction is accomplished using binary tournament selection and standard recombination and mutation operators.

2.5 EA Comparison Methods

With an assortment of algorithms at one's disposal, methods by which to compare their results are required. Several methods are available for comparing solution sets, and with multiple runs of a single algorithm, these methods can be extended to compare algorithms, thereby elucidating benefits and shortcomings of the various algorithms. This section discusses the comparison methods that are pertinent to this research and, without loss of generality, assumes a minimization problem.

Following Zitzler et al. [24], Table 2 defines five kinds of binary relations that can exist between two solutions and extends those concepts to solution sets. The table omits equality (=), which is a special case of incomparability but is nonetheless a sixth binary relation in this category.

Table 2. Binary relations defined for this research as they exist between objective vectors (solutions) and approximation sets (solution sets). Adapted from Zitzler at al. [24]

relation	objective vectors		approximation sets	
strictly dominates	$z^1 \lessdot z^2$	z^1 is better than z^2 in all objectives	$A \lessdot B$	every z^2 in B is strictly dominated by at least one z^1 in A
dominates	$z^1 \prec z^2$	z^1 is at least as good as z^2 in all objectives and better in at least one objective	$A \prec B$	every z^2 in B is dominated by at least one z^1 in A
superior[1]			$A \triangleleft B$	every z^2 in B is weakly dominated by at least one z^1 in A and $A \neq B$
weakly dominates	$z^1 \preceq z^2$	z^1 is not worse than z^2 in all objectives	$A \preceq B$	every z^2 in B is weakly dominated by at least one z^1 in A
incomparable	$z^1 \| z^2$	neither z^1 weakly dominates z^2 nor z^2 weakly dominates z^1	$A \| \| B$	neither A weakly dominates B nor B weakly dominates A

Any valid solution set comparison method is able to make at least one of the six types of claims about relations on the space of solution sets. However, all methods known to the author are limited to some proper subset of these relations. Each method collapses complex relationships between sets into a single scalar measure in a distinct way, and the various methods are therefore capable of answering different kinds of questions about the relationships between solution sets. The remainder of this section discusses five solution set comparison methods: Visual, Binary Hypervolume, Coverage, Binary ϵ-Indicator, and Binary Additive ϵ-Indicator. The discussion includes the relations that can be determined by each, which are summarized in Table 4 in Chapter IV.

Visual.

Visual comparison is the oldest and most subjective method. In this method, two solution sets are plotted in a single graph, and then visually evaluated. It is capable of determining whether one solution set dominates another in a MOP with two objectives, but is not capable of quantitatively evaluating the domination, nor can it quantify relationships where neither set fully dominates the other (intuitively, this is the case whenever the Pareto fronts "cross"). It is far less capable of evaluating three-objective MOPs, and essentially can not be applied to problems with more than three objectives. This method is not employed for this research and is not included in Table 4.

Binary Hypervolume.

Hypervolume (HV) is the union of polytopes formed between each point in a solution set and some reference point [12]. For example, in two dimensions (a two-objective problem), it is the area formed by the union of rectangles between each

solution and the reference point. In three dimensions, such as the problem addressed in this research, it is volume of the union of cuboids formed between each solution and the reference point. Taken by itself, it describes the volume of objective space weakly dominated or "covered" by the solution set under consideration. Zitzler [43] defines binary hypervolume

$$I_{H2}(A, B) = HV(A + B) - HV(B), \tag{2}$$

where $HV(A + B)$ is the hypervolume of the union of Pareto front A and Pareto front B. Thus, $I_{H2}(A, B)$ indicates the volume of decision space weakly dominated by solution set A, but not by solution set B, which Zitzler proposes as a method to compare two solution sets using hypervolumes.

To decisively answer any questions about the relationship between the two solution sets, this measure should be performed in both directions ($HV\,(A, B)$ and $HV\,(B, A)$). Doing so renders an indication of overall "betterness," and can, at best, indicate if one set weakly dominated the other.

Coverage.

For solution sets A and B, the coverage indicator $I_C(A, B)$ is the fraction of points in B that are weakly dominated by A [43, 44]

$$I_C(A, B) = \frac{|\{b \in B \mid \exists\, a \in A : a \preceq b\}|}{|B|}. \tag{3}$$

Like binary hypervolume, this indicator relies on weak dominance and, therefore, $I_C(A, B)$ is not guaranteed to equal $1 - I_C(B, A)$. As such, I_C also should be evaluated in both directions. Unlike binary hypervolume, which deals with volumes of objective space, the coverage indicator relies on pairwise comparisons of objective

vectors. Consequently, it is more susceptible to skewing by a few exceptional solutions in an otherwise mediocre solution set. Zitzler [43] recommends using I_{H2} and I_C as complementary tools to build a fuller picture of the relationship between solution sets.

Binary ϵ-Indicator.

Zitzler et al. [24] introduce the ϵ-Indicator (I_ϵ) to allow one to not only make claims about the binary relationship between two solutions sets, but to also quantify the degree of certain relationships. For example, in addition to determining if A strictly dominates B, it can also quantify the factor by which it dominates B. This capability is very useful for the objective comparison of algorithms, as it can be combined with other characteristics of the algorithms, such as running time, to quantify the amount of qualitative gain or loss in contrast with the non-qualitative factors.

In simple terms, $I_\epsilon(A, B)$ computes A's ϵ-dominance of B, outputting the value of ϵ. Specifically, ϵ-dominance is defined as follows: for a minimization problem with n positive objectives ($Z \subseteq \mathbb{R}^{n^+}$), an objective vector $z^1 = (z_1^1, z_2^1, ..., z_n^1) \in Z$ ϵ-dominates another objective vector $z^2 = (z_1^2, z_2^2, ..., z_n^2) \in Z$, written $z^1 \preceq_\epsilon z^2$, if and only if

$$\forall 1 \leq i \leq n : z_i^1 \leq \epsilon \cdot z_i^2 \tag{4}$$

for a given $\epsilon > 0$. The binary ϵ-indicator I_ϵ for two solution sets, A and B is defined as follows:

$$I_\epsilon(A, B) = \inf_{\epsilon \in \mathbb{R}} \{\forall z^2 \in B \, \exists \, z^1 \in A : z^1 \preceq_\epsilon z^2\} \tag{5}$$

The value of $I_\epsilon(A, B)$ indicates the value by which every objective value in B can be multiplied to scale the entire solution set in objective space to a point where B is weakly dominated by A.

Binary Additive ϵ-Indicator.

Zitzler et al. [24] also propose the Binary Additive ϵ-Indicator. It is essentially the same indicator as the Binary ϵ-Indicator with the same capabilities and limitations. This version uses an additive ϵ-dominance ($\preceq_{\epsilon+}$) to find the largest value that can be added to every objective value in B to translate the entire solution set in objective space to be weakly dominated by A. This indicator is therefore defined as

$$I_{\epsilon+}(A, B) = \inf_{\epsilon \in \mathbb{R}} \{\forall z^2 \in B \, \exists \, z^1 \in A : z^1 \preceq_{\epsilon+} z^2\} \tag{6}$$

where $z^1 \preceq_{\epsilon+} z^2$ if and only if

$$\forall 1 \leq i \leq n : z_i^1 \leq \epsilon + z_i^2 \tag{7}$$

Despite its nearly identical function, this indicator does not always reveal the same objective relationships as the standard ϵ-Indicator. The use of addition instead of multiplication makes it possible to get different relative results on problems where objectives use different scales and are not normalized.

2.6 Space Situational Awareness

The real-world problem of designing a near-optimal space surveillance network is a complicated issue of aerospace and systems engineering. While this research does not seek to break new ground in those disciplines, some familiarity with the material is appropriate. What follows is a brief introduction to the basic concepts of SSA and some recent research in the design of near optimal SSNs.

SSA is central to a nation's ability to operate in space, and is defined by the Joint Chiefs of Staff as "the requisite foundational, current, and predictive knowledge and characterization of space objects and the [operational environment] upon which space

operations depend" [45]. While there are many ways to gather information in support of SSA, one of the primary tools is a SSN of optical telescopes devoted to making detailed observations of the RSOs in orbit around the Earth.

[46] divides SSA into four functional capabilities:

1 Detect, Track, Identify (DTID) - the ability to search, discover, and track space objects in order to maintain custody of objectsand events; distinguish objects from others; and recognize objects as belonging to certain types, missions, etc.

2 Threat Warning and Assessment (TW&A) - the ability to predict and differentiate between potential or actual attacks, space weather environment effects, and space system anomalies, as well as provide timely friendly force status.

3 Characterization - Characterization is the ability to determine strategy, tactics, intent, and activity, including characteristics and operating parameters of all space capabilities (ground, link, and space segments) and threats posed by those capabilities.

4 Data Integration and Exploitation (DI&E) - the ability to fuse, correlate and integrate multi-source data into a UDOP and enable decision-making for space operations.

The latter three items rely heavily upon DTID as a primary source of data. This research mainly supports DTID, focusing on the detect-and-track capabilities provided by a space surveillance network.

While there are a number of technologies available for observing objects in orbit, the Stern and Wachtel model [9] focuses exclusively on optical telescopes. An SSN can consist of both ground- and space-based telescopes. Ground-based telescopes are limited to the hours of darkness to make observations, while space-based telescopes may make continuous observations, limited only by objects, such as the Earth passing

in front of RSOs or sources of light, such as the Moon and Sun, passing behind RSOs. The relative motion of the Sun, Moon, and Earth, makes it necessary to employ multiple telescopes at physically distant locations to enable observation of the entire catalog of observable RSOs. The typical use of an SSN is to schedule all telescopes to routinely scan through some subset of observable RSOs in an efficient manner to maintain current data on the current orbit of each RSO. For the purposes of this research, only RSOs in geostationary orbit are considered.

Optimizing SSN designs using models, with or without EAs, is not a new idea. As far back as 2005, Fahnestock and Erwin [47] used a brute-force gridding technique and a basic (unidentified) EA to optimize the design of a constellation of space-based telescopes for observation of GEO. Among their other results, the research showed that the EA found comparable results at a fraction of the computational cost. Yates, Spanbauer, and Black [48] devised a process for evaluating entire constellations of space-based telescopes while studying their theoretical performance in different types of orbits. Ackerman et al [49, 8] began to evaluate entire SSN architectures, rather than individual components, evaluating gaps in current coverage and ways to supplement them ground- and/or space-based telescopes. Building on their work, Stern and Wachtel devised a comprehensive model for an SSN allowing for ground-based telescopes, as well as space-based constellations in three different types of orbits, applied an EA to search for near optimal solutions, and employed HPC resources to accomplish the search. One common factor to these is that there has been no formal, explicit evaluation and comparison of search algorithms for this problem. [50] extends Stern and Wachtel's research by expanding design boundaries and refining the methodology.

2.7 Stern and Wachtel

Stern and Wachtel's work [9] forms the basis for this research, with the pertinent details summarized here. A high level depiction of the methodology developed by Stern and Wachtel is shown in Algorithm 1. Their model has 77 sensor configurations and uses 813 simulated RSOs in GEO. Sensor configuration refers to a particular location on the ground or a constellation of space-based telescopes in a particular orbit. Each of the 77 configurations are individually simulated, producing three text files for each sensor/target pair in the simulation, meaning that a total of 62,601 pairs are simulated, producing 187,803 files per simulation. These files record data about line-of-sight, distance, angles, illumination conditions, and relative positions between the sensor, target, Sun, and Moon. Two different 24-hour time periods are simulated to address some of the more challenging scenarios for a SSN, at the cost of doubling the computational load and output of the simulation phase. Once simulated, the data can be reused for the entire optimization phase, and further simulations are not necessary.

Optimization is accomplished using NSGA-II [37] and consists of up to 100 generations where the population consisted of 96 possible architectures. The computational load is spread across multiple nodes by splitting a generation into groups of individuals equal to the number of cores available in the HPC nodes (8, in their case), and then collecting the results back to the node running the optimization algorithm. Each generation is a new job submitted to the HPC queue, meaning that there were 100 separate jobs submitted per run of the optimizer. The final output of the optimization is a text file containing the full architecture and objective values of each non-dominated solution selected by the search algorithm.

Table 3. Architectural Parameters and Ranges in the Stern and Wachtel Model [9]

Architectural Parameters (genes)	Lower bound	Upper bound	Step size
GBT count (at each of nine locations)	0	4	1
GBT aperture diameter	0.5 (m)	4	0.5
LEO sun-synchronous altitude	500 (km)	1000	100
LEO sun-synchronous satellites per plane	0	2	1
LEO sun-synchronous planes	1	2	1
LEO sun-synchronous aperture diameter	0.15 (m)	1	Varies
LEO equatorial altitude	500 (km)	1000	100
LEO equatorial observer count	0	4	1
LEO equatorial aperture diameter	0.15 (m)	1	Varies
Near-GEO observer altitude (Δ from GEO)	-1000 (km)	1000	500
Near-GEO observer count	0	4	1
Near-GEO observer aperture diameter	0.15	1	Varies

Model.

The model developed by Stern and Wachtel consists of 28 decision variables and 3 objectives (i.e a 28-dimensional decision space and a 3-dimensional objective space). It includes four classes of telescope and a discretized alphabet of values for each decision variable. The model is summarized in Table 3.

The four sensor classes allowed are as follows: ground-based telescopes (GBT), equatorial low-earth orbit (LEO) observation satellites (obsats), sun-synchronous obsats, and near-GEO obsats. Ground based telescopes are limited to nine real-world locations already in use for SSA, and allows for zero to four telescopes at each location. Though separate locations could have different telescope designs, if multiple telescopes were used at a single location, they are required to be identical. Up to one constellation of equatorial obsats can be placed at one of six altitudes, with up to four satellites in the constellation. Up to two sun-synchronous obsats can be placed into each of up to two orbital planes in one of six altitudes. Finally, up to one constellation of up to four near-GEO obsats can be placed at one of four altitudes: two lower than GEO, and two higher than GEO. For the three possible orbits of obsats,

different parameters were allowed for each constellation, but all telescopes within a constellation are required to be identical.

Objective Functions.

The ultimate goal is the minimization of three objectives: cost, latency, and detection size. Cost, which is fairly self-explanatory, includes estimates of acquisition costs, operation costs (for 10 years), maintenance costs, and launch costs for space-based telescopes. It is shown in Equation 8.

$$C(X) = \sum C_{Sat} + \sum C_{Tel} + \left(\sum C_{SatOp} + \sum C_{TelOp}\right) \times 10yrs + \sum C_{Launch} \quad (8)$$

The individual cost functions are listed in Equations 9 through 12; D is the aperture diameter in meters. Note that C_{Launch} does not have an equation. It is determined using a launch vehicle selection script, based on a aperture diameter and estimated weight, selecting the least expensive launch vehicle assuming no ride sharing and limiting launch sites to either Cape Canaveral or Vandenberg Air Force Base.

$$C_{Sat} = \$400,000,000 \times D \quad (9)$$

$$C_{Tel} = \$4,000,000 \times D^{2.45} \quad (10)$$

$$C_{SatOp} = \$9,900,000 \times numConstellations \quad (11)$$

$$C_{TelOp} = C_{Tel} \times 0.20 \quad (12)$$

Latency, which is the mean maximum of RSO observation gap (MMOTG), represents the longest gap in observation for an average target satellite and is shown in

Equation 13.

$$L(X) = \frac{\sum_{RSO=1}^{numTgt} \left[\max_{1 \leq o \leq numObs} (t_1 - t_{start}, t_{o+1} - t_0, t_{end} - t_{numObs}) \right]_{RSO}}{numTgt} \tag{13}$$

The final objective is Detection Size. Also fairly self-explanatory, this is the average minimum RSO size that a given architecture should be able to detect in GEO. It is calculated by determining the average minimum detection size for all scheduled observations. This objective function depends not only on a telescope's aperture size, but a number of other environmental factors, as well as physical characteristics of the photoelectric detector used in a telescope and the instantaneous distance to the given target. These issues are explored in detail in [9]. The final objective function, however, is given in Equation 14.

$$S(X) = \frac{\sum_{RSO-1}^{numTgt} \left(\frac{\sum_{o=1}^{numObs} size_o}{numObs} \right)_{RSO}}{numTgt} \tag{14}$$

Methodology.

Stern and Wachtel's methodology consists of two broad phases: simulation and search. Algorithm 1 summarizes the overall methodology. For both phases, a "full" complement of RSOs is 813, but using a small number of RSOs allows for test runs to be performed quickly, even on regular workstations.

Simulation.

Cost can be calculated using just the genotypic representation of an architecture, however latency and minimum detection size both require simulation data before they can be calculated. Data was simulated using AGI's Systems Toolkit (STK)

software [51] running on AFRL's now decommissioned "Spirit" HPC. Rather than simulating every possible architecture, each of the 77 possible sensor configurations are simulated individually with 813 simulated targets in GEO. The data can then be mixed and matched into any combination of configurations needed to create a data set appropriate to any possible architecture. The simulation does not model sensors or RSOs (those characteristics are calculated during the search phase), but only gathers data about line-of-sight, angles, illumination conditions, and range between each sensor, each target, and the Sun and the Moon. Much of this is collected in the form of STK's built-in AER report which provides azimuth, elevation, and range between two appropriate objects. Each of the 77 simulations are run on an HPC node as a separate job using STK for Linux. Data was output to text files directly by STK using pre-defined report templates. Due to software limitations of the time this methodology was developed, this file-based method is necessary as the simulations are orchestrated using the *connect* system, which is a one-way communication method in which commands can be scripted and sent to STK, but no data can be returned [52]. Unfortunately, this method creates a total of three text files per sensor/target pair, and a total of 187,303 files per 24-hour period simulated. Their work includes simulations of two days which each offer a different challenge to SSA. First, summer solstice is simulated as it provides the shortest night of the year for most of the GBTs. Second, the vernal equinox is simulated as it puts each GEO RSO in eclipse for 70 minutes, making passive detection impossible.

Search.

Search is accomplished using NSGA-II as implemented in the *inspyred* software and four different search techniques. In addition to a standard unconstrained MO search, they also perform a constrained MO search, and a constrained and uncon-

strained single-objective search. For the single-objective searches, the three objectives were normalized and equally weighted to collapse them into a single value.

Actual search is accomplished by identifying which text files contained the data required for a given architecture, reading in and parsing the data files, converting values to the correct data types, and then performing a series of calculations to determine cost, latency, and detection size values. Population size is set to 96, mutation rate was set to 5%, and each trial was terminated after 100 generations.

Algorithm 1: Original SSN Search Algorithm

input : A set A of valid space surveillance architecture parameters, a set of sets L of sensor locations, set T of targets (satellites in GEO)
output: A Pareto-optimal set of architectures found by optimizer

foreach *sensor* $s \in S$ **do**
    Simulate in STK
    foreach *target* $t \in T$ **do**
        Calculate access
        Calculate AER for (s, t)
        Calculate phase angles
        if *s is a GBT* **then**
            Calculate zenith angles
        Create and save access reports to disk
        Create and save AER reports to disk
        Create and save angle reports to disk
        Check (read in) all reports for valid data
`/* Start optimizer (NSGA-II)  */`
foreach *configuration c generated by optimizer from A* **do**
    Ingest all reports associated with sensors in c
    Combine report data
    Calculate objective functions

Computational Cost.

There are several factors that contribute to the "costliness" of this methodology. Some can be easily quantified while others are limited to generalization and anecdotal

[1]AER is stuff...

evidence. The driving factor in computational cost is the optimization. The simulations are not particularly time consuming, relatively speaking, largely due to the fact that they only needed to be completed once. Non-computational factors contributing to the overall time required are file I/O and the HPC queue wait times, both of which inject a large amount of idle time simply waiting for something to happen.

Optimization.

Optimization is a very costly aspect of the methodology. It involves a great deal of arithmetic, file I/O, and text parsing. Compounding this is the repetitive nature of search. Successful genotypes may be expressed in whole or in part for many generations, but every architecture is treated as new and the files are read in every time they are needed. It is difficult to quantify these effects as each architecture has a different number and combination of data files associated with it. In comparison with a call to cache or memory, disk access is orders of magnitude slower [53]. When dealing with hundreds of thousands of files, the file-based method can contribute vast periods of time to otherwise simple operations over methods that use fewer files or make better use of memory.

HPC Queue Wait Times.

The use of shared HPC resources necessitates the use of a queue system to ensure that no single user can monopolize an inappropriate portion of the HPC for extended periods of time. AFRL's HPC resources use the Portable Batch System (PBS) to manage the queue [54]. The very nature of the queue is that there is a delay before your job is run. There are also multiple queues for tasks of higher or lower priority. Most work falls into the "standard" queue, which can have extremely variable wait times depending on a number of factors. Typical wait times can range from a few

hours to over a day during exceptionally busy periods. Therefore, one cannot easily predict the delay that will be incurred for any one job, and the more jobs submitted, the greater the compounded delay. The search phase is implemented on the HPC by breaking each generation into a collection of parallel jobs, submitting a batch of jobs for each generation, waiting for results, building the next generation, and submitting another batch of jobs. This is a very inefficient use of PBS which causes each trial, at a minimum, to have incurred 100 successive waits in the queue (one for each generation). A more efficient, albeit more complicated to implement, technique is to use a single job for each trial, spreading the computational load via software across as many nodes as necessary to complete the trial in a reasonable period of time.

2.8 Chapter Summary

This literature review uncovered several important items. First, while some work has been done in applying HPC resources and EAs to the problem of SSN architecture optimization, no clear attempt has been made to formally evaluate the relative effectiveness of different algorithms or classes of algorithms against the problem. There is no one algorithm known to be the most effective multi-objective, but there are many multi-objective algorithms readily available via the *Platypus* library, each employing different techniques intended to get the best possible solution sets. There are also several methods by which one may compare one multi-objective solution set to another. These methods each make comparisons based on different characteristics of the solution sets. The comparison methods, therefore, are able to identify different subsets of the possible relationships solution sets may have. Therefore, no one method is a clear best method for comparing algorithms. Furthermore, these methods are specific to comparing individual solution sets; using these methods to compare algorithms require multiple runs to be performed and aggregated.

Second, the existing methodology developed by Stern and Wachtel is computationally expensive and depends on HPC resources. The use of HPCs led to some design choices that introduced substantial overhead that likely dominated the overall runtime of the optimization code and, to a lesser degree, the simulations. In order to perform multiple optimization runs with a variety of algorithms, overhead must be addressed. Overall runtime also seems to be strongly linked to the number of RSOs present in the simulation data. Reducing that number may prove to be a useful tool in driving down the computational cost of evaluating many algorithms against this problem.

This research applies the model described in [9], a variety of binary comparison techniques, and a selection of algorithms to build a better understanding of which algorithms are most effective on this problem, as well as to explore techniques to drive down computational costs associated with it. Chapter III describes how these tools are integrated to meet those goals.

III. Methodology

3.1 Chapter Overview

The purpose of this chapter is to explain the methods used to perform the computational experiments necessary to compare the performance of multiple search algorithms with the SSN architecture problem. Modification of data management scheme, pre-simulation of data, application of HPCs, and the software and hardware used are discussed. The chapter concludes with an explanation of the computational experiments that are conducted.

3.2 High-Level Approach

The computational experiments used to collect performance data are of a simple design. Simulation data is produced in advance. Each algorithm is run on the problem multiple times. Resulting solution sets are aggregated for each algorithm, and compared pairwise using the four binary comparison methods described in Chapter II. In addition to performing this process on the 813-RSO simulation data, it is repeated with data from simulations of 20, 81, 203, and 407 RSOs, or 2.5%, 10%, 25%, and 50%, respectively. For each set of data, an equal number of runs are performed.

3.3 Measure Performance

Many techniques are discussed in the literature by which to measure the performance of an algorithm depending on one's objectives and tools. The methods used to evaluate performance in this research are detailed below. After a brief explanation of how the overall workflow was modified from what was found in Stern and Wachtel's work, the general measurement strategy and the comparison of optimization runs and algorithms will be discussed.

Revised Algorithm.

Recall from Chapter II that Stern and Wachtel's original methodology relies on a somewhat inefficient file-based data management system that would likely dominate the running time of a search. Prior to comparing algorithms, the overall algorithm required some modest performance tuning in order to remove unnecessary time costs. The result of this tuning is shown in a revised version of the high-level algorithm that is used in this research, shown in Algorithm 2. The changes are fairly straightforward; the overall technique is not changed, but data flow was streamlined. Instead of using dozens of simulations to generate hundreds of thousands of files, simulations are consolidated, simulating entire sensor classes in four large simulations. Likewise, data is consolidated into four custom data structures that store simulation data for each class of sensor. These structures are saved using Python's *pickle* module at the end of the simulation, which preserves data types and data structures when the file is later read into memory. This class-based strategy leaves the algorithm open to parallelization, albeit on a coarser scale, while consolidating the output data into a manageable and portable footprint and enabling greater flexibility for anyone using the data (e.g. simulating the data once and then sharing with multiple users, teams, or platforms). This did come at a cost of approximately doubling the disk space required to store the data files, however, and required a larger memory footprint during optimization runs, relative to the original methodology.

The optimization phase deserializes the class simulation files into memory at the start of optimization. That data is used for the entire course of the optimization, eliminating all file I/O and string parsing from the actual optimization loop. These minor changes effectively mitigate the worst of the overhead, reducing average evaluation time from the roughly 22 minutes observed with the old methodology to approximately 5 minutes (when using the full set of simulation data), and enabling more

efficient utilization of processing power during the optimization phase. More details regarding the performance tuning process can be found in Appendix B.

Algorithm 2: Revised SSN Search Algorithm

input : A set A of valid space surveillance architecture parameters, a set C of sensor classes, containing sets S_c of possible possible locations (or constellations) of sensors within the class, set T of targets (satellites in GEO)

output: A Pareto-optimal set of architectures found by optimizer

```
Create a custom data structure for each class of sensor
foreach class c ∈ C do
    Simulate in STK
    foreach sensor s ∈ S_c do
        foreach target t ∈ T do
            Calculate access data and append to sensor-level access report
            Calculate AER data for (s, t) and append to sensor-level AER report
            Calculate phase angle data and append to sensor-level phase angle
             report
            if s is a GBT then
                Calculate zenith angles
                Append zenith data to sensor-level zenith angle report list
        Store sensor data to its class-specific data structure
    Serialize class-specific data structure to a single file on disk

foreach optimizer do
    Deserialize data files to memory
    foreach configuration c generated by optimizer from A do
        Calculate objective functions
```

Measurement Strategy.

Beirenvand, Hare, and Lucet [30] identify efficiency, reliability, and quality of solution as three common measures of performance. Measuring efficiency is normally done in terms of running time, fundamental evaluations, or, in rare cases, CPU time [30]. The number of fundamental evaluations is another possible measure of efficiency in the performance of an EA, especially in cases where convergence detection is used as a termination condition. This problem is not a real-time optimization, so efficiency is

not of particular interest, and was not used as a performance metric for the purposes of this research.

The reliability of an optimizer, how closely the results of one run resemble others, could be a useful metric in selecting a "best" algorithm. Each optimizer is run multiple times and the resulting solutions sets are analyzed using the four comparison methods described in Chapter II. The pairwise comparison of those solution sets result in eight comparison values per pair of runs, two for each comparison method.

Measuring the solution quality of the optimizers is accomplished in a similar fashion. Each of the comparison methods identified in Chapter II are designed for comparing solution sets, and not algorithms, but can easily be applied to algorithms by aggregating multiple runs of the same algorithm into a single Pareto front and then comparing to another algorithm's aggregated front [55]. Aggregating the runs done with each algorithm, quality comparisons are performed pairwise between algorithms' aggregate fronts using the same four comparison methods.

As discussed in earlier chapters, reducing the number of RSOs in the simulation data is one possible method of reducing the computational cost of performing fitness evaluations. Measuring the effect of using fewer RSOs is accomplished by repeating the process described above on multiple sets of simulation data. Each data set is generated using STK in the same way as the full data set, but with fewer RSOs than the original simulation. The full set of RSOs are stored in a randomized list designed to provide a realistic distribution of satellites for simulations of less than 813 RSOs, so selection of subsets consisted of simply selecting the first k RSOs, where k was the desired RSO count for that subset. This not only kept code simple, but also ensured simulations would be easily reproducible by anyone wishing to do so.

Finally, as recommended by [56], the specific parameters of the selected algorithms are be reported as well. These will include pertinent probabilistic rates (mutation,

crossover, etc), population limits, and operators used. Specific operators are reported in Chapter II, and other parameters are be listed in this chapter.

3.4 Selected Optimization Algorithms

In real world problems such as these, the choice of optimizer can be difficult as there is no known optimal solution, and often only general knowledge of the search space as it relates to the specific problem is available. There are an infinite number of possible optimization algorithms. Even limiting oneself to the "well-known" algorithms leaves dozens to consider. It is, therefore, impractical to attempt to exhaustively evaluate every optimization algorithm on a given problem. This section discusses the algorithms used in this research and why they were selected.

Previous Optimizer.

In Stern and Wachtel's work, optimization is accomplished using Garret's implementation of NSGA-II [31, 57]. Performance of the optimizer in [9] provide some hints that it may not be the best fit. The strongest hint is extremely rapid convergence. In experimental runs performed by Stern and Wachtel, evolution is limited to 100 generations, and typically they converge in under 30 generations. Given that the creators of NSGA-II required 500 generations to achieve convergence near the true Pareto-optimal front [57] for several well known multi-objective benchmark problems, such rapid convergence suggests that either the problem may be solvable using simpler methods or the algorithm is prematurely converging to a set of local optima.

Algorithms Used.

Five algorithms are used for this research. Two "classical" algorithms and three EAs are used to test a variety of factors. First, Random Search (RS) was applied.

While there is little chance that random search will be the best algorithm for real-world optimization problems, it serves as a baseline against which to compare more sophisticated algorithms. The refinements implemented in each of the other algorithms can be expected to garner quantifiable improvements over the performance of random search. In cases where two algorithms are deemed to be incomparable, there is a good chance that they will both still be comparable with random search, providing an additional possible method of comparison.

The other classical algorithm is the Random Restart Hill Climber (RRHC) [58]. Despite its traditional name, this algorithm is capable of either hill climbing (maximization) and hill descending (minimization). It is among the simplest search algorithm available over random search. At its core, it is just a random search with a local search heuristic applied, and can render surprisingly good results for many problems despite its simplicity. With the rapid convergence observed in Stern and Wachtel's work, this algorithm serves as an intermediate step between random search and the more sophisticated EAs.

The three EAs that are used are NSGA-II, IBEA, and SPEA2. NSGA-II was selected not only because it is a very well known and successful multi-objective evolutionary algorithm (MOEA), but also because it is the original optimizer used by Stern and Wachtel. Due to many changed variables, and the use of a single 24-hour period, the results of their work cannot be compared to this research. Running the algorithm again serves as somewhat of a surrogate for their results. Collectively, the three EAs were selected to compare different operators. Table 1 lists the operators used for a number of algorithms available in the Platypus library. Of them, roughly half use binary tournament for selection. Each of these were selected because they each use binary tournament, but different survivor selection operators (see Table 1). Differences in performance could inform future research efforts in this area.

3.5 HPC Implementation

Through the course of this research, it became apparent that, while the simulations themselves could be run in a reasonable amount of time on a high-end workstation instead of an HPC, the size of the search space and the costliness of the fitness evaluations do demand extreme computing power to effectively explore the space. After performance tuning was completed, evaluations averaged 303.2 seconds when using the full simulation data. With 2.428×10^{21} possible architectures in the underlying model, an exhaustive search evaluating 100 architectures in parallel at a time would take approximately 2.334×10^{14} *years* to complete, and more than two trillion years to explore just 1% of that space. Even using heuristics such as evolutionary techniques, the computational burden of a multi-objective search space this large was too great to place on even a very high-end workstation.

HPC Migration.

Migrating search to an HPC was a reasonably straightforward process. The Platypus library natively supports distributing computations via MPI [34], so only relatively minor changes were required to adapt the code to run on the HPC. The random search and random restart hill climber algorithms, which are not part of Platypus, did not require substantial coordination between parallel runs, thanks to the relative independence of one evaluation with all previous evaluations. With minimal modification, those algorithms are simply run as parallel, individual instances, reporting back final results to a central node where results were consolidated into a single solution set. Computations are parallelized into $n * k$ processes, where n is some number of HPC nodes, and k is the number of duplicates of simulation data that could fit in a node's memory. The value of n is calculated for each job to ensure that, for the designated number of generations, jobs do not exceed the maximum time limit for

jobs on the HPC, which is 168 hours (7 days).

Parallel Performance Tuning.

Parallelization revealed that using a few large data files resulted in an unreasonable memory footprint when running multiple instances. For the full data set, approximately 72 GB of memory was required for each parallel evaluation, resulting in a situation where a single HPC node would be required to run as few as few as two parallel evaluations on a standard Mustang node, or ten on a large memory node. This was a very costly use of the HPC, with up to 90% of the cores in a node going unused. In response to this, the code was modified to take 77 simulation data files that correspond to the 77 "locations" in Stern and Wachtel's model and new simulations were run to produce the necessary files. The result was an 80% reduction in memory requirement, just under 15 GB, with the added cost of loading up to 13 files for each evaluation. This was more than offset by the ability to run five times the number of parallel evaluations per HPC node.

3.6 Hardware and Software Used

The computational experiments are accomplished using a variety of platforms and tools. In line with the recommendations found in [56] for reporting the results of computational experiments, this section lists the most significant hardware and software applied to this research.

HPC Mustang.

AFRL's Mustang is the primary HPC used to perform the searches for this research. It offers 1,128 standard compute nodes, 24 large-memory compute nodes, and 24 GPU compute nodes, with a total of 56,448 compute cores. It features 244

terabytes TB of memory and is rated at 4.88 peak petaflops PFLOPS. Both standard and large-memory nodes were used. Both feature 48 cores per node, and either 192 gigabytes GB 768 GB of memory [14].

HPC Thunder.

AFRL's Thunder HPC is also used to perform searches on the smaller data sets. It offers 3,216 standard compute nodes, 4 large-memory compute nodes, 178 Xeon Phi compute nodes, and 178 GPU compute nodes, totalling 125,888 compute cores. It has 460 TB of memory and is rated at 5.62 peak PFLOPS. Only standard nodes were used on Thunder, which offers 36 cores and 128 GB of memory per node [14].

Workstation.

A Dell T5600 workstation is used to perform all development, STK simulations, and data analysis. It features two 8-core Intel Xeon e5-2680 processors and 128 GB of memory. Development and simulations are done on Windows 10 Professional.

STK.

STK 11.5 [51] is used to perform all simulations. Simulations are executed with the STK engine, reducing the time required by orders of magnitude. Python scripting is used to orchestrate the simulations and to collect and save the resulting data.

Python and Optimization Libraries.

Python 3.6 is used for all simulation and optimizations. The *inspyred* library [31] was used in early tests and updating of Stern and Wachtel's code. The Platypus library [34] is used for all EA-based optimization runs. PyGMO's [33] hypervolume tool is used in the binary hypervolume indicator.

3.7 Chapter Summary

The methodology described above makes use of a powerful SSN model, a selection of Evolutionary and Classical algorithms, and binary comparison methods to evaluate the effectiveness of different algorithms with the model, as well as to determine if using reduced sets of simulated RSOs can be used to evaluate algorithms for this problem. Chapter IV discusses and analyzes the results obtained by the computational experiments described in this chapter. Final conclusions are presented in Chapter V.

IV. Results and Analysis

4.1 Chapter Overview

This chapter presents the results obtained through the computational experiments, as well as the analysis of those results. First, individual runs are aggregated, compared, and analyzed to determine overall effectiveness of algorithms. Tables 6 and 7 list the comparison values obtained by the pairwise comparison of the aggregated fronts. Table 1 summarizes some basic characteristics of the aggregate fronts to provide context for the subsequent comparisons between them. Analysis of aggregate data is performed in Section 4.4. An analysis of the relative reliability of each algorithm is also conducted in that section, comparing individual runs of different algorithms to determine which outperformed one another, in terms of the binary comparison methods, with statistical significance.

Next, the impact of using fewer RSOs is considered. Results from the optimization runs performed on the smaller data sets are subjected to the same methodology as the full data set, and results are compared to determine trends. Tables 9 through 12 list the results of all four comparison methods across the different data subsets, and Tables 13 through 16 list the results of reliability analysis across subsets.

This chapter contains many tables indicating results of the comparison methods. Each comparison method is built around the idea of comparing two arbitrary Pareto fronts, A and B. Table 4 describes the relationships that can be determined when comparing those Pareto fronts with the four methods used in this research. With respect to that guide, the tables in this chapter should be read as the left-most column listing the algorithm which produced Pareto front A, and the top row listing the algorithm that produced Pareto front B. Furthermore, general guidelines for interpreting the data are listed below. For each, when a larger or smaller value is said

to be "better" for A, one should take it to mean that the largest or smallest value in any row indicates the B algorithm against which A performed the best, and that a relative ranking of algorithms from best to worst, relative to A, can be determined by sorting the values from largest to smallest, or smallest to largest, respectively.

- Binary hypervolume is the difference of the hypervolume of B subtracted from the hypervolume of the union of A and B. It is measured with respect to a reference point indicating the maximum theoretical values for each objective function. Reading across any row in the table, larger values are better for A.

- Coverage is the number of points in B that are equal to or weakly dominated by at least one point in A, divided by the total number of points in B. Again, larger values are better for A.

- The ϵ-Indicator is the value by which every objective value in B can be multiplied and still be weakly dominated by A. The ϵ value indicates how much B must be scaled in order for the entire Pareto front to be weakly dominated by A, so smaller ϵ values are better for A.

- The Additive ϵ-Indicator is the value that can be added to every objective value in B and still be weakly dominated by A. In other words, the ϵ value indicates how much Pareto front B needs to be *translated* to be weakly dominated by Pareto front A, so smaller ϵ values are, again, better for A.

Section 2.5 summarizes the capabilities of each comparison method. It is the basis for assertions that are made based on the comparison method values.

4.2 Computational Experiments

Five algorithms are run five times each for each of the five data sets. Ideally, more than five runs per algorithm would be performed, but this value was selected to

Table 4. Binary indicators and their capabilities [55]

Indicator	can determine relation: $\succcurlyeq$	$\succ$	$\triangleright$	$\succeq$	$=$	$\parallel$
Epsilon Indicator (I_ϵ)	$I_\epsilon(A,B) < 1$	n/a	$I_\epsilon(A,B) \leq 1$ $I_\epsilon(B,A) > 1$	$I_\epsilon(A,B) \leq 1$	$I_\epsilon(A,B) = 1$ $I_\epsilon(B,A) = 1$	$I_\epsilon(A,B) > 1$ $I_\epsilon(B,A) > 1$
Additive Epsilon Indicator ($I_{\epsilon+}$)	$I_{\epsilon+}(A,B) < 1$	n/a	$I_{\epsilon+}(A,B) \leq 0$ $I_{\epsilon+}(B,A) > 0$	$I_{\epsilon+}(A,B) \leq 0$	$I_{\epsilon+}(A,B) = 0$ $I_{\epsilon+}(B,A) = 0$	$I_{\epsilon+}(A,B) > 0$ $I_{\epsilon+}(B,A) > 0$
Coverage (I_C)	n/a	$I_C(A,B) = 1$ $I_C(B,A) = 0$	$I_C(A,B) = 1$ $I_C(B,A) < 1$	$I_C(A,B) = 1$	$I_C(A,B) = 1$ $I_C(B,A) = 1$	$0 < I_C(A,B) < 1$ $0 < I_C(B,A) < 1$
Binary Hypervolume (I_{H2})	n/a	n/a	$I_{H2}(A,B) > 0$ $I_{H2}(B,A) = 0$	$I_{H2}(A,B) \geq 0$ $I_{H2}(B,A) = 0$	$I_{H2}(A,B) = 0$ $I_{H2}(B,A) = 0$	$I_{H2}(A,B) > 0$ $I_{H2}(B,A) > 0$

Table 5. The parameters used for each algorithm considered. No parameter tuning was performed in this research.

	Rate of Mutation	Rate of Crossover	Population Size	Generations	Evaluations
RS	n/a	n/a	n/a	n/a	25000
RRHC	n/a	n/a	n/a	n/a	25000
NSGA-II	.05	1.0	100	250	n/a
IBEA	.05	1.0	100	250	n/a
SPEA2	.05	1.0	100	250	n/a

allow for statistical significance when performing reliability analysis, while remaining feasible to complete with the HPC resources available. Each run terminates after evaluating 25,000 architectures (classical search algorithm) or 250 generations with a population size of 100 architectures per generation (EAs). The large differences in scale between the objectives affect the results of two of the four comparison methods, so all objective values are normalized to a scale of zero to one before applying the binary comparison methods. Altogether, 25 optimization runs are performed with each data subset, producing a total of 125 Pareto fronts.

4.3 Comparison of Algorithms

First, algorithms are compared to one another, directly. In order to make these comparisons, it was necessary to produce a single Pareto front for each algorithm. Using solution sets produced using the full 813-RSO simulation data, Pareto fronts from individual runs are merged and dominated solutions removed to form a single Pareto front for each algorithm. Algorithms are compared pairwise using the four comparison methods. See Table 7 for the results of these comparisons. To analyze these results, it was useful to collect the basic characteristics of the Pareto fronts in question. Understanding that each comparison method tends to reward different characteristics of a Pareto front, and therefore tend to produce conflicting results, this descriptive data can help clarify how these apparent contradictions come about. Table 6 summarizes the aggregate fronts in terms of their individual hypervolumes, as well as the range of values found for each objective. Again, Table 4 is used to interpret the results of the comparisons. Graphs of the aggregated Pareto fronts can further clarify the results, and are shown in Figures 2 through 6.

Aggregate Pareto fronts are vulnerable to over-representing a single exceptional run. Therefore, the algorithms are further tested to determine their relative reliability

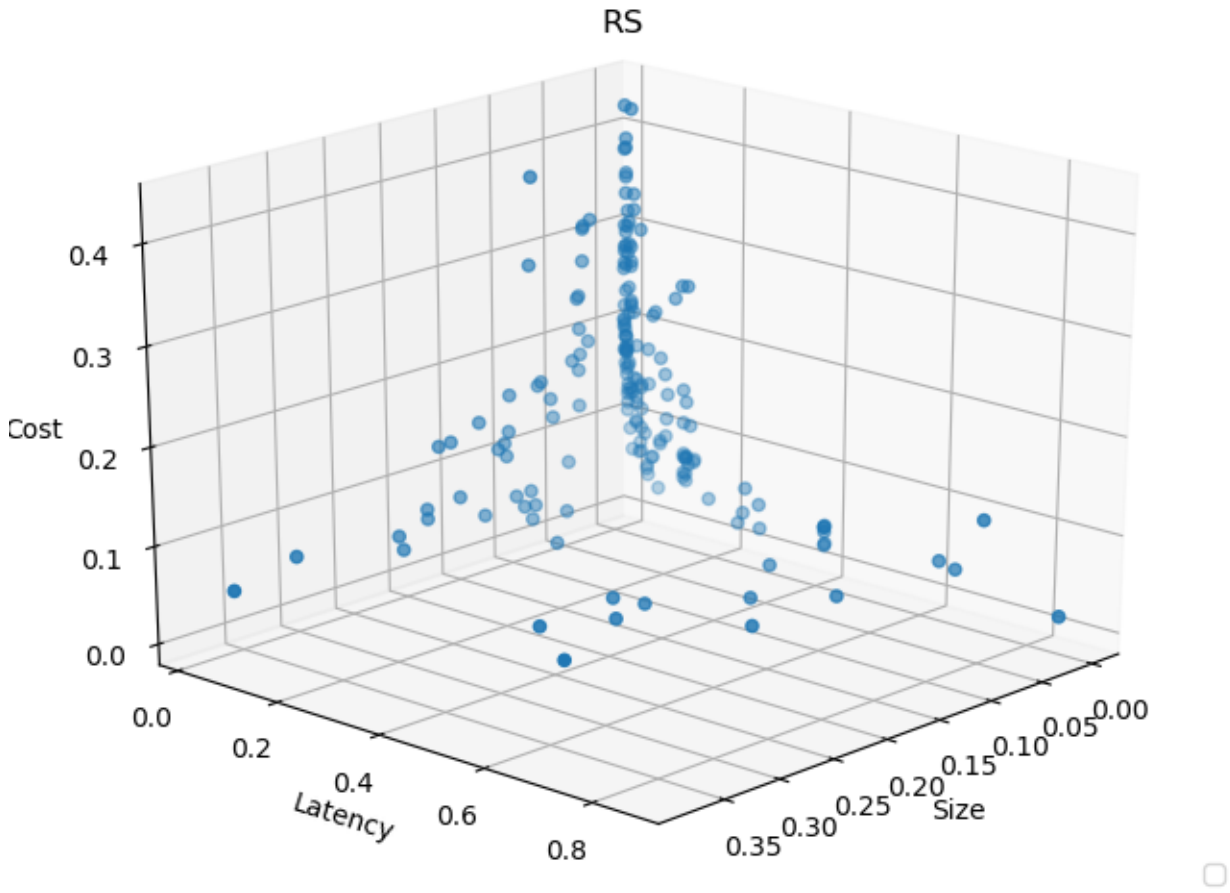

Figure 2. The aggregated front produced by Random Search.

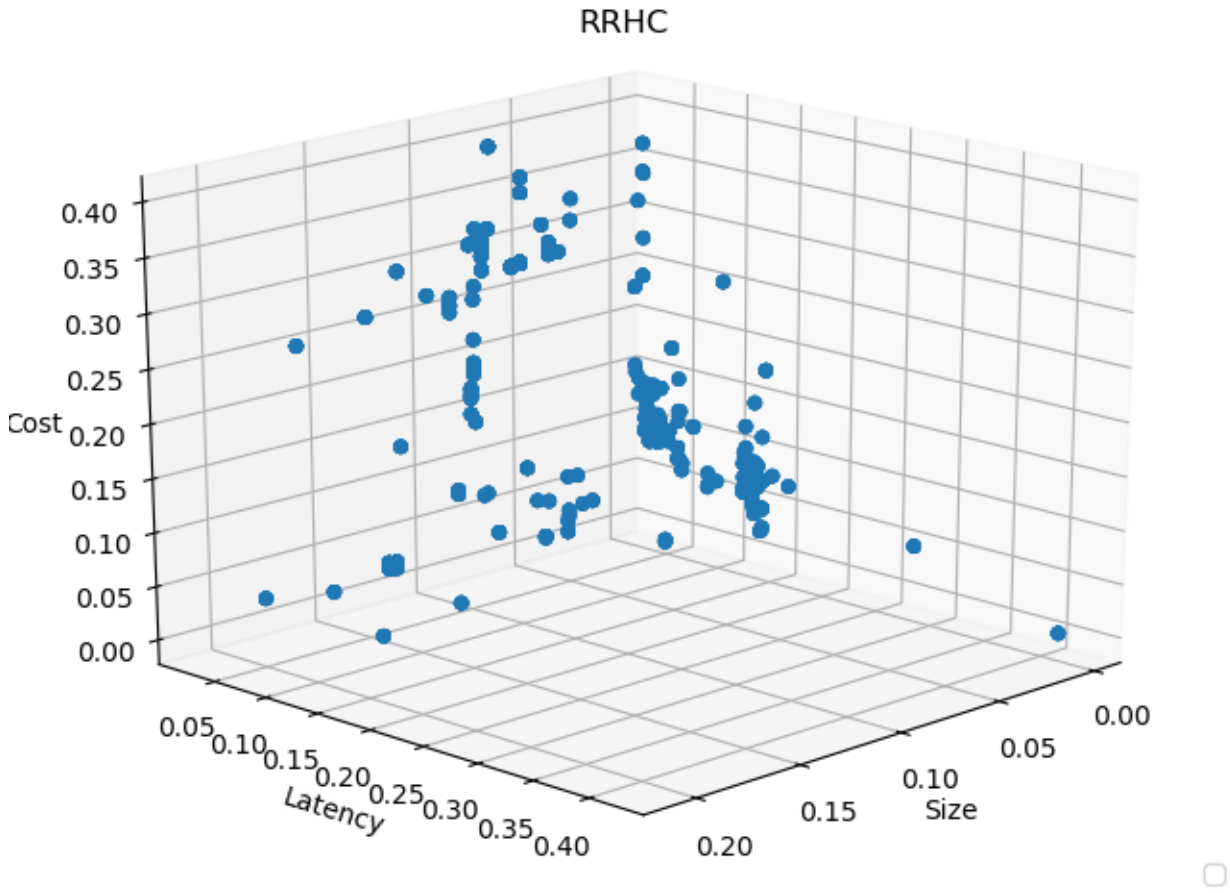

Figure 3. The aggregated front produced by Random Restart Hill Climber.

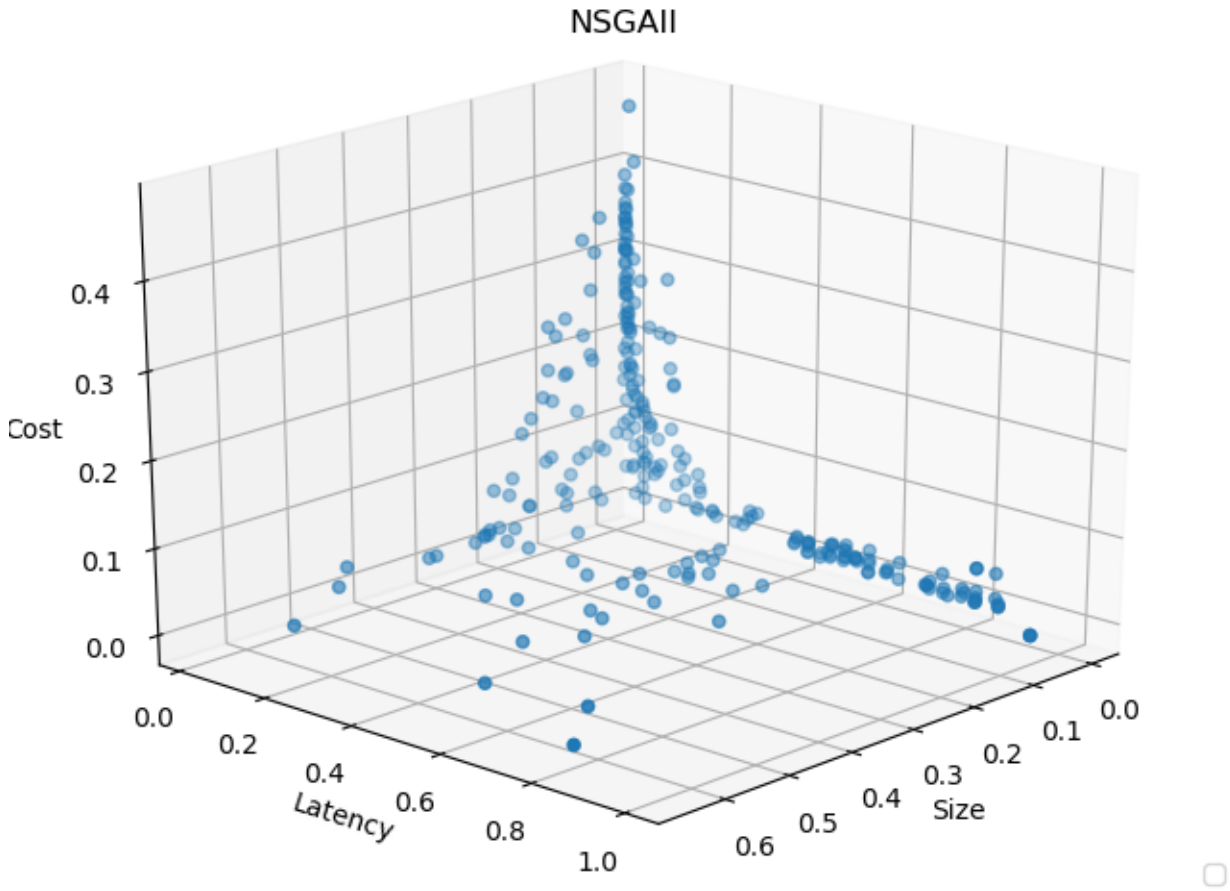

Figure 4. The aggregated front produced by NSGA-II.

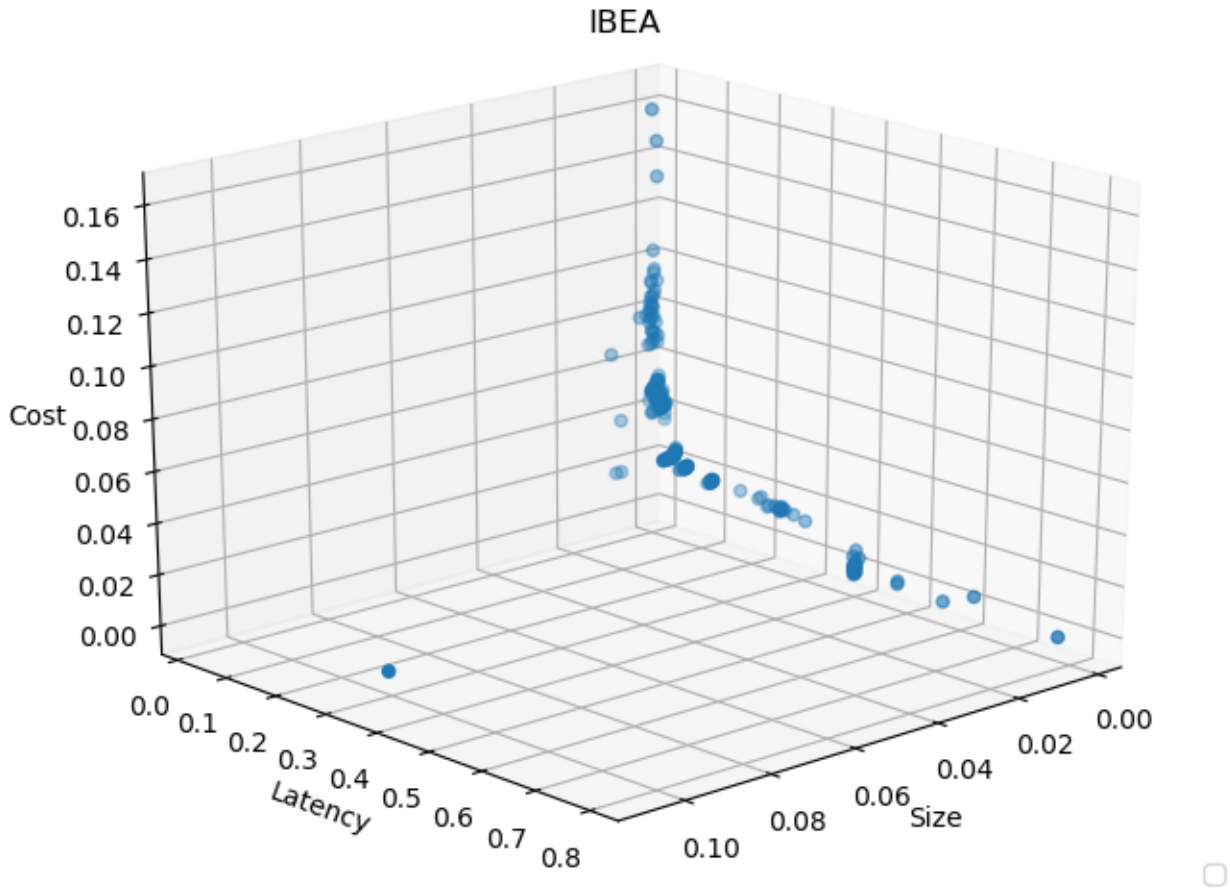

Figure 5. The aggregated front produced by IBEA.

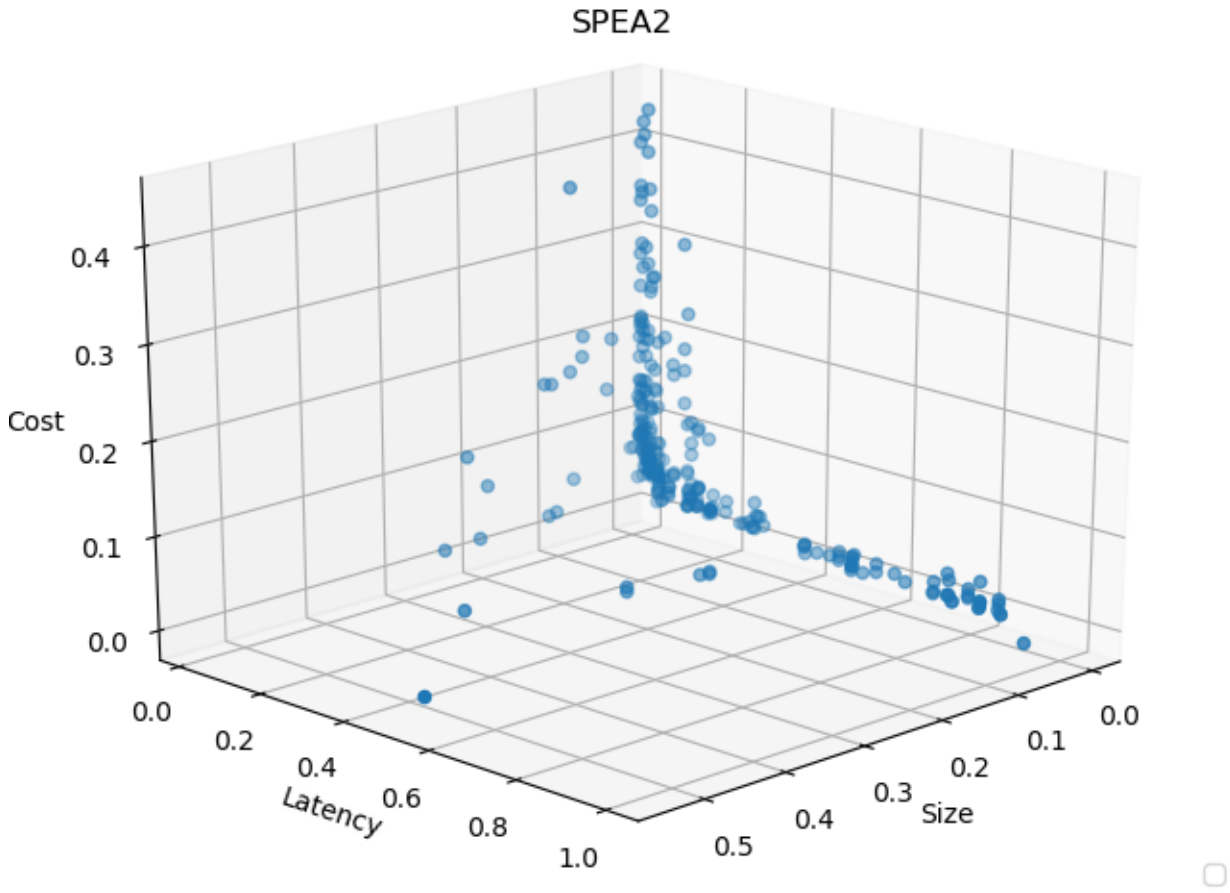

Figure 6. The aggregated front produced by SPEA2.

using the individual runs' data. Each algorithm was compared to the others n times, where n is the number of individual runs performed on each algorithm. For two Algorithms, A and B, forward comparison data from each of the four comparison methods for corresponding runs (e.g. binary hypervolume for A1 to B1, A2 to B2, etc) were used as one sample, and reverse comparison data (binary hypervolume for B1 to A1, B2 to A2, etc) were used as the second sample. Testing the data revealed that at least some of it was not normally distributed, so the non-parametric equivalent to a t-test, the Mann-Whitney U test, was used to test each pairing of runs was tested to show if there was a statistical difference in how frequently one algorithm beat the other. When this is the case, the U statistic can indicate the more reliable algorithm with respect to that particular comparison method. Table 8 summarizes the results.

4.4 Analysis of Algorithm Comparisons

Two different tests of the relative performance of algorithms were conducted. The comparisons of aggregate Pareto fronts is simply evaluated using the guidelines set out in the introduction of this chapter.

The results are somewhat mixed. From Table 6, one can being to make inferences about how the algorithms performed. It becomes immediately clear that IBEA did not explore the search space as thoroughly as the others, as evidenced by its tendency to have the smallest range for any objective, relative to the other algorithms. It is, however, quite competitive in terms of total hypervolume, so one can infer that it better exploits the areas it has explored. Likewise, NSGA-II demonstrates with its very large ranges for each objective that it is exploring the search space most effectively, but is not as aggressively exploiting the search space as IBEA seems to be. Following that line of thought, it seems that SPEA2 tends to lie somewhere between NSGA-II and IBEA in terms of both exploration and exploitation. Finally,

Table 6. Summary of aggregate Pareto fronts' characteristics (normalized data used)

Algorithm	RSOs	Hypervolume	Detection Size		Latency		Cost	
			min	max	min	max	min	max
RS	**20**	9.85e-01	0.00	0.01	0.00	0.64	0.01	0.55
	81	9.83e-01	0.00	0.01	0.00	0.73	0.00	0.56
	203	9.78e-01	0.00	0.13	0.01	0.54	0.01	0.43
	407	9.68e-01	0.00	0.27	0.01	0.56	0.01	0.47
	813	9.56e-01	0.00	0.38	0.02	0.86	0.01	0.43
RRHC	**20**	9.34e-01	0.00	0.01	0.00	0.42	0.06	0.47
	81	9.62e-01	0.00	0.01	0.00	0.19	0.03	0.34
	203	9.64e-01	0.00	0.13	0.01	0.42	0.02	0.34
	407	9.52e-01	0.00	0.13	0.01	0.50	0.03	0.34
	813	9.58e-01	0.00	0.21	0.02	0.42	0.01	0.39
NSGA-II	**20**	9.91e-01	0.00	0.05	0.00	1.0	0.0	0.48
	81	9.89e-01	0.00	0.05	0.00	1.0	0.0	0.60
	203	9.85e-01	0.00	0.27	0.01	1.0	0.0	0.49
	407	9.79e-01	0.00	0.42	0.01	1.0	0.0	0.48
	813	9.67e-01	0.00	0.65	0.02	1.0	0.0	0.47
IBEA	**20**	9.89e-01	0.00	0.00	0.00	0.70	0.00	0.19
	81	9.83e-01	0.00	0.01	0.01	0.86	0.00	0.15
	203	9.82e-01	0.00	0.01	0.01	0.62	0.00	0.21
	407	9.79e-01	0.00	0.12	0.01	0.51	0.00	0.18
	813	9.66e-01	0.00	0.11	0.02	0.79	0.00	0.16
SPEA2	**20**	9.91e-01	0.00	0.01	0.00	0.85	0.00	0.51
	81	9.89e-01	0.00	0.02	0.00	0.86	0.00	0.47
	203	9.85e-01	0.00	0.19	0.01	0.86	0.00	0.55
	407	9.78e-01	0.00	0.51	0.01	0.88	0.00	0.55
	813	9.66e-01	0.00	0.54	0.02	1.0	0.0	0.44

Table 7. Results for aggregated fronts, 813 RSOs

(a) Binary Hypervolume

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		1.67e-03	2.10e-06	7.41e-04	1.33e-07
RRHC	3.82e-03		2.79e-05	5.46e-04	5.48e-06
NSGA-II	1.10e-02	8.87e-03		1.05e-03	5.61e-04
IBEA	1.13e-02	8.99e-03	6.47e-04		8.17e-04
SPEA2	1.08e-02	9.66e-01	3.80e-04	1.04e-03	

(b) Coverage

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		0.65	0.03	0.0	0.00
RRHC	0.23		0.05	0.01	0.01
NSGA-II	0.81	0.84		0.01	0.09
IBEA	0.45	0.67	0.28		0.24
SPEA2	0.96	0.99	0.52	0.04	

(c) ϵ-Indicator

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		4.01	2.42	11.69	2.41
RRHC	1.54		1.83	10.11	1.99
NSGA-II	2.44	6.88		12.22	1.68
IBEA	1.36	2.47	1.13		1.21
SPEA2	2.02	6.88	1.48	12.77	

(d) Additive ϵ-Indicator

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		0.44	0.06	0.27	0.07
RRHC	0.08		0.05	0.23	0.08
NSGA-II	0.38	0.65		0.55	0.31
IBEA	0.00	0.37	0.00		0.00
SPEA2	0.27	0.58	0.04	0.43	

judging just by raw ranges and hypervolumes, Random Search and the Random Restart Hill Climber both seem to be exploring the search space somewhat better than IBEA and worse than SPEA2 and NSGA-II, but are exploiting the space far less effectively as IBEA. It is not clear how they relate to NSGA-II or SPEA2 in terms of exploitation. Regardless, this table is very informative but, by itself, should not be the only tool a researcher uses. If one was, for example, interested in the best exploitation available while still getting close to the extreme ends of the search space, then he or she might be able to conclude right away that SPEA2 was best for their needs, but this table does not provide enough information to objectively conclude the relationships between the algorithms tested.

Moving on to Table 7, there is a great deal of information that both confirms and expands one's understanding of how the algorithms compare. At a glance, IBEA still tends to do very well here. In fact, it does the best in all comparison methods except Coverage, where it is beaten by both SPEA2 and NSGA-II. A closer look at the

Coverage results shows that none of the algorithms covered more than a few of IBEA's solutions, while IBEA was only able to cover about a quarter of the solutions produced by the other EAs, and half to two-thirds of the solutions produced by the classical search methods. The implication here is that, as expected, IBEA did very well in a small portion of the search space, but the other algorithms had larger numbers of solutions that explored far beyond the extents of IBEA's search. Furthermore, in all cases, SPEA2 ranks higher than NSGA-II, suggesting that there may be a need to perform some parameter tuning and further compare IBEA and SPEA2, and that NSGA-II is probably not as appropriate of an EA for the problem. Finally, in a rather surprising turn of events, the Random Restart Hill Climber performed quite well in both variants of the ϵ indicator, actually out-performing all but IBEA. This suggests that the problem itself may not need such sophisticated techniques as EAs to explore the search space, and that perhaps some refinement on the hill climber, such as those described in Chapter II, would be sufficient.

Using just the comparison data in Table 7, the following relative rankings are observed:

- Binary Hypervolume
  1. IBEA
  2. SPEA2
  3. NSGA-II
  4. RRHC
  5. RS
- ϵ-Indicator
  1. IBEA
  2. RRHC
  3. SPEA2
  4. NSGA-II
  5. RS
- Coverage
  1. SPEA2
  2. NSGA-II
  3. IBEA
  4. RRHC
  5. RS
- Additive ϵ-Indicator
  1. IBEA
  2. RRHC
  3. RS
  4. SPEA2
  5. NSGA-II

The results of the Mann-Whitney tests are listed in Table 8. Though it is a slightly more complicated analysis to perform, it gives a valuable insight into which algorithms outperformed another, more often, with respect to the four comparison methods, which often did not reflect the rankings found among aggregate front comparisons. For Binary Hypervolume, none of the EAs could outperform another with significance and, overall, Random Restart Hill Climber was the most successful, outperforming all other algorithms. Random Search also outperformed all of the EAs. Coverage saw an identical performance from both classical algorithms, and IBEA was the worst performing of all algorithms. Both ϵ-Indicators saw NSGA-II in first place, and IBEA in last place. Overall, these results are somewhat of a reversal of the analysis of the aggregated Pareto fronts. While somewhat surprising, this does indicate that

Table 8. Relative reliability of algorithms as indicated by the Mann-Whitney U Test on comparison method values taken from individual runs with the 813 RSO data set. A indicates the algorithm in the left column outperformed the algorithm at the top with statistical significance, B indicates the algorithm at the top of the column outperformed the algorithm in the leftmost cell with statistical significance. ($\alpha = 0.05$)

(a) Binary Hypervolume

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		B	A	A	A
RRHC	A		A	A	A
NSGA-II	B	B		N/A	N/A
IBEA	B	B	N/A		N/A
SPEA2	B	B	N/A	N/A	

(b) Coverage

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		B	A	A	A
RRHC	A		A	A	A
NSGA-II	B	B		A	A
IBEA	B	B	B		B
SPEA2	B	B	B	A	

(c) ϵ-Indicator

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		A	N/A	A	N/A
RRHC	B		B	A	N/A
NSGA-II	N/A	A		A	A
IBEA	B	B	B		B
SPEA2	N/A	N/A	B	A	

(d) Additive ϵ-Indicator

	RS	RRHC	NSGA-II	IBEA	SPEA2
RS		A	B	A	N/A
RRHC	B		B	B	B
NSGA-II	A	A		A	A
IBEA	B	B	B		B
SPEA2	N/A	A	B	A	

some of the aggregate fronts are, indeed, over-representing some exceptional results hidden among otherwise mediocre runs. IBEA, for example, would seem to have a handful of exceptional architectures each time it runs, but not enough to outperform any of the other algorithms tested in a single run. In other words, it appears that it tends to produce somewhat erratic Pareto fronts for this problem. On the other hand, some algorithms such as NSGA-II and Random Restart Hill Climber, tended to rank similarly or better in the reliability tests when compared to the aggregate front comparisons. It would seem that the Pareto fronts produced by these algorithms are less erratic and more consistent from one run to the next. While they may not produce the rare gems that IBEA tends to, the results of any one run tend to be of high enough quality to compete strongly against the other algorithms. Again, the takeaway is that some parameter tuning may be in order. Possibly some tuning may allow IBEA or SPEA2 to more reach those high-quality solutions with greater regularity, or possibly to allow NSGA to better exploit the search space without loosing its consistency. Again, the Random Restart Hill Climber performed better than one would expect against the much more sophisticated EAs, which suggests that some refinement could elevate that algorithm without the need for all of the sophistication present in the EAs.

Using the data found in Table 8, the following rankings of reliability are observed with respect to the different comparison methods:

- Binary Hypervolume
  1. RRHC
  2. RS
  3. NSGA-II, IBEA, SPEA2 (tie)
- Coverage
  1. RRHC
  2. RS
  3. NSGA-II
  4. SPEA2
  5. IBEA
- ϵ-Indicator
  1. NSGA-II
  2. RS
  3. RRHC, SPEA2 (tie)
  4. IBEA
- Additive ϵ-Indicator
  1. NSGA-II
  2. RS, SPEA2 (tie)
  3. RRHC, IBEA (tie)

4.5 Impact of Reduced Data Sets

This section details the results of the repeating the analysis described in the previous section on each of the subsets of simulation data. The first section addresses the quality comparisons between the aggregate Pareto fronts. The following section addresses the reliability comparisons between individual runs of each algorithm. For both sections, each table gathers values for an individual comparison method across all subsets. When relating these tables back to the comparison methods, they should be read as Algorithm "A" in the left column, and Algorithm "B" in the top row.

Quality Comparisons Across Data Subsets.

Tables 9 through 12 show the results of each comparison method across all subsets of simulation data. The tables show that, though values do tend to change as the number of simulated RSOs vary, relative ranking of algorithms from one subset to the

next tends to remain consistent for each of the comparison methods. The implication here is that, though objective values obtained with smaller subsets of simulation data do not correctly indicate the expected performance of the SSN being evaluated, the *relative ranking* of algorithms using any one of the subsets tends to be representative of the relative rankings produced using the full set of simulation data. In other words, evaluating candidate algorithms against this problem can be done faster and at a lower computational cost with reasonable confidence using far fewer simulated RSOs.

Table 9. Summary of Binary Hypervolume comparisons across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	**20**		5.06e-02	9.97e-05	2.14e-03	1.18e-04
	81		2.10e-02	3.68e-06	5.87e-03	9.19e-07
	203		1.41e-02	1.61e-05	3.48e-03	4.68e-05
	407		1.70e-02	5.50e-06	5.37e-04	6.25e-07
	813		1.67e-03	2.10e-06	7.41e-04	1.33e-07
RRHC	**20**	6.59e-05		1.12e-05	1.60e-03	1.40e-07
	81	7.37e-06		1.12e-06	5.62e-03	2.98e-07
	203	2.05e-06		7.76e-06	2.86e-03	1.45e-08
	407	2.45e-04		1.36e-05	5.16e-04	8.32e-08
	813	3.82e-03		2.79e-05	5.46e-04	5.48e-06
NSGA-II	**20**	6.71e-03	5.71e-02		2.85e-03	5.26e-04
	81	6.19e-03	2.72e-02		6.85e-03	7.74e-04
	203	6.82e-03	2.09e-02		4.05e-03	7.22e-04
	407	1.04e-02	2.72e-02		9.37e-04	1.27e-03
	813	1.10e-02	8.87e-03		1.05e-03	5.61e-04
IBEA	**20**	6.69e-03	5.66e-02	7.90e-04		5.42e-04
	81	5.91e-03	2.66e-02	6.99e-04		7.21e-04
	203	7.36e-03	2.08e-02	1.12e-03		1.22e-03
	407	1.13e-02	2.81e-02	1.31e-03		1.59e-03
	813	1.13e-02	8.99e-03	6.47e-04		8.17e-04
SPEA2	**20**	6.85e-03	5.72e-02	6.55e-04	2.73e-03	
	81	5.97e-03	2.69e-02	5.59e-04	6.66e-03	
	203	6.61e-03	2.06e-02	4.87e-04	3.91e-03	
	407	9.98e-03	2.68e-02	8.31e-04	7.74e-04	
	813	1.08e-02	8.67e-03	3.80e-04	1.04e-03	

Table 10. Summary of Coverage comparisons across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	**20**		0.87	0.28	0.0	0.14
	81		0.57	0.16	0.00	0.07
	203		0.73	0.14	0.0	0.03
	407		0.41	0.04	0.00	0.02
	813		0.65	0.03	0.0	0.00
RRHC	**20**	0.38		0.04	0.0	0.06
	81	0.33		0.13	0.0	0.08
	203	0.19		0.07	0.0	0.0
	407	0.26		0.03	0.00	0.02
	813	0.23		0.05	0.01	0.01
NSGA-II	**20**	0.57	0.62		0.0	0.16
	81	0.48	0.45		0.01	0.20
	203	0.39	0.36		0.00	0.16
	407	0.57	0.60		0.0	0.15
	813	0.81	0.84		0.01	0.09
IBEA	**20**	0.09	0.32	0.26		0.31
	81	0.27	0.43	0.27		0.34
	203	0.33	0.37	0.20		0.19
	407	0.33	0.35	0.27		0.25
	813	0.45	0.67	0.28		0.24
SPEA2	**20**	0.65	0.75	0.48	0.01	
	81	0.72	0.73	0.46	0.01	
	203	0.83	0.94	0.48	0.02	
	407	0.85	0.87	0.40	0.01	
	813	0.96	0.99	0.52	0.04	

Table 11. Summary of ϵ-Indicator comparisons across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	**20**		3.01	2.15	6.78	2.14
	81		3.80	2.51	4.51	2.40
	203		3.10	1.98	17.18	2.68
	407		2.11	2.00	15.13	2.02
	813		4.01	2.42	11.69	2.41
RRHC	**20**	2.49		1.90	5.24	2.37
	81	1.57		2.16	3.26	1.94
	203	2.83		2.61	14.56	2.37
	407	1.63		2.12	10.23	2.33
	813	1.54		1.93	10.11	1.99
NSGA-II	**20**	10.94	13.43		18.56	7.53
	81	9.24	12.48		16.05	8.16
	203	2.18	3.01		20.44	5.86
	407	2.33	4.14		16.39	2.13
	813	2.44	6.88		12.21	1.68
IBEA	**20**	1.08	1.94	1.09		1.07
	81	1.30	4.47	1.31		1.30
	203	1.32	1.49	1.15		1.36
	407	1.08	1.15	1.11		1.15
	813	1.36	2.47	1.13		1.21
SPEA2	**20**	3.32	3.69	1.47	5.29	
	81	2.10	4.47	1.65	4.37	
	203	1.80	2.90	1.95	14.31	
	407	2.83	5.04	1.39	15.17	
	813	2.02	6.88	1.48	12.77	

Table 12. Summary of Additive ϵ-Indicator comparisons across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	**20**		0.22	0.07	0.35	0.03
	81		0.54	0.07	0.41	0.09
	203		0.12	0.06	0.21	0.10
	407		0.14	0.05	0.29	0.06
	813		0.01	0.44	0.06	0.07
RRHC	**20**	0.06		0.05	0.27	0.06
	81	0.03		0.05	0.19	0.04
	203	0.04		0.05	0.13	0.08
	407	0.03		0.03	0.16	0.37
	813	0.08		0.05	0.23	0.08
NSGA-II	**20**	0.36	0.58		0.30	0.15
	81	0.27	0.81		0.44	0.14
	203	0.46	0.58		0.38	0.14
	407	0.44	0.50		0.49	0.12
	813	0.39	0.65		0.55	0.31
IBEA	**20**	0.05	0.27	0.00		0.00
	81	0.13	0.67	0.01		0.00
	203	0.08	0.20	0.00		0.00
	407	0.00	0.02	0.00		0.00
	813	0.00	0.38	0.00		0.00
SPEA2	**20**	0.20	0.42	0.05	0.32	
	81	0.13	0.67	0.05	0.31	
	203	0.32	0.44	0.08	0.34	
	407	0.33	0.41	0.09	0.39	
	813	0.27	0.58	0.04	0.43	

Reliability Analysis Across Data Subsets.

Tables 13 through 16 show relative reliability across all subsets of simulation data. With the exception of ϵ-Indicator for IBEA and random restart hill climber in the 81-RSO data set, reliability was found to be consistent in all cases where significance was found for all comparison methods. This clearly demonstrates that, for this problem, the *relative quality* of solutions produced by algorithms does not tend to vary based on the fidelity of the data. Again, the raw objective values were affected by the use of fewer RSOs in the simulations, but the rate at which algorithms out performed one another as indicated by the binary comparison methods did not tend to change. This further confirms that, when comparing the performance of MOO algorithms on this problem, one will tend to receive the same relative rankings, even when using as little as 2.5% of the full complement of RSOs in the simulations.

4.6 Chapter Summary

This chapter presents the results of the computational experiments described in Chapter III. First, binary comparisons are performed between each of the five algorithms using the full simulation data set. Relative rankings of the algorithms are not consistent between the binary comparison methods, confirming that the comparisons tend to reward different aspects of the compared Pareto fronts. Overall, the comparison of aggregate Pareto fronts indicates that IBEA and SPEA2 tended to be the best-performing algorithms on this problem.

Next, individual runs are compared to determine overall reliability of the algorithms, relative to one another. The results do not align well with the aggregate comparison results, with IBEA and SPEA2 performing poorly in terms of reliability, and NSGA-II and Random Restart Hill Climber performing as good or better than in the aggregate comparisons. This suggests that IBEA and SPEA2 tend to produce

Table 13. Summary of Binary Hypervolume reliability analysis across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	20		B	N/A	N/A	A
	81		B	A	N/A	A
	203		B	A	A	A
	407		B	A	A	A
	813		B	A	A	A
RRHC	20	A		A	A	A
	81	A		A	A	A
	203	A		A	A	A
	407	A		A	A	A
	813	A		A	A	A
NSGA-II	20	N/A	B		N/A	N/A
	81	B	B		B	N/A
	203	B	B		B	N/A
	407	B	B		N/A	N/A
	813	B	B		N/A	N/A
IBEA	20	N/A	B	N/A		A
	81	N/A	B	A		A
	203	B	B	A		N/A
	407	B	B	N/A		N/A
	813	B	B	N/A		N/A
SPEA2	20	B	B	N/A	B	
	81	B	B	N/A	B	
	203	B	B	N/A	N/A	
	407	B	B	N/A	N/A	
	813	B	B	N/A	N/A	

Table 14. Summary of Coverage reliability analysis across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	20		B	N/A	A	A
	81		B	A	A	A
	203		B	A	A	A
	407		N/A	N/A	A	A
	813		B	A	A	A
RRHC	20	A		N/A	A	A
	81	A		A	A	A
	203	A		N/A	A	A
	407	N/A		A	A	A
	813	A		A	A	A
NSGA-II	20	N/A	N/A		A	A
	81	B	B		A	A
	203	B	N/A		A	A
	407	B	B		A	A
	813	B	B		A	A
IBEA	20	B	B	B		B
	81	B	B	B		B
	203	B	B	B		B
	407	B	B	B		B
	813	B	B	B		B
SPEA2	20	B	B	B	A	
	81	B	B	B	A	
	203	B	B	B	A	
	407	B	B	B	A	
	813	B	B	B	A	

Table 15. Summary of ϵ-Indicator reliability analysis across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	**20**		A	N/A	A	N/A
	81		A	N/A	A	N/A
	203		A	N/A	A	N/A
	407		A	N/A	A	N/A
	813		A	N/A	A	N/A
RRHC	**20**	B		B	N/A	N/A
	81	B		B	B	B
	203	B		B	A	N/A
	407	B		B	A	N/A
	813	B		B	A	N/A
NSGA-II	**20**	N/A	A		A	N/A
	81	N/A	A		A	N/A
	203	N/A	A		A	N/A
	407	N/A	A		A	N/A
	813	N/A	A		A	A
IBEA	**20**	B	N/A	B		B
	81	B	A	B		B
	203	B	B	B		B
	407	B	B	B		B
	813	B	B	B		B
SPEA2	**20**	N/A	N/A	N/A	A	
	81	N/A	A	N/A	A	
	203	N/A	N/A	N/A	A	
	407	N/A	N/A	N/A	A	
	813	N/A	N/A	B	A	

Table 16. Summary of Additive ϵ-Indicator reliability analysis across all data subsets

Algorithm	RSOs	RS	RRHC	NSGA-II	IBEA	SPEA2
RS	20		A	B	A	N/A
	81		A	B	N/A	N/A
	203		A	B	A	N/A
	407		A	B	A	B
	813		A	B	A	N/A
RRHC	20	B		B	N/A	B
	81	B		B	B	B
	203	B		B	N/A	B
	407	B		B	N/A	B
	813	B		B	B	B
NSGA-II	20	A	A		A	N/A
	81	A	A		A	N/A
	203	A	A		A	N/A
	407	A	A		A	N/A
	813	A	A		A	A
IBEA	20	B	N/A	B		B
	81	N/A	A	B		B
	203	B	N/A	B		B
	407	B	N/A	B		B
	813	B	B	B		B
SPEA2	20	N/A	A	N/A	A	
	81	N/A	A	N/A	A	
	203	N/A	A	N/A	A	
	407	A	A	N/A	A	
	813	N/A	A	B	A	

somewhat mediocre Pareto fronts with a few exceptional solutions scattered across them, while NSGA-II tends to produce relatively consistent Pareto fronts from one run to the next. The net result being that the few exceptional solutions, when aggregated, overstated the typical performance of IBEA and SPEA2, while the lack of unusually good solutions tends to rank NSGA-II and Random Restart Hill Climber lower against their more erratic competition.

Finally, aggregate fronts and reliability are evaluated across the different simulation data subsets. In both cases, the data supports the hypothesis that using fewer RSOs does not affect the relative performance of the algorithms. While there was some variation between subsets, the tendency is for relative rankings between algorithms to be consistent from one subset to the next.

V. Conclusion

5.1 Chapter Overview

This chapter summarizes the outcomes of this research. The findings are discussed through the lens of the research questions and hypotheses stated in Chapter I. Specific contributions are identified, and recommendations for future research are listed.

5.2 Research Questions Answered

This section lists the research questions are with the respective findings.

1 Which of the representative algorithms is (are) most effective?

The answer to this question largely depends on what one wants the algorithm to do. Each of the comparison methods highlights different characteristics of a Pareto front, and therefore are more sensitive to different aspects of an algorithm's behavior. This resulted in inconsistent ranking from one comparison method to the next. Taken at face value, IBEA was ranked as best for three of the four comparison methods, and SPEA2 ranked second best, when comparing aggregate Pareto fronts.

Taking into consideration raw metrics about the Pareto fronts (range of objective values and individual hypervolume), it becomes clear that IBEA is exploring far less of the search space than its competitors, and making up for it by very aggressively exploiting what little search space it does explore to get the best possible answers. NSGA-II was shown to most effectively explore the search space, and SPEA2 explored nearly as well. The classical algorithms were both generally mediocre in terms of exploration.

Finally, incorporating the results of the reliability analysis further complicates the issue by revealing that IBEA and SPEA2 are the least and second least reliable algorithms, respectively, meaning that neither was able to reliably outperform their competitors when comparing individual optimization runs. Meanwhile, NSGA-II tied for most reliable with Random Search and Random Restart Hill Climber. Taken as a whole, these pros and cons create a situation where the degree to which a researcher seeks exploitation versus exploration could make any one of the EAs the most effective for his or her purposes.

Hypotheses 1 through 3 directly relate to this research question:

- H1 - For each pair of algorithms, one will tend to produce better Pareto fronts than the other.

  Confirmed. Comparing aggregate Pareto fronts revealed a difference in all cases with each comparison method.

- H2 - The Pareto fronts produced by random search tend to be worse than those of all remaining algorithms.

  Confirmed. Considering only the results of the comparison methods with the aggregated Pareto fronts produced using the 813-RSO data set, Random Search ranked the worst in three of the four comparison methods.

- H3 - Each evolutionary algorithm tends to produce better Pareto fronts than a random-restart hill climber.

  Not Confirmed. Considering only the results of the comparison methods with the aggregated Pareto fronts produced with the 813-RSO data set, and summing the algorithms' ranks across the four comparison methods reveals that Random Restart Hill Climber ranked slightly better than NSGA-II

2 What useful insights are provided by various means of comparing the results of the algorithms?

This question is somewhat subjective and, as a result, receives a somewhat subjective answer. Usefulness is not easily quantified and each provided some insight that the others could not. Binary Hypervolume was the only comparison method that gave a hint that some algorithms explored the search space more effectively than others, which is an important consideration in large-scale problems such as this. Coverage provides insight into how much of the solution set is dominated by a competing algorithm, but gives no indication to the magnitude of the differences. It is also vulnerable to overstating the quality of a rather mediocre set with a few exceptional solutions. Conversely, the ϵ-Indicator provides a quantification of the difference between solution sets, but does not provide any way of knowing how much of one solution set is dominated by the other if the sets are incomparable. The Additive ϵ-Indicator was more sensitive to differences in scale between objectives, which necessitated the normalization of objective values. This measure differs from the standard ϵ-Indicator in that it is a translation of the Pareto front instead of a scaling, so it is a more direct comparison of the fronts. In problems such as this one where the shapes of fronts being compared are quite different, this measure may be preferred over the standard ϵ-Indicator. Ultimately, Binary Hypervolume, coverage, and either ϵ-Indicator worked together to provide a multifaceted picture of the relationship between algorithms, though the use of both ϵ-Indicators tended to bring more confusion than clarity to the situation.

3 What is the impact of using fewer simulated RSOs on the quality of solutions produced by these algorithms?

The impact of using fewer simulated RSOs is observable in two major ways.

First is its effect on the relative performance of search algorithms, and second is its effect on objective values obtained when evaluating candidate architectures. With regard to the latter, the number of simulated RSOs directly affects the objective values produced by the evaluation functions, resulting in objective values that do not accurately indicate the real-world performance that one could expect from a given architecture. However, the effects of using smaller simulation data sets on the effectiveness of an algorithm has much subtler effects. In general, the preferred algorithms, based on the results of the comparison methods, were fairly consistent from one data set to the next. While there were instances where relative rankings varied between simulation data subsets, the overall trend was for rankings to remain consistent across subsets. Furthermore, in the cases where significance was found, reliability analysis were consistent across data subsets in all but one instance.

Hypothesis 4 directly relates to this research question:

- H4 - Simulating fewer RSOs does not tend to change the relative effectiveness of the algorithms.

  Confirmed. Taken together, the comparisons of aggregate Pareto fronts and the reliability analysis strongly suggests that comparing algorithms with a reduced number of RSOs tends to produce relative rankings that are consistent with those obtained with the full simulation data.

5.3 Future Work

This section discusses a number of areas where this research could be improved or extended. Much like the research, the focus in this section is on the computational aspects of the problem. The authors of the underlying model addressed twelve recommendations for future work which were not addressed here, and still warrant

further exploration.

- Incorporate other measures of algorithm performance.

  The four comparison methods used in this research are not the only measures of algorithm performance. Other factors that capture other practical aspects of an algorithm's performance, such as convergence time for EAs, could be incorporated into this methodology. While each comparison method added is another opportunity for a ranking that does not agree with the other methods, it is also another tool for a researcher to make an educated choice of algorithm.

- Explore relationship between objective values and the number of RSOs.

  It was noted that the Latency and Detection Size objective values were impacted by using fewer RSOs in the simulation data, but those changes were not thoroughly analyzed in this research. Evaluating a large, pre-defined set of architectures on each of the data subsets may reveal a predictable relationship between the number of RSOs and the change to objective values. If such a relationship did exist, it would be possible to estimate architecture performance using the smaller subsets. While it would not be a substitute for performing search on the best data available, it could be a useful tool for initial searches or for quickly testing modifications to the underlying model.

- Perform parameter tuning for these algorithms with this problem. Includes performing further trials with more generations.

  Each of the EAs evaluated in this research showed promise, but each also had problems with their performance, relative to the others. It is a near guaranteed that performing even modest parameter tuning on each would render improvements to their overall effectiveness on this problem, and potentially revealing one algorithm to be a better overall choice with fewer compromises.

- Compare algorithms of different types.

  There are many types of multi-objective search algorithms that could be applied to this problem. This research compares relatively similar algorithms, but there is great value in understanding the performance of algorithms that use very different techniques. Some examples of successful algorithms that might be useful are swarm-based searches, such as Ant Colony Optimization [59]; Simulated Annealing [60], which is modeled explicitly after the molecular behavior observed in the annealing process; and Evolution Strategies [18], which use self-adaptation to tune the parameters of the algorithm as it runs, but also to evolve better parameters for the algorithm as it runs.

- Incorporate preferences into tools.

  The tools developed for this research do not currently address preferences in the search. Development of a built-in capability to identify candidate preference schemes and incorporate them into the search process could result in software tools with greater real-world utility.

- Explore finer discretization in optimizations or simulations where appropriate.

  In an effort to reduce computational cost, the underlying model was rather coarsely discretized. Taking advantage of the improvements to evaluation time gleaned through the performance tuning of the Stern and Wachtel methodology, it may be possible to discretize the search space more finely, or even convert some discrete variables to continuous variables. Changes to certain variables, such as constellation altitudes and GBT locations, would require new simulations, but other values such as aperture size could be changed and applied to the problem using existing data.

- Assess of the accuracy of the model.

This research focused exclusively on the comparison of multi-objective search algorithms applied to the SSN architecture design problem. It did not assess the model for accuracy, and the model's creators have pointed out areas where simplifications will degrade accuracy [9]. A thorough assessment of the model's accuracy and application of enhancements to address the findings could render valuable improvements to the model and its applications.

- Incorporation of domain-specific knowledge into search.

  This research effort assumed that little was known of the search space. In reality, there is a great deal of knowledge regarding space surveillance networks. There could be great gains realized by incorporating this domain-specific knowledge into the search algorithms.

5.4 Chapter Summary

GEO SSA is an important issue for both national and civilian interests. The search space for this problem is enormous, and the search is computationally expensive. A method of comparing the effectiveness of candidate algorithms at a relatively low computational cost on this problem is highly desirable to ensure that the highest quality solutions to this multi-billion dollar problem can be obtained in a practical time and computational cost. This research shows that combining multiple binary comparison methods provides a multifaceted picture of the relative effectiveness of two or more multi-objective algorithms. It further shows how fewer simulated RSOs can be used to perform this comparison at a low computational cost, relative to using a full complement of simulated RSOs. Incorporating the techniques described in this methodology into current space system engineering will undoubtedly improve quality, speed, and efficiency of future expansions to the current space surveillance networks.

Appendices

METHODOLOGY FOR COMPARISON OF ALGORITHMS FOR REAL-WORLD MULTI-OBJECTIVE OPTIMIZATION PROBLEMS: SPACE SURVEILLANCE NETWORK DESIGN

A. Relationships Among Objective Functions

The objective functions in for this problem are not independent, meaning that changes made to improve one objective will worsen another. This phenomenon was described in general terms in Chapter I. Here, a specific example is presented. A solution is selected at random from the aggregated Pareto front produced by IBEA using the full simulation data set. Each allele is then mutated one step in both directions (where possible) and reevaluated. The resulting fitness values are compared to the original to demonstrate the way each allele affects multiple objectives. Table 17 lists each architecture compared and Table 18 holds the respective objective values. The objective values have been color coded where green is an improvement over the base architecture, and red is a deterioration. There are two cases where there is no impact on any of the objective values, and four (lines 12, 15, 18, and 23) that is actually an overall improvement over the base architecture. As an aside to this discussion, these four examples demonstrate that this example, though produced by one of the most successful algorithms evaluated in this research, is not on the True Pareto Front. The remaining 23 architectures demonstrate that, in general, improvements to one objective gained by changing a single allele come at the detriment of one or more other obejctives.

Table 17. List of architectures compared to demonstrate that objective functions are not independent for this problem. The first is the starting architecture, and each subsequent architecture is a variant on the original in which just one allele is mutated one step in either direction.

Architecture	# Grnd Telescopes, Diego Garcia	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Haleakala, HI	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, La Palma	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Mauna Kea, HI	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Indian Astro. Obs.	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Mount Graham, AZ	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Paranal	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Siding Spring	Grnd Telescope Aperture Dia. (m)	# Grnd Telescopes, Socorro, NM	Grnd Telescope Aperture Dia. (m)	LEO Sun-Synch Altitude (km)	LEO Sun-Synch Sats per Plane	LEO Sun-Synch Planes	LEO Sun-Synch Aperture Dia. (m)	LEO Equatorial Altitude (km)	LEO Equatorial Number of Sats	LEO Equatorial Diameter (m)	Near-GEO Observer Alt. (km from GEO)	Near-GEO Observer Number of Sats	Near-GEO Observer Aperture Dia. (m)
0	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
1	1	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
2	0	1	1	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
3	0	1	0	2	0	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
4	0	1	0	2	2	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
5	0	1	0	2	1	2.5	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
6	0	1	0	2	1	3.5	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
7	0	1	0	2	1	3	1	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
8	0	1	0	2	1	3	0	1	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
9	0	1	0	2	1	3	0	2	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
10	0	1	0	2	1	3	0	1.5	1	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
11	0	1	0	2	1	3	0	1.5	0	4	1	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
12	0	1	0	2	1	3	0	1.5	0	4	0	4	3	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
13	0	1	0	2	1	3	0	1.5	0	4	0	4	4	1	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
14	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	1	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
15	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	3	1	800	2	2	0.15	1000	4	0.15	42664	4	0.15
16	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	0.5	800	2	2	0.15	1000	4	0.15	42664	4	0.15
17	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1.5	800	2	2	0.15	1000	4	0.15	42664	4	0.15
18	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	700	2	2	0.15	1000	4	0.15	42664	4	0.15
19	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	900	2	2	0.15	1000	4	0.15	42664	4	0.15
20	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	1	2	0.15	1000	4	0.15	42664	4	0.15
21	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	1	0.15	1000	4	0.15	42664	4	0.15
22	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.3	1000	4	0.15	42664	4	0.15
23	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	900	4	0.15	42664	4	0.15
24	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	3	0.15	42664	4	0.15
25	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.3	42664	4	0.15
26	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	41664	4	0.15
27	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	43164	4	0.15
28	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	3	0.15
29	0	1	0	2	1	3	0	1.5	0	4	0	4	4	0.5	0	3.5	4	1	800	2	2	0.15	1000	4	0.15	42664	4	0.3

Table 18. Fitness values for each evaluated architecture. Line 0 holds the objective values of the base architecture, and is the standard to which all other lines are compared. Green font indicates an improvement and red font indicates a deterioration in an objective value, relative to line 0.

Architecture	Detection Size (cm)	Latency (min)	Cost ($100M)
0	63.59118036	34.01414514	16.00711229
1	63.64952302	34.01230012	16.12711229
2	61.93594264	34.01353014	16.66281161
3	78.16977623	34.01414514	14.23648024
4	63.59118036	34.01414514	17.60068113
5	64.0460524	34.01414514	15.36923075
6	63.24121935	34.01414514	16.8196203
7	62.62447993	34.01353014	16.33115692
8	63.59118036	34.01414514	16.00711229
9	63.59118036	34.01414514	16.00711229
10	1055.82302	34.01230012	19.58995897
11	55.06657121	34.01414514	19.58995897
12	63.59118036	34.01414514	15.98857162
13	56.78504642	34.01414514	16.36103205
14	1776.878302	34.01353014	18.59025235
15	63.59118036	34.01414514	15.90580311
16	67.24683506	34.01414514	15.65319253
17	62.0975765	34.01414514	16.74371341
18	63.56365627	34.00861009	16.00711229
19	63.56590575	34.01722017	16.00711229
20	62.85847269	40.98216482	14.01711229
21	62.83052785	40.98093481	14.98111229
22	63.3966146	34.01414514	18.57311229
23	63.55778121	34.0104551	16.00711229
24	63.39222233	37.0301353	15.50056643
25	63.53327831	34.01414514	18.17311229
26	62.07059898	34.01537515	16.00711229
27	64.42856199	34.01660517	16.00711229
28	54.28705137	37.19372694	15.50056643
29	50.57749216	34.01414514	18.17311229

B. Performance Tuning

Introduction

Section 2.7 outlines several performance and overhead concerns in the original algorithm. Before optimizers could be evaluated, performance tuning was required to mitigate those problems. Initial evaluation identified three key areas where inefficiencies were likely to negatively impact performance, which are listed in Section B. Informed by those key areas, performance tuning had three primary objectives: 1) move to the Object Model (OM), 2) consolidate data into a few large files, and 3) eliminate the need for data parsing in the optimizer. Of these, converting to the OM was the most difficult, as it required most *connect* commands in the existing code to be identified and converted to the OM equivalent. It also required moving from a Linux environment to Windows as the object model for Linux is only available through a Java API, and would have required an unacceptable investment of resources to port the code to another programming language.

Interestingly, the other two stages were solved simultaneously with a single change made possible by moving to the OM. With raw data exposed to Python script, it was possible to perform simulations of entire sensor classes (i.e. GBT, sun-synchronous, etc), retrieve data directly, perform checks for validity, consolidate results into class-specific data structures, and then save those data structures to disk. Data was saved using Python's *pickle* serialization module which saves entire data structures to binary files, and restores them in their original form when "unpickling" the files back into memory. This greatly improved performance of the optimizer because data were supplied to the optimizer in the correct types, completely removing the need for string manipulation in the optimization stage. During this performance tuning, it was found that the simulations of space-based telescopes were simulating duplicate sensors; 40%

of the data produced for these classes were duplicates due to this minor error. The move to this sensor-class-based simulation strategy allowed these simulations to be merged together, eliminating the excess computational load.

The final result of this tuning was a complete overhaul to the way data is managed. In the original algorithm, 77 parallel simulations, each simulating a single class and configuration of sensors and the 813 RSOs in geosynchronous orbits, would be run on an HPC to gather three text files of key data for each sensor/target pair. These files would be consumed by the optimizer as needed, creating a file I/O bottleneck. The new data flow consists of the Python program running four simulations on a desktop workstation, harvesting data directly from STK. It would use the correct data types to store the collections of key data from each pair in custom data structures and *pickle* (save) each dictionary to a binary file. As is often the case, this was a trade-off in which the size of the data files increased to a total of about 16 gigabytes, but the number disk accesses required to generate and evaluate the data was reduced from millions per simulated time frame to just eight.

Modification Strategy

While not the focus of this research, the original code required dramatic changes to reduce the file I/O burden previously discussed. The general modification strategy was to make the fewest changes possible to mitigate overhead and facilitate drop in replacement of the optimizer. The goal was not to squeeze every ounce of performance out the existing code, but rather to limit modifications to addressing performance issues at algorithm level and enabling efficient testing. This included refactoring where necessary to facilitate evaluation and modification, streamlining the data flow as described in the next section, and modifications to decouple the objective functions' code from the optimizer code. It did not include low-level changes, such as the choice

of programming language, one library over another, or any other factor that falls into the area of preference or coding style unless absolutely necessary, as these are factors that are independent of the algorithm design. These limitations resulted in an overall reorganization of code and an overhaul of the data management, but virtually no changes to the core code implementing the model and equations described in Stern and Wachtel's work.

One noteworthy exception to this is an overall update of the codebase to Python 3. The initial codebase was written in Python 2, which will be within six months of its End of Life date [61] by the conclusion of this research. This seemed most appropriate as the necessary updates were relatively minor, and would be beneficial to any future researchers.

Key Areas

The following are areas of the original algorithm where improvements could potentially be made. There is some overlap between some of the areas, but each represent some aspect of potential degradation of overall performance.

STK Interaction.

The previous research used the *connect* interface to interact with STK. This interface relies on building a series of commands as an array of strings which are supplied to the engine sequentially via a network connection, or a loopback connection for local execution. This interface allows users to configure an entire simulation, run it, and command STK to save the results to a file, however does not allow for interactive features, such as retrieving results directly from the engine. An alternative is the Object Model in which the simulation is treated as an object in the calling program. This model allows for an interactive simulation in which values are available directly

to the calling program, and dynamic behaviors based on results. This is a much more powerful interface, and has potential to be much more efficient, overall, than the *connect* interface. In particular, it could be leveraged to reduce the overall reliance on file I/O.

File I/O.

File I/O is a major concern for the efficiency of this algorithm. When compared to random access memory (RAM) or cache, disk access is among the slowest ways to retrieve stored data in typical computing scenarios. According to [53], an average memory reference will incur about 100nS of latency, while a solid state drive (SSD) random read requires 160 times that at 16μS, and a hard disk drive (HDD) seek incurs 30,000 times greater latency, or 3mS. The current algorithm constructs, stores to disk, reads in, and evaluates nearly 188,000 text files per simulation, all before handing control over to the optimizer.

Once the optimizer takes over, file I/O actually becomes a larger issue. Since each individual would be evaluated in isolation, none of the raw data was stored in memory for reuse. Instead, each architecture evaluation would read in all of the files necessary to represent each sensor/target pairs represented in the particular architecture being evaluated. At three files per pair, and 813 target satellites, each sensor in an architecture would require 2,439 files to be read in. In the worst case, where there are telescopes at all nine of the ground stations, four equatorial observer satellites, four sun-synchronous observer satellites, and four near-GEO observer satellites, a single architecture would require 51,219 files. Recall also that there are multiple altitudes possible for each class of observer satellite, so there are many unique worst-case combinations possible. This means that a single generation could, in the worst case of 96 worst-case architectures, require 4,917,024 files (more than thirteen times

the original number created) to be read in and parsed. The data generated in the simulation phase amounts to over six gigabytes of text data, but that data would be repeatedly ingested by the optimizer many times over the course of the optimization, easily amounting to hundreds of gigabytes of data worth of file I/O to evaluate some subset of a six gigabyte data set.

Data Representation.

Though the simulation data consists exclusively of numbers (floats) and time stamps, all values were stored as text and parsed as needed. There is an incredible amount of string manipulation taking place in a typical run of the optimizer. Recalling the number of files being read in for a worst case scenario, a worst-case file (one for a sensor/target pair for which there was continuous access), there would be 11,524 strings to be parsed by the optimizer. While parsing text to some other type calls for a relatively minor amount of computational power, this is happening many millions of times in a run of the optimizer. For example, reading in the data for the pairing of GBT #4 and target #11 (by no means a worst-case pairing) requires the parsing of 22,305 individual strings into their respective data types. Furthermore, through profiling on an interim version which was using data stored in memory instead of files, it was found that parsing time stamp strings to a *datetime* object is an exceptionally expensive computation, representing as much as 80% of the computation performed by the optimizer. Storing and processing data in native data types was a key target for reducing unnecessary overhead.

C. User's Guide

HPC Tips and Tricks

Determine Python Version.

At the command line, run the *python* command. This opens Python's interactive interpreter. The first lines displayed should start with the current version of Python and will look like the following:

```
[your_username@mustang08 ~ ]$ python
Python 2.7.5 (default, May 31 2018, 09:41:32)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Getting Python3.

Python 3 is required to run the code produced for this research. If Python 2.7 is the current version, Python 3 will need to be loaded. Luckily it is a very simple thing to do. Simply run the *module load python3* command. Much of this code uses Message Passing Interface (MPI), so you will also want to load the mpi4py library by running the *module load mpi4py* command. This is not a permanent change, and will need to be accomplished every time you log in. A permanent solution is to edit your *.personal.bashrc* file to include those commands.
On systems using the Common Open Source Tools (COST) module (e.g. Thunder), first run *module load costinit*, then load Python modules as above.

Installing Python Modules.

Some modules may not be readily available. One notable example is the Platypus library. There are two ways of getting it. With Python3 already loaded, Platypus is available via the *pip* command by running ***pip install --user Platypus-Opt*** from the webshell. Note the use of the *--user* option. The installation will fail without it due to lack of permissions to install to the Python directories. At the time of writing, the version available via pip is 1.0.3. This version has two bugs that are relevant to the optimization code, that have been fixed in the version available via GitHub. To get this version, follow this procedure:

1 Download the latest version from *https://github.com/Project-Platypus/Platypus* as a zip file.

2 In your home directory on the HPC, create a folder named "Platypus" and upload the zip file to this folder. (Answer yes when prompted to expand it, or manually expand it if needed.)

3 Using the webshell, navigate to the Platypus directory you just created.

4 Enter ***python setup.py develop --user*** to install the library locally. Again, note the use of the double dash with the user flag.

Running serial and parallel evaluations.

Parallel uses MPI, requires MS MPI to run at home. Spawning processes will cause Python to fail in interactive mode. MPI can't be debugged with a traditional debugger.

Submitting PBS Jobs.

This research relied heavily on PBS to schedule jobs on the HPC. What follows is an example of a PBS script that was used for this research effort. This is not intended to be a full tutorial on using PBS or shell scripting, but just a template specific to this research. Anyone wishing to extend this research should familiarize themselves with the PBS guide associated with the HPC he or she will use.

Submitting a script is done by creating a valid .pbs job script and then entering the `qsub your_job_script.pbs` command. A typical script looks like the following:

```
#!/bin/bash
## Required Directives ------------------------------------
#PBS -l select=8:ncpus=48:mpiprocs=26:bigmem=1
#PBS -l walltime=85:00:00
#PBS -q standard
#PBS -A <YOUR_PROJECT_ID>

## Optional Directives ------------------------------------
#PBS -N 813_EAs
#PBS -j oe
#PBS -M <YOUR_EMAIL_ADDRESS>
#PBS -m bae

## Environment Setup --------------------------------------
JOBID=‘echo ${PBS_JOBID} | cut -d ’.’ -f 1‘
# change directory to job-specific directory within scratch
# directory in /p/work1
cd ${JOBDIR}
```

```
# FIRST DATA SUBSET
# stage input data $HOME
cp ${HOME}/thesis/813_tgts_77/*.res .

# copy the executable from $HOME
cp ${HOME}/thesis/inspyred_mpi.py .
cp ${HOME}/thesis/best_arch_platypus_parallel_77.py .
cp ${HOME}/thesis/clearSky.py .
cp ${HOME}/thesis/thesis_classes.py .

## Execution ----------------------------------------------
module load python3
module load mpi4py
mpiexec_mpt -n ${BC_MPI_TASKS_ALLOC} python ./best_arch_platypus_parallel_77.py
    -c 3 -t 813 -a NSGAII -p 100 -e 25000 > nsga_813_77_output.out
mpiexec_mpt -n ${BC_MPI_TASKS_ALLOC} python ./best_arch_platypus_parallel_77.py
    -c 3 -t 813 -a IBEA -p 100 -e 25000 > ibea_813_77_output.out
mpiexec_mpt -n ${BC_MPI_TASKS_ALLOC} python ./best_arch_platypus_parallel_77.py
    -c 2 -t 813 -a SPEA2 -p 100 -e 25000 > spea_813_77_output.out

## Cleanup ------------------------------------------------
cd ${JOBDIR}
rm *.py
rm 1.res 2.res 3.res 4.res 5.res 6.res 7.res 8.res 9.res 10.res 11.res
rm 12.res 13.res 14.res 15.res 16.res 17.res 18.res 19.res 20.res 21.res
rm 22.res 23.res 24.res 25.res 26.res 27.res 28.res 29.res 30.res 31.res
rm 32.res 33.res 34.res 35.res 36.res 37.res 38.res 39.res 40.res 41.res
rm 42.res 43.res 44.res 45.res 46.res 47.res 48.res 49.res 50.res 51.res
```

```
rm 52.res 53.res 54.res 55.res 56.res 57.res 58.res 59.res 60.res 61.res
rm 62.res 63.res 64.res 65.res 66.res 67.res 68.res 69.res 70.res 71.res
rm 72.res 73.res 74.res 75.res 76.res 77.res
rm moon_phase.res sim_dates.res
```

Notes on PBS scripts:

1 A PBS script is just a just a shell script (bash, in this case) with some PBS directives added to the beginning. Anything that can be done in a shell script can be done here, as well.

2 The first mandatory directive selects resources for the job. Most important, "select" determines how many physical nodes will be used, then "ncpus" is the number of cores to be used per node (this must be set to the total number of cores in a standard node) and "mpiprocs" determines the total number of MPI processes per node. Total CPUs and MPI processes are found by multiplying the latter two numbers by the number of nodes selected. To save users from multiplication errors, the `${BC_MPI_TASKS_ALLOC}` variable is the total number of MPI processes for the current job. This can be replaced with an integer if a task requires fewer than the total requested processes, for some reason.

3 Of the optional directives, the most useful may be the `PBS -M` directive. Setting this to a valid email address will tell the system to email updates on the events specified in the `#PBS -m` directive.

4 Each mpiexec_mpt command is a single line in the script, but is broken into two lines to fit on the page. Similarly, all of the rm commands could be done in a single line.

5 Carefully calculate time required. This should be done using the *debug* queue to perform small runs to determine how long a full-scale job will require. In the case of this research, that means running test jobs on each subset of data, since evaluation times varied depending on the number of RSOs in the simulation data.

Data Generation

Running simulations is a fairly simple process, but it does have some dependencies. First, a full educational STK license is required to run the engine with all of the necessary tools. Next, ActiveX needs to be enabled on the machine, and the *wx* Python module must be installed. Finally, there must be enough memory. For full 813-RSO simulations, at least 80 GB of memory should be available to complete the simulations.

Granular control of the simulation is possible through command line arguments. The -t argument is used to specify the number of RSOs to use. The -g, -s, -e, and -n are flags that specify that ground-based, sun-synchronous, equatorial LEO, and Near-GEO telescopes should be simulated, respectively. While it is completely possible to do all simulations at once, it is usually more practical to only do one set at a time for the larger simulations (407 and 813 RSOs). Start an 813-RSO simulation with GBTs and equatorial obsats by running the following command:

```
python thesis_gen_77.py -t 813 -g -e
```

For help, run `python thesis_gen_77.py -h`

Running Optimizations

Optimization runs should be run on an HPC following the examples in the PBS script shown earlier in this appendix. Similar to the simulation script, the optimiza-

tion scripts take command line arguments to customize what is to be done. Running `python best_arch_platypus_77.py -h` will display instructions on how to use the arguments properly.

Small scale testing on a workstation is also possible. On a Windows machine, install Microsoft MPI. At the time of writing, the latest version is 10.0, and is available for download at https://www.microsoft.com/en-us/download/details.aspx?id=57467. Once installed, MPI programs can be run locally. Running IBEA on 4 MPI processes with the 20-RSO data set would be accomplished with the following command:

```
mpiexec -n 4 python best_arch_platypus_77.py -a IBEA -t 20 -p 10 -e 100
```

Performing comparisons

Running the post-processing scripts is done in a similar fashion to the other scripts, but it makes some assumptions about where data is stored. The script assumes that the result files from optimization runs exist in a sub-directory called "Results" and will fail if they are stored anywhere else. It also assumes that files have been renamed to the naming scheme of ALGORITHM_TARGETS_RUN.res, where runs are lettered, starting with A. So for the first IBEA run with the 203-RSO data set, it would be named `IBEA_203_A.res`.

To perform binary comparisons, use the `merge_platypus_fronts.py` script. This script takes two arguments: -t specifies the number of targets, and -m is a flag specifying that fronts should be merged. Omitting the -m flag will result in comparisons between individual runs instead of aggregated fronts.

Bibliography

1. J. Chin, R. Coelho, J. Foley, A. Johnstone, R. Nugent, D. Pignatelli, S. Pignatelli, N. Powell, J. Puig-Suari, W. Atkinson, J. Dorsey, S. Higginbotham, M. Krienke, K. Nelson, B. Poffenberger, C. Raffington, G. Skrobot, J. Treptow, A. Sweet, J. Crusan, C. Galica, W. Horne, C. Norton, and A. Robinson, "CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers," no. October, p. 96, 2017. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf

2. T. Berg, "Irvine students launch second satellite in a month," 2018. [Online]. Available: https://www.irvinestandard.com/2018/irvine-students-launch-second-satellite-in-a-month/

3. NASIC, "Competing in space," Tech. Rep., 2018.

4. E. Howell, "What Is a Geosynchronous Orbit? — Space," 2015. [Online]. Available: https://www.space.com/29222-geosynchronous-orbit.html

5. L. Grush, "FCC approves SpaceX's plan to launch more than 7,000 internet-beaming satellites," 2018. [Online]. Available: https://www.theverge.com/2018/11/15/18096943/spacex-fcc-starlink-satellites-approval-constellation-internet-from-space

6. J. Porter, "Amazon will launch thousands of satellites to provide internet around the world," 2019. [Online]. Available: https://www.theverge.com/2019/4/4/18295310/amazon-project-kuiper-satellite-internet-low-earth-orbit-facebook-spacex-starlink

7. L. Matsakis, "Facebook Confirms It's Working on a New Internet Satellite," 2018. [Online]. Available: https://www.wired.com/story/facebook-confirms-its-working-on-new-internet-satellite/

8. P. R. Author Mark Ackermann, P. C. Zimmer, J. McGraw, l. T. John McGraw, and l. D. David Cox, "A Systematic Examination of Ground-Based and Space-Based Approaches to Optical Detecction and Tracking of Satellites," in *31st Space Symposium*, 2015.

9. J. Stern, S. Wachtel, J. Colombi, D. Meyer, and R. Cobb, "Multi-objective optimization of Geosynchronous Earth Orbit space situational awareness system architectures," in *15th Annual Conference on Systems Engineering (CSER)*, 2017. [Online]. Available: http://www.incose.org/ChaptersGroups/Chapters/ChapterSites/los-angeles/chapter-events/conferences/cser2017

10. TOP500.org, "Spirit." [Online]. Available: https://www.top500.org/system/177935

11. D. H. Wolpert and W. G. Macready, "No-Free-Lunch Theorem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

12. C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed., D. E. Goldberg and J. R. Koza, Eds. New York: Springer Science+Business Media, 2007.

13. D. J. Delgado, R. Torres-Sáez, and R. Llamosa-Villalba, "Develop an executable architecture for a System of Systems: A teaching management model," *Procedia Computer Science*, vol. 36, no. C, pp. 80–86, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2014.09.041

14. HPCMP, "HPC Centers - Unclassified Systems." [Online]. Available: https://centers.hpc.mil/systems/unclassified.html#Thunder

15. J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd ed. Cambridge: The MIT Press, 1992.

16. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading: Addison-Wesley, 1989. [Online]. Available: https://dl.acm.org/citation.cfm?id=534133

17. D. B. Fogel, *Evolutionary Computation, 2005*, 3rd ed. Hoboken: Wiley, 2005.

18. H.-g. Beyer and H.-p. Schwefel, "Evolution strategies – A Comprehensive Introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

19. A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44874-8

20. D. Moomey, "A call to action: Aid geostationary space situational awareness with commercial Telescopes," *Air and Space Power Journal*, 2015.

21. A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Essex: Pearson, 2003.

22. C. A. C. Coello, G. B. Lamont, D. A. V. Veldhuizen, D. E. Goldberg, and J. R. Koza, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*, 2006.

23. K. Y. Szeto and J. Zhang, "Adaptive genetic algorithm and quasi-parallel genetic algorithm: Application to knapsack problem," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3743 LNCS, pp. 189–196, 2006.

24. E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert, "Performance Assessment of Multiobjective Optimizers : An Analysis and Review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

25. W. B. T. Mock, "Pareto Optimality," in *Encyclopedia of Global Justice*, D. K. Chatterjee, Ed. Dordrecht: Springer, 2011, pp. 808–809.

26. S. K. Smit and A. E. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pp. 399–406, 2009.

27. M. Sipper, W. Fu, K. Ahuja, and J. H. Moore, "Investigating the parameter space of evolutionary algorithms," *BioData Mining*, vol. 11, no. 1, pp. 1–15, 2018.

28. M. Mobin, S. M. Mousavi, M. Komaki, and M. Tavana, "A hybrid desirability function approach for tuning parameters in evolutionary optimization algorithms," *Measurement: Journal of the International Measurement Confederation*, vol. 114, no. June 2017, pp. 417–427, 2018. [Online]. Available: https://doi.org/10.1016/j.measurement.2017.10.009

29. O. Kramer, "Evolutionary self-adaptation : a survey of operators and strategy parameters," 2010.

30. V. Beiranvand, W. Hare, and Y. Lucet, "Best practices for comparing optimization algorithms," *Optimization and Engineering*, vol. 18, no. 4, pp. 815–848, 2017.

31. A. Garrett, "inspyred (Version 1.0.1)," 2012. [Online]. Available: https://github.com/aarongarrett/inspyred

32. ESA ACT, "What is the Advanced Concepts Team? / ACT / ESA." [Online]. Available: http://www.esa.int/gsp/ACT/about/theteam.html

33. ——, "Pagmo / Pygmo / ACT / ESA." [Online]. Available: http://www.esa.int/gsp/ACT/open_source/pagmo.html

34. D. Hadka, "Playtypus (Version 1.0.3)," 2015. [Online]. Available: https://github.com/Project-Platypus/Platypus

35. E. Zitzler and S. Künzli, "Indicator-Based Selection in Multiobjective Search," pp. 832–842, 2010.

36. Z. Qingfu and L. Hui, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4358754

37. K. Deb, S. Pratab, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NGSA-II," *IEEE Transactions on Evolutionary Computing*, vol. 6, no. 2, pp. 182–197, 2002.

38. J. B. Kollat and P. M. Reed, "The Value of Online Adaptive Search: A Performance Comparison of NSGAII, ϵ-NSGAII and ϵMOEA," in *Evolutionary Multi-Criterion Optimization*, C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, Eds. Heidelberg: Springer Berlin Heidelberg, 2005, pp. 386–398.

39. M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining Convergence and Diversity in Evolutionary Multi-Objective Optimization," *Evolutionary Computation*, vol. 10, no. 3, pp. 263–282, 2002.

40. H. Jain and K. Deb, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 602–622, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6600851%5Cnhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6600851

41. J. Knowles and D. Corne, "The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation," *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, vol. 1, pp. 98–105, 1999.

42. E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

43. E. Zitzler, "Evolutionary algorithms for multiobjective optimization: Methods and applications (PhD dissertation)," Ph.D. dissertation, Institut für Technische Informatik und Kommunikationsnetze, 1999.

44. E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

45. U.S Joint Cheifs of Staff, "Joint Publication 3-14 Space Operations," *Joint Publication 3-14*, no. April 2018, 2013.

46. United States Air Force Curtis E. Lemay Center, "Annex 3-14 - Counterspace Operations," pp. 1–35, 2018.

47. E. Fahnestock and R. S. Erwin, "Optimization of Hybrid Satellite and Constellation Design for GEO-Belt Space Situational Awareness Using Genetic Algorithms," in *American Control Conference*, 2005, pp. 2110–2115.

48. J. M. Yates, B. W. Spanbauer, and J. T. Black, "Geostationary orbit development and evaluation for space situational awareness," *Acta Astronautica*, vol. 81, no. 1, pp. 256–272, 2012.

49. M. R. Ackermann, P. C. Zimmer, and W. T. Vestrand, "Alternatives for Ground-Based, Large-Aperture Optical Space Surveillance Systems," *Advanced Maui Optical and Space Surveillance Technologies Conference*, pp. 1–23, 2013.

50. M. S. Felten, "Optimization of Geosynchronous Space Situational Awareness Architectures Using Parallel Computation," Ph.D. dissertation, Air Force Institute of Technology, 2018. [Online]. Available: https://www.dtic.mil/DTICOnline/downloadPdf.search?collectionId=tr&docId=AD1056485

51. AGI, "Systems Toolkit," 2019. [Online]. Available: https://www.agi.com/products/engineering-tools

52. ——, "STK Programming Interface," 2018. [Online]. Available: http://help.agi.com/stkdevkit/index.htm

53. C. Scott, "Latency Trends," 2012. [Online]. Available: https://colin-scott.github.io/blog/2012/12/24/latency-trends/

54. AFRL, "Mustang PBS Guide," 2018. [Online]. Available: https://www.afrl.hpc.mil/docs/mustangPbsGuide.html

55. E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

56. R. S. Barr, B. L. Golden, J. Kelly, W. R. Stewart, and M. G. C. Resende, "Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods," *SciencesNew York*, vol. 1, no. 1, pp. 1–15, 2001. [Online]. Available: http://www.springerlink.com/index/10.1007/BF02430363

57. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, 2002.

58. J. Brownlee, *Clever Algorithms*, 2011. [Online]. Available: http://www.cleveralgorithms.com

59. M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge: The MIT Press, 2004.

60. K. A. Dowsland and J. M. Thompson, "Simulated Annealing," in *Handbook of Natural Computing*. Heidelberg: Springer-Verlag, 2012, vol. 4, pp. 1623–1655.

61. Python Software Foundation, "PEP 373 – Python 2.7 Release Schedule — Python.org," 2008. [Online]. Available: https://www.python.org/dev/peps/pep-0373/

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
13–06–2019	Master's Thesis	Sept 2017 — Jun 2019

4. TITLE AND SUBTITLE

Comparing Multi-objective Search Algorithms When Applied to Real World Problems as Demonstrated with the Space Surveillance Network (SSN) Architecture Optimization Problem

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Troy B. Dontigney, MSgt

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 an Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-19-J-003

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RV
LtCol Christopher Allen
AFRL/RVEP
Kirtland AFB, NM 87117
DSN 246-1246, COMM 505-846-1246
Email: christopher.allen.3@us.af.mil

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

14. ABSTRACT

Space Situational Awareness (SSA) is an activity vital to protecting national and commercial satellites from damage or destruction due to collisions. Recent research has demonstrated a methodology using evolutionary algorithms (EAs) which is intended to develop near-optimal Space Surveillance Network (SSN) architectures in the sense of low cost, low latency, and high resolution. That research is extended here by (1) developing and applying a methodology to compare the performance of two ore more algorithms against this problem, and (2) analyzing the effects of using reduced data sets in those searches. Computational experiments are presented in which the performance of five multi-objective search algorithms are compared to one another using four binary comparison methods, each quantifying the relationship between two solution sets in different ways. Relative rankings reveal strengths and weaknesses of evaluated algorithms empowering researchers to select the best algorithm for their specific needs. The use of reduced data sets is shown to be useful for producing relative rankings of algorithms that are representative of rankings produced using the full set.

15. SUBJECT TERMS

Space Situational Awareness, Space Surveillance Network Architecture Design, Multi-objective Optimization, Multifidelity Analysis

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. L. D. Merkle, AFIT/ENG
U	U	U	UU	115	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4526; laurence.merkle@afit.edu