3-2020

Heuristic Approaches for Near-Optimal Placement of GPS-Based Multi-Static Radar Receivers in American Coastal Waters

Brandon J. Hufstetler

HEURISTIC APPROACHES FOR NEAR-OPTIMAL PLACEMENT OF GPS-BASED MULTI-STATIC RADAR RECEIVERS IN AMERICAN COASTAL WATERS

THESIS

Brandon J. Hufstetler, Capt, USAF

AFIT-ENS-MS-20-M-154

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-20-M-154

HEURISTIC APPROACHES FOR NEAR-OPTIMAL PLACEMENT OF GPS-BASED MULTI-STATIC RADAR RECEIVERS IN AMERICAN COASTAL WATERS

THESIS

Presented to the Faculty

Department of Operational Sciences

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 Masters of Science in Operations Research

Brandon J. Hufstetler, B.S.M.E.

Capt, USAF

26 March 2020

AFIT-ENS-MS-20-M-154

HEURISTIC APPROACHES FOR NEAR-OPTIMAL PLACEMENT OF GPS-BASED MULTI-STATIC RADAR RECEIVERS IN AMERICAN COASTAL WATERS

THESIS

Brandon J. Hufstetler, B.S.M.E.
Capt, USAF

Committee Membership:

Lt. Col. Bruce A. Cox, Ph.D.
Chairman

Dr. Brian J. Lunday, Ph.D.
Reader

Dr. Julie A. Jackson, Ph.D.
Reader

Abstract

Narcotics smuggling across the Caribbean ocean is a growing concern for the United States Coast Guard. One vector for this illicit trafficking is via small aircraft. This thesis proposes a multi-static radar architecture using the Global Positioning System (GPS) constellation as a transmission source to detect these aircraft as they transit a detection fence. The multi-static radar system developed in this thesis relies on the forward-scatter phenomenon in which a radar shadow is cast by a target as it crosses in front of a transmitter, creating a measurable difference in the signal amplitude at the receiver. The forward-scatter radar shadow is detectable only within a narrow region outside of syzygy or the line-of-sight vector extending from the transmitter through the target to the receiver. Thus, a receiver must be placed in a strategic location to maximize the probability of conjunction with the target and transmitter.

This thesis first develops a mathematical model parametrizing such a multi-static radar system. This model is then used to build a novel simulation, and output from the simulation is used as input in a vast set covering problem whose goal is both to determine the smallest number of sensors along with their locations in order to detect 100% of transiting aircraft, and to determine the near optimal location of a fixed number of sensors. Towards this end two binary linear programs are created and solved to optimality for small instances of the problem, and to near optimality for larger instances including the full problem of interest. These solutions are compared against a genetic algorithm and a geometric heuristic.

The research proves the problem can be modeled, albeit at great computational expense. It further demonstrates that near optimal solutions can be generated with almost no computational expense using the geometric heuristic.

Acknowledgements

To my wife and children, thank you for your unwavering love and support. And to my advisor, your patience and relentless pursuit of knowledge has motivated me to explore more facets of this problem than I ever imagined possible.

Brandon J. Hufstetler

Table of Contents

Page

List of Figures

List of Tables

HEURISTIC APPROACHES FOR NEAR-OPTIMAL PLACEMENT OF GPS-BASED MULTI-STATIC RADAR RECEIVERS IN AMERICAN COASTAL WATERS

I. Introduction

1.1 Background

Narcotics such as cocaine and marijuana are known to be trafficked from South America to North America by air over the Caribbean Sea [9]. The US Coast Guard is tasked to interdict these illicit goods [39]. To increase the chance of a successful interception, current interdiction methods require near real time information about shipments underway [26]. This information can come in the form of human intelligence, visual observations, radio transmissions, or radar observations. This thesis examines an alternative information avenue: GPS-based multistatic radar.

The scale of the illegal shipments problem in the Atlantic prompted the US Coast Guard in 2012 to increase its deployment of vessels and aircraft for conducting drug interdiction patrols in the vicinity of Puerto Rico and the US Virgin Islands. Customs and Border Protection initiated operations to intercept illegal weapons, drugs, and money along the southern coast of Puerto Rico and the Dominican Republic. That same year, the Immigration and Customs Enforcement in San Juan, Puerto Rico, brought together over 120 agencies to conduct enforcement operations and seized 450 pounds of narcotics, $543,000 in US currency, 650 illegal weapons and over 40,000 rounds of ammunition. Finally, the Transportation Security Administration intensified searches of luggage, parcels, and freight transported on flights from Puerto Rico

destined for the continental United States [3].

Despite these agencies increasing operations in the area, there are simply not enough resources available to identify and track all nefarious aircraft smuggling goods across the Caribbean Sea. The goal of this paper is to determine a theoretical deployment of GPS receivers to act as the receive antennas for a multistatic radar system that uses the forward-scatter phenomenon to locate target aircraft with a reasonable probability of detection and for an acceptable cost.

1.2 Problem Statement

This thesis concerns the method of observing and tracking targets known as passive coherent location (PCL). PCL is a radar method consisting of a non-cooperative transmission source, such as radio or cell phone towers, and a geographically separated receiver. This architecture with separate transmitters and receivers is known as bistatic or multistatic radar. Existing reliable transmission sources of opportunity provide an environment where bistatic radar can be employed cheaply and covertly. Benefits of employing PCL include the ability to identify targets of interest without compromising the location of the detector and the low cost associated with fielding such a system.

With an omnipresent transmission source, such as the Global Positioning System (GPS) satellite constellation, it is currently possible to execute PCL globally. It is theoretically possible to use PCL in a multi-static radar system that can provide 24-hour airspace surveillance for any given region. One specific area where GPS-based PCL may be of interest is the detection of smuggling aircraft traveling between South and North America. Early identification of uncooperative aircraft could provide authorities with a longer opportunity to interdict these aircraft. Receivers in this system could be attached to floating platforms in the Caribbean Sea.

This paper examines the feasibility of such an architecture specifically using a mixture of simulation, optimization, and heuristic search techniques to determine an optimal, or near optimal, sensor geometry to establish a detection fence one degree in longitude by two degrees in latitude in the Caribbean Sea. The radar geometry developed in this thesis uses a forward-scatter effect. The forward-scatter bistatic radar effect occurs when a target approaches the line-of-sight vector, known as the baseline, between a transmission source and receiver. A receiver in this configuration can measure the decrease in energy received, caused by the target interfering with the signal.

The system under consideration can be modeled as a set covering problem. Set covering problems require elements to be covered and sets which cover them. Each receiver location is at a static latitude and longitude and consists of a GPS receiver capable of receiving signals from any overhead GPS satellite. The detection of an aircraft by a receiver is determined if the aircraft passes through a detection region, defined as the line-of-sight vector between a GPS satellite and receiver plus a small area around that vector as defined by radar physics. The aircraft are the elements to be covered by the set covering problem. Each potential sensor location in this thesis defines a set which covers every aircraft detected by a receiver placed at that location.

1.3 Research Questions

- Can one model the forward-scatter GPS receiver/target problem over a geographic area of interest as a set covering problem?
- Can a meta-heuristic search algorithm solve for the pseudo-Pareto frontier to maximize the probability of detection and minimize cost to setup such a GPS

based multistatic network over that defined area?

1.4 Approach

This research develops a physics based mathematical model and implements it in a simulation to track the detectable radar shadows created by an aircraft as it flies under GPS satellites. By implementing various heuristic search methods, the research solves a set covering problem identifying the optimal location of receive antennas to maximize the probability of detection of an aircraft while minimizing the quantity of receivers.

Two variations of the set covering problem are solved in this thesis. The first solves for the minimal number of receivers required to detect all of the targets. The second solves for the maximum number of targets that can be detected given a fixed number of sensors.

1.5 Summary

In this thesis I make the following contributions: I develops a reproducible physics based model of a GPS-based forward-scatter radar architecture. I create a GPS-based forward-scatter radar detection matrix for aircraft flying at 10,000 feet over a 1 degree by 2 degree area of the Caribbean Sea. I develop genetic algorithms tuned to the data to solve a minimal and maximal set covering problem. I validate the results of the genetic algorithm against known optimal solutions and time-limited deterministic methods.

The remainder of the paper proceeds as follows: Chapter II details current known trafficking routes and interdiction techniques as well as the physics of the bistatic radar system proposed to augment current capabilities. It also gives background on the meta-heuristic search methods used to optimize the stated objectives and

describes the modeling techniques employed to evaluate the system.

Chapter III discusses radar modeling, specifically the development of the model for detectable GPS radar shadows. Chapter III further outlines the dimensionality reduction necessary to ensure the problem is tractable. Finally, it shows the heuristic approaches taken and walks the reader through the iterative steps of the algorithms.

Chapter IV examines the results of the model and meta-heuristics and provides an analysis of the proposed system under different constraints.

Finally, Chapter V summarizes results and conclusions to the thesis, and establishes possible future research avenues.

II. Literature Review

2.1 Overview

The literature reviewed for this thesis includes relevant papers and books on the topics of narcotics smuggling, interdiction methods, radar phenomenology, optimization techniques, and heuristic search methods.

2.2 Trafficking Routes

I examined the detection and observation of air and maritime smuggling vessels in support of US Coast Guard trafficking interdiction operations [34] using back-scatter bistatic radar, but after researching the effectiveness of back-scatter radar for detecting small smuggling vessels [8, 41], decided to omit maritime vessels due to the difficulty of detecting them at long ranges using this method. Thus, this thesis focuses solely on the detection of aircraft suspected of transporting illegal goods over the Caribbean Sea. The detection method evaluated is a multistatic forward-scatter radar system using GPS satellites as the transmission sources and ocean buoy based GPS antennas as the receivers.

To paraphrase Bethel [9], after the Cold War, the rise of globalization provided an opportunity for domestic crime organizations to easily expand into transnational operations. These crimes include illegal migration, money laundering, as well as narcotics and arms smuggling. These extensive criminal networks undermine democratic stability, economic development, and social well being while also creating an international threat spawning security issues, territorial disputes, and economic declines that strain available resources. The narcotics trade in particular is often the center of gravity for many problems associated with transnational crime.

Atkinson [3] states that in the western hemisphere, Central America and the

Caribbean are the passageway for illicit goods manufactured in the world's leading producer of cocaine and marijuana, South America, to the world's largest consumer of these goods, the United States of America. The Caribbean Sea offers smugglers a largely unguarded passageway between South and North America. The geography of the tiny islands of the Caribbean, known as the third border of the United States [3], increases the complexity of counter-drug measures. The islands cover a large area from Trinidad, near the northern tip of Venezuela, to the Bahamas, just fifty miles off the United States east coast. This distribution of ample coastlines combined with their proximity to major sea trade routes like the Panama Canal make this region a natural transit route for narcotics trafficking [9].

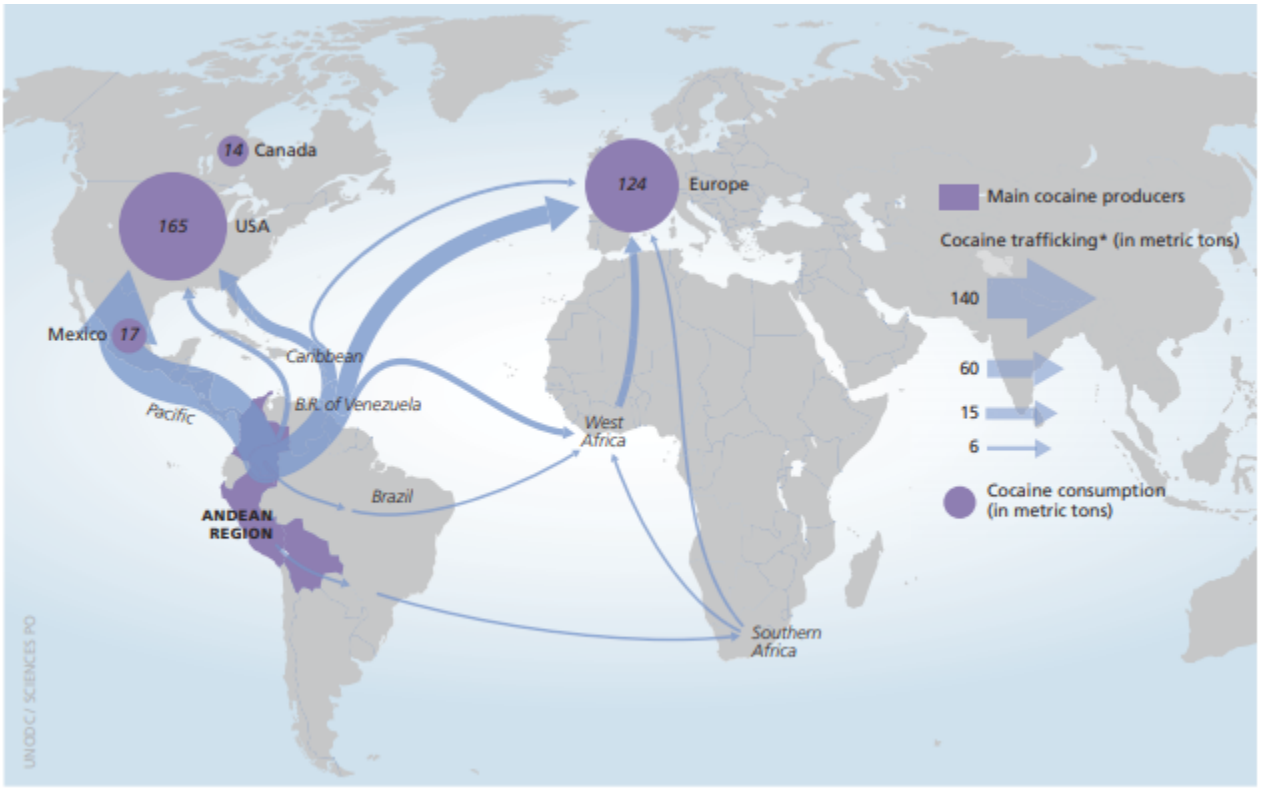


Figure 1. Main global cocaine sources, transportation vectors, and consumer markets in 2008

The United Nations Office on Drugs and Crime (UNODC) World Drug Report [40] describes cocaine as, second to opiates, the most problematic drug globally, with

almost one million US citizens dependent on the substance. Figure 1, taken from the World Drug Report, illustrates the quantity of cocaine smuggled globally. While the majority of US cocaine is smuggled through the Pacific Ocean and Mexico, some US and almost all other global cocaine distribution goes through the Caribbean.

Figure 2, from the UNODC [39], shows the vectors by which cocaine is smuggled into the US. The bulk of the drugs smuggled by air is seen to transit the Caribbean.

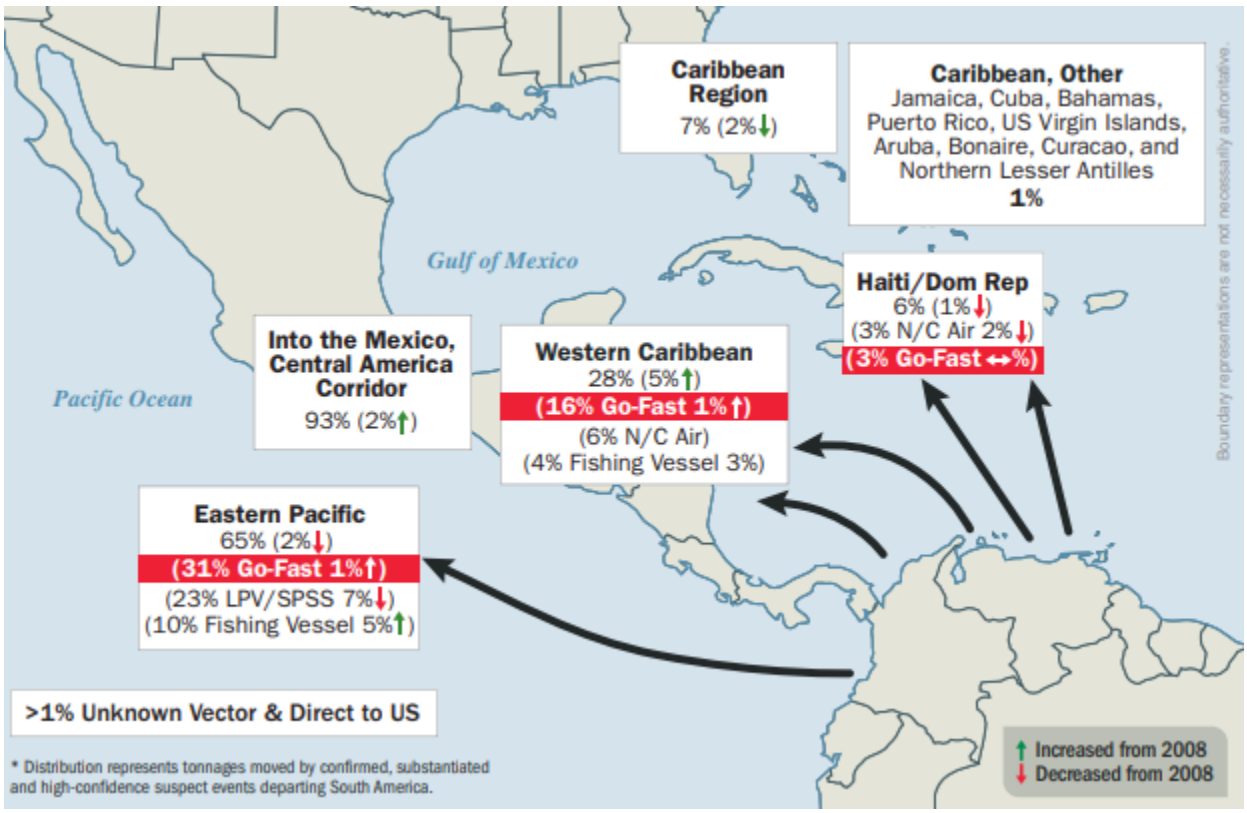


Figure 2. Distribution of cocaine trafficking vectors toward the united states in 2009 showing nine percent of cocaine is transported by non-commercial aircraft (N/C Air) over the Caribbean Sea and the remaining ocean borne traffic by go-fast boats, fishing vessels, or self-propelled semi submersibles (SPSS) [39].

According to Office of National Drug Control Policy (UNODC) [39], "one of the primary missions of the US Southern Command (USSOUTHCOM) is to disrupt the flow of drugs from Central and South America to the United States via the southern approaches." and Section 888 of the Homeland Security Act of 2002 categorizes "drug

interdiction" as one of the United States Coast Guard's (USCG) Homeland Security Missions [44]. The methods by which the USCG carries out this mission are outlined in Coast Guard Joint Publication 1-0 [26]. Resources for drug interdiction are limited and unable to fully combat the problem of detecting and intercepting nefarious aircraft en route [12]. This thesis seeks to outline a fiscally responsible method for detecting aircraft as they cross the Caribbean in support of the USCG's drug interdiction mission.

2.3 Bistatic Radar

RAdio Detection And Ranging, RADAR, is an electronic system used for the detection and location of objects [42]. It measures electromagnetic energy between 3 MHz and 300 GHz. By listening for radio waves with a known source it is able to infer information about the environment in which it is employed and the objects it encounters [38]. Since its inception, radar has be developed for a variety of environments for myriad remote sensing applications [37]. Three broad categories of radar are mono-static, where the transmission source and receiver are co-located, bistatic, where they are geographically separated, and multistatic, where multiple transmitters and receivers work in conjunction with each other [60]. This thesis is concerned with bistatic applications.

Bistatic radar may operate with a cooperative transmission source that is actively transmitting as a part of a bistatic radar architecture, or with an uncooperative transmission source of opportunity where the receiver is the only controlled component in the bistatic radar architecture. Such an uncooperative system is referred to as passive bistatic radar [60]. Advances in bistatic radar and the proliferation of electromagnetic transmission sources has created a global environment ripe for passive radar observation [25]. Bistatic radar is an attractive option because it does not compromise

the operator's location and can detect targets with geometries designed specifically to hide from mono-static radar [63].

GPS has a demonstrated capability as a good transmission source of opportunity for non-cooperative bistatic radar: Its ability to penetrate the atmosphere and weather with multiple concurrent transmission sources overhead at any given moment, the US Air Force's declaration to maintain 95% operational reliability, and the carrier wave frequencies it broadcasts on has led to GPS being used in many bistatic radar missions [1, 18, 22, 23]. Because GPS transmits in the L-band frequencies at large distances, most applications for targeting or tracking utilize far-field radar physics, which are simpler to theorize and simulate than their near-field counterparts [28, 56]. Bistatic radar, specifically with GPS as the transmission source has already demonstrated the ability to function in many sensing applications.

Powell [43] uses real world data to prove GPS can be used to provide altitude measurements for a National Oceanic and Atmospheric Administration (NOAA) hurricane hunter. A GPS receiver was installed in the aircraft and compared the direct signal received from a satellite to a secondary signal reflected off the surface of the ocean. Masters [36], mounted a different type of GPS receiver to an NCAR C-130 aircraft to produce altitude and soil moisture measurements using the Doppler shift of a GPS signal reflected off of the ground. Azemati [5] extended the soil moisture measurements into a vegetation analysis by mounting a receiver to a Twin-Otter DHC-6.

Daout [17] developed equations for GPS based multistatic image resolution analysis. Luengo [14] discusses work with the Cyclone Global Navigation Satellite System (CyGNSS) data which compared using a space based GPS reciever to perform soil moisture measurements versus an active microwave radiometry baseline. Luengo shows that the GPS bistatic radar architecture is capable of performing as well as

other more expensive methods, even when the reciever is space based. Saïd [50] discusses calibrations of the CyGNSS measurements and operational successes of the system. Gleason [21] summarizes work on the UK-DMC satellite which used a GPS receiver to perform a bistatic radar capable of measuring the wind speed over the open ocean by relating the Doppler shifted signal to the surface roughness of the water. Zavorotny and Voronovich also demonstrate work done to calculate wind speed by measuring ocean surface roughness via the diffusion of GPS based bistatic radar scattering [58, 62].

In a bistatic radar system, there are two distinct geometries which allow a target to be observed. Willis [60] describes back-scatter as occurring when the bistatic angle β, between the transmitter, target, and receiver is sufficiently small (less than 140 degrees) such that the receiver is able to detect the signal reflected off of the target. When the bistatic angle is greater than 140 degrees, the receiver is instead able to detect a reduction in signal, because the original signal is scattered by the target. Willis refers to this geometry as forward-scatter bistatic radar.

Transitioning away from air and space based receivers using bistatic radar to measure features of the land, Kaiser [31] demonstrated the ability to detect aircraft in flight using back-scattered GPS signals. Li [35] investigates the problem of target position estimation with a single-observer back-scatter radar system. Sakhawat [51] quantified the power budget of GPS signals reflected off of an aircraft. Other transmitters of opportunity in this geometry have been explored in detail by Inggs [30] in a cognitive radar model that uses multiple transmission sources. Inggs outlines benefits of passive coherent location stating,

> *It is clear that such a system, consisting of multiple, cooperating receivers, can achieve excellent performance in the presence of deliberate jamming, difficult terrain, and attempts at target stealth. In the civilian radar domain, the technology offers opportunities for bandwidth conservation.*

While it is physically possible for GPS to be used in a back-scatter bistatic configuration [60], the weak signal power at the receiver makes target identification and tracking more difficult than traditional mono-static radar [18]. Pui [45] demonstrates the use of phased array receivers to improve the GPS signal quality when detecting aircraft in a bistatic radar system. Pastina [41] demonstrates that long integration times of the radar signal can also be used to improve the probability of detection but the target would need to be either stationary or slow moving.

Gashinova [20] suggests it is far easier to detect a target using a forward-scatter configuration given the proper geometry can be achieved. Barton [6] concludes that the forward-scatter bistatic effect is the most significant and compelling use for bistatic radar applications because of the large RCS observed in forward-scatter geometries. Willis [60] recounts some of the earliest uses of radar in the US and the UK, which were in a forward-scatter configuration. Forward-scatter radar was used to create a fence and low flying aircraft crossing that fence were detected because they effectively blocked a part of the signal that would have been observed at the receiver. Willis states that because the RCS of any type of target is significantly increased as that target approaches the baseline connecting the transmitter to the receiver, forward-scatter radar can have useful counter-stealth applications.

Behar [7] analyzed a GPS based forward-scatter bistatic radar system for the detection of multiple aircraft types, including the aircraft under consideration in this thesis. Behar calculated the maximum distance that each aircraft type could be from a receiver and maintain a significant probability of detection in back-scatter and forward-scatter geometries. Behar shows analytically that the forward-scatter geometry offers a dramatically larger detection range as compared to the range in a back-scatter geometry.

Forward-scatter bistatic radar is not without its disadvantages, though. Accord-

ing to Griffiths [25] the most significant limitation to ground based forward-scatter bistatic radar is the precise geometry required for it to work. In terrestrial systems, aircraft would have to be flying at very low altitudes in order for their forward-scatter shadow to intersect the receiver. Griffiths alludes that an elevated transmitter or receiver would greatly expand the capabilities of a forward-scatter radar system.

This thesis develops a multistatic system with sufficient space borne transmitters and terrestrial receivers, strategically placed to maximize the opportunity for forward-scatter bistatic geometries, for the detection of aircraft.

2.4 Heuristic Approaches

The physics based simulation model created in this thesis describes the sensor-aircraft relationship as a matrix with each sensor location along the horizontal axis and each aircraft along the vertical axis. The cells represent a single sensor and aircraft instantiation interaction. Each cell is assigned a binary value, 0 if a sensor placed at the associated location would not detect a particular aircraft instantiation and 1 if it would. This research leverages this matrix to determine which subset of possible sensor locations should be occupied in order to minimize the number of aircraft that are able to move through the model undetected. This means there are potentially 2^n possible solutions to analyze, where n is the number of potential sensor locations being considered and each solution is a subset of sensor locations.

The number of feasible solutions can be reduced by setting a maximum or fixed number of nodes that can be occupied. Radmard [46] analyses this type of set covering geometry optimization problem in a multistatic radar system configuration, but with a tractable number of potential transmitters and receivers. Regardless, the number of possible solutions grows exponentially with the number of nodes under consideration. Gonzalez [24] states that "It was clear that even for small values of n, exponential time

complexity equates to computational intractability if the algorithm actually performs an exponential number of operations for some inputs." The complexity of the system developed in this thesis fits into the latter category and quickly becomes intractable.

Karp [33] classifies the problem being solved in this thesis as a Set Covering Problem (SCP) and lists it as one of Karp's 21 NP-complete problems. Karp defines SCPs as a finite family of finite sets $\{S_j\}$, and a positive integer k with the property that there is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$. Caprara [13] evaluates several exact approaches to solving the SCP and redefines it in a manner that is better suited to illustrate the problem posed in this thesis. Caprara defines SCP in the following way:

> Let $A = (a_{ij})$ be a 0-1 $m \times n$ matrix, and $c = (c_j)$ be an n-dimensional integer vector. In the following I refer to the rows and columns of A simply as rows and columns. Let $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$. The value c_j $(j \in N)$ represents the colst of column j, and I assume without loss of generality $c_j > 0$ for $j \in N$. I say that a column $j \in N$ *covers* a row $i \in M$ if $a_{ij} = 1$. SCP calls for a minimum-cost subset $S \subseteq N$ of columns, such that each row $i \in M$ is covered by at least one column $j \in S$. A natural mathematical model for SCP is
>
> $$v(SCP) = \min \sum_{j \in N} c_j x_j$$
>
> subject to
>
> $$\sum_{j \in N} a_{ij} x_j \geq 1, \qquad i \in M,$$
>
> $$x_j \in \{0, 1\}, \qquad j \in N,$$
>
> where $x_j = 1$ if $j \in S$, $x_j = 0$ otherwise. For notational convenience, for each row $i \in M$ let
>
> $$J_i = \{j \in N : \ a_{ij} = 1\}$$
>
> be the set of columns covering row i. Analogously, for each column $j \in N$ let
>
> $$I_j = \{i \in M : \ a_{ij} = 1\}$$
>
> be the row subset covered by column j. Moreover, let $q = \sum_{i \in M} \sum_{j \in N} a_{ij}$ denote the number of nonzero entries of A, and note that generally q is

much smaller than mn.

Caprara's work shows that the most effective exact approach to solve an SCP is to use a branch-and-bound algorithm with a primal-dual linear program. Caprara also demonstrates in the analysis that the solution time grows exponentially with instance size. Where an instance size of 300×3000 took about 5 seconds to solve, an instance size of 1000×10000 took 1885 seconds. The problem in this thesis is significantly larger than the largest problem analysed by Caprara [13]. Therefore, the author seeks to use an approximation technique to find a solution of acceptable quality without necessitating it be the exact optimal solution. As described by Caprara, a strict mathematical approximation method does not exist for this class of problem. Due to the nature of the SCP as a known NP-complete problem, it is reasonable to utilize heuristic search methods to identify an acceptable solution in a reasonable amount of time [48].

According to Bolc [10], the term heuristic comes from the Greek word *heuriskein*, meaning the art of problem solving or discovering new methods of solving problems. Bolc describes a heuristic as a 'rule-of-thumb' method that should provide a better solution than random selection but doesn't guarantee the optimal solution. Bolc describes exhaustive heuristic search strategies utilizing information about the problem to minimize an objective function and reduce the computational effort required to solve a problem. Examples of these methods are the best-first strategy, comparable to the well known nearest-neighbor approach used in the Travelling Salesman Problem [19], which constructs a solution by selecting the best node at any given step.

Talbi [55] classifies heuristic search algorithms into two classes: specific heuristics and meta-heuristics. The class of interest here is meta-heuristics, which "represent more general approximate algorithms applicable to a large variety of optimization problems" [55]. Talbi further separates meta-heuristics into single-agent based meta-

heuristics and population based meta-heuristics. Evans [19] and Williamson [59] outline several specific heuristics for combinatorial problems but their solution quality is, on average, inferior to many of the meta-heuristic approaches.

For the type of binary combinatorial node selection problem under consideration in this thesis, Gonzalez [24] recommends using a meta-heuristic search algorithm such as simulated annealing, ant colony optimization, or an evolutionary computation.

Xiao [61] describes animal models in computing. These meta-heuristic approaches rely on the social communication systems of the specific animal species being modeled as opposed to the mate selection and reproductive nature used in evolutionary algorithms. Xiao states that one of the major challenges these models are able to overcome is that of decentralized coordination of many separate agents. Many animals "travel and forage out of sight of most of the rest of their social group" and are able to pseudo-optimize the survival of their community. One specific instance of an animal based optimization model is ant colony optimization.

Bonabeau [11] writes in great detail about ant colony optimization. The premise of an insect swarm model such as ant colony optimization is that many simple agents can be allowed to make pseudo random choices in the solution space and their interactions should lead to a consensus around the optimal solution. Bonabeau cites the benefits of modeling a meta-heuristic search algorithm using a social insect metaphor to include distributedness, direct and indirect interactions among relatively simple agents, flexibility, and robustness. Bonabeau claims that the number of successful applications of swarm intelligence meta-heuristics is growing exponentially in combinatorial optimization, communications networks, and robotics.

Evans [19] specifically recommends using an evolutionary algorithm for the type of problem under examination in this thesis. Evans states that "when we compare evolutionary algorithm solutions with other types of solutions, we see that evolutionary

algorithms work pretty well."

Chiong [15] divides evolutionary algorithms into two subclasses: "Those that describe something that changes incrementally over time, such as the software requirements for a payroll accounting system" and "those that describe an evolutionary system that changes from generation to generation via reproductive variation and selection". The second class, which represents reproduction, is described as being: an evolution strategy, which focuses heavily on selection and mutation; an evolutionary program, which considers individual solutions as independent species; a memetic computation, which seeds a population based on a priori knowledge of the system; or a genetic algorithm, which relies on reproduction and mutation between generations to produce better solutions over time.

Chiong [15] outlines the basic cycle of an evolutionary algorithm as starting with an initial population of random individuals. Next, each individual is mapped into the scenario being modeled. An evaluation of each individual determines its fitness, or objective value. A selection of individual members is made based on the desirability of their fitness. Reproduction occurs in some manner of combining the selected individuals. Finally, the next generation is mapped into the scenario and the process continues until the population converges on a single solution.

Iba [29] differentiates genetic algorithms into real-valued genetic algorithms, particle swarm optimizers, bug-based search strategies, and differential evolution. Iba's study of the optimum door placement for evacuation planning is highly correlated with the problem of interest in this thesis. Iba says, "In contrast to refining the escape behavior model for evacuees, finding an optimal placement of exits can be another research direction with a much broader scope." Similarly, rather than changing the behavior of traffickers I can exert greater control over detecting them where they are.

The problem tackled in this thesis is also a placement problem with many competing factors such as the path an aircraft may follow, the movement of satellites, and the time when the aircraft is flying. It must balance all of these factors to find a best solution for any case it is presented.

Coley [16] explores improvements that can be made to genetic algorithms in their robustness. Coley elaborates on parameter selections that affect mutation, selection, elitism, crossover, and initialization. Coley further explores foundations of schema processing, which is the art of finding the useful qualities of an individual and promoting them as opposed to promoting the entire individual. Coley [16] then provides advanced replication apparatus specific to combinatorial optimization.

Kargupta [32] describes gene expression, or schema processing, to determine the best features of any individual in a genetic algorithm. Using Kargupta's method of gene ranking, the good schemata are identified and a selection operator combines subsets of schemata to create a new population. Haupt [27] also offers alternative crossover methods for binary genetic algorithms.

As observed by Whitley [4], "It can be argued that there are only two primary factors (and perhaps only two factors) in genetic search: population diversity and selective pressure [...] In some sense this is just another variation on the idea of exploration versus exploitation that has been discussed by Holland and others." Thus, an intelligent approach to controlling intensification and diversification is used in this thesis.

Based on the available research regarding the optimization of binary combinatorial problems, this thesis uses a genetic algorithm with a specific focus on schema description, mutation, and memetic initialization. This provides a robust model with outputs that highlight areas of optimal sensor placement.

2.5 Research Summary

To determine an acceptable system configuration to augment the USCG drug interdiction mission, capable of providing a reliable probability of detection at a reasonable cost, I employ a variety of methods. A physics-based simulation is used to used to model the behavior of drug smuggling aircraft. A mathematical model is developed to model the satellite/aircraft/receiver geometries over the time frame of the simulation. Finally, a meta-heuristic is tailored to solve the set covering problem of where to employ GPS receivers for the forward-scatter bistatic radar network with the goals of maximizing probability of detection and minimizing cost.

III. Methodology

3.1 Overview

This thesis derives physics based mathematical models to describe the properties of a GPS based forward-scatter bistatic radar configuration. These models are then coded via a novel implementation of the Software Toolkit (STK) program to simulate the terrestrial, airborne, and space based aspects of a multistatic radar architecture. The resultant matrix output from this model is the input to the actual set covering problems of interest. Finally, three heuristic search approaches are utilized to calculate optimal or near optimal architectural geometries.

The remainder of this chapter proceeds as follows: In Section 3.2 the physical properties of the forward-scatter bistatic radar system are calculated. A radar detection zone is modeled as a simple cone emanating from a target aircraft with some expansion angle extending for some distance. This represents a significant simplification of the actual radar phenomenon but is adequate for the purpose of evaluating the optimization technique that is the focus of this thesis.

In Section 3.3 the simulation model of the system is defined. Several steps are taken to reduce the quantity of aircraft and sensors necessary describe a pseudo continuous problem space. The model consists of scenarios built in STK, a software product from Analytical Graphics, Inc. (AGI). Even after significant reduction, the full problem space is too complex to be analyzed by a single model, so it is broken down into hundreds of smaller scenarios.

Each scenario is comprised of aircraft flying on potential paths from South America to North America via the Caribbean Sea, the GPS constellation available in the STK almanac, and a set of GPS receivers located on the surface of the sea. The scenarios are designed in the Python programming language and executed in STK.

The output from each scenario is read using the R programming language and compiled into sensor-aircraft detection matrices. All detection matrices are then merged together into a full detection matrix and formatted to be read by the International Business Machines Corporation (IBM) C Programming Language Simplex Method (ILOG CPLEX) Optimization Programming Language Integrated Development Environment (OPLIDE).

In Section 3.4 two optimization models are developed for the OPLIDE. These models solve for the minimum number of sensors required to detect all aircraft and for the maximum number of aircraft which can be detected given a fixed number of sensors. The calculations required to solve these problems to optimality quickly overwhelm the computer used. This is because the solution space grows exponentially with linear growth of the problem space. It is shown that to solve the full problem using these exact methods would require billions of times the age of the universe (and a computer that could run continuously for that long).

In Section 3.5 a genetic algorithm (GA) meta-heuristic is developed to generate optimal, or near optimal, solutions for the problem instances where the optimization model becomes too computationally complex. The GA is validated on the known optimal solutions and applied to the full problem. The solutions given by the GA are then compared to projections made using the optimal solutions.

3.2 Radar Properties

Peebels [42] describes radar as an electronic system used for the detection and location of objects. These objects are referred to as 'targets'. The function of a radar is related to properties and characteristics of electromagnetic waves as they react to targets. In the most basic sense, radar systems function by detecting a change in electromagnetic energy. According to Peebles, there are three overall classes of

considerations that focus a radar system design for any new application: those related to system choices, those related to the transmission source, and those concerning the receiver.

3.2.1 Architecture and Configuration Selection.

The system choices involve the architecture, configuration, transmission control, and the medium through which the electromagnetic energy propagates. The architecture aspect considers how the radar functions and integrates with other systems. The architecture controls for the mission and purpose of the radar. The configuration is of great concern in this thesis, relating to whether the system is mono-static, bistatic, or multistatic. Mono-static radars co-locate the transmitter and receiver whereas bistatic and multistatic radars have geographically separated transmitters and receivers. Mono-static and bistatic radar can operate by measuring the back-scatter, or reflected electromagnetic energy, off of a target, or the forward-scatter, the 'shadow' caused by electromagnetic energy reflecting off of a target. Forward-scatter radar is characterized by the reduction of an expected signal. Transmission control refers to whether the system has transmitters under its control, active radar, or if it consists only of receivers, called passive radar. Among passive radar, the transmission source could be cooperative, which would be included in the control and decision making architecture, or non-cooperative, where the receivers use transmitters of opportunity. The medium is the environment through which the electromagnetic energy propagates and may be empty space or may include ground clutter, unwanted reflections, or rain.

The architecture chosen for this thesis utilizes a forward-scatter bistatic geometry. Figure 3 illustrates the shadow created by a target in this geometry. Studied by Griffiths [25] and Willis [60], the shadow can be modeled as a typical radar aperture

originating at the target. This shadow aperture can be described as transmitting a null energy complete with a main lobe and side lobes where the energy from the original transmitter would have been detected.

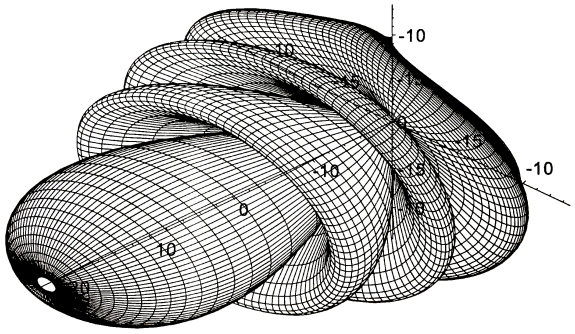


Figure 3. Forward-scatter radar shadow created by a target showing main and side lobes, Meikle [37].

Figure 4 illustrates the geometry of forward-scatter architecture. Only four metrics are necessary to determine the probability of detection of the target via forward-scatter radar. These metrics are the baseline, L, which is the distance between the transmitter and receiver, the distances from the target to the transmitter and receiver, D_t and D_r respectively, and the bistatic angle, β.

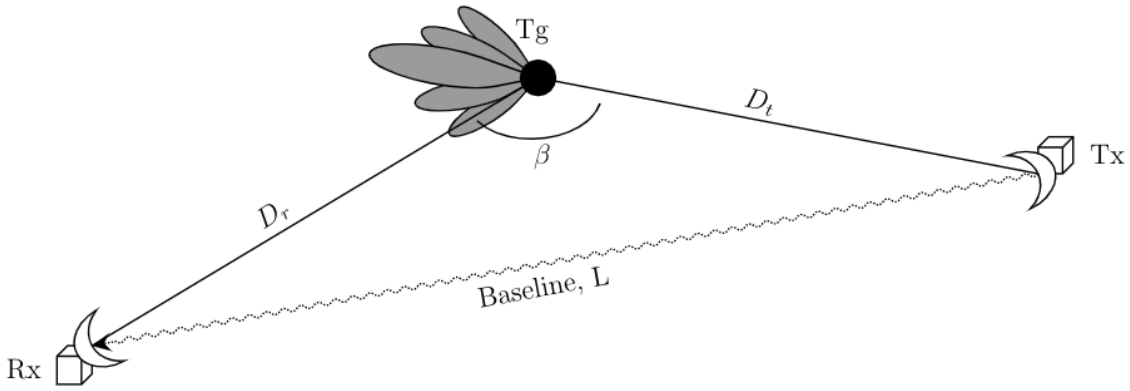


Figure 4. Basic forward-scatter bistatic radar architecture. Tx represents a transmitter of electromagnetic energy. Rx represents a receiver. A baseline is drawn from the Tx to the Rx. A target is shown with its distance from the Tx and Rx denoted as well as the angle β it creates between the Tx and Rx. The forward-scatter shadow is drawn and is projected along the vector connecting the Tx to the target.

3.2.2 Frequency Selection.

The transmission source features include power, frequency, antenna, and waveform. The target, medium, and mission all play a role in determining the transmission features. Power is needed to cover long distances, the carrier frequency and corresponding wavelength of the electromagnetic energy react differently depending on the size and shape of the target and medium. According to Nathanson [38], radar can operate from HF (3 MHz) to millimeter wave (300 GHz). A further breakdown of the uses of each band are analyzed in Table 1. The antenna impacts directionality and gain of the signal. Finally the waveform can be continuous or pulsed, and can be unmodulated or modulated in the amplitude, frequency, phase, or polarization.

Table 1. Common uses of the radar bands available for radar applications as described by Nathanson [38].

HF	Over-the-horizon radar, combining very long range with lower resolution and accuracy. More useful over the oceans.
VHF & UHF	Long-range, line-of-sight surveillance with low to medium resolution and accuracy and freedom from weather effects.
L-band	Long-range surveillance with medium resolution and slight weather effects.
S-band	Short-range surveillance, long-range tracking with medium accuracy. Subject to moderate weather effects in heavy rain or snow.
C-band	Short-range surveillance, long-range tracking with high accuracy. Subject to increased weather effects in light to medium rain.
X-band	Short-range surveillance in clear weather or light rain; long-range tracking with high accuracy in clear weather, reduced to short range in rain.
K_u- & K_a-band	Short-range tracking, real and synthetic aperture imaging, especially when antenna size is very limited and when all-weather operation is not required or ranges are short.
V-, W-, & mm-band	Limited to short ranges in a relatively clear atmosphere, very short ranges in rain. Generally for tracking and missile homing and "smart seekers" with very small antennas. Remote sensing of clouds.

The receiver aspect considers noise sensitivity, antenna, and signal processing methods. The temperature and electronics of the receiver affect the noise sensitivity, or how well the receiver is able to distinguish the intended signal from all other electromagnetic energy in the environment. The antenna design refers to the gain in power received by focusing incoming energy. The signal processing methods, such as detection logic, coherent or non-coherent processing, or the use of multiple pulse integration, allows the receiver to integrate multiple signals to increase its chance of isolating and comprehending the information contained in the received signal.

The architecture in this thesis utilizes a space born transmitter for long-range all weather surveillance in a bistatic forward-scatter configuration for the detection of small aircraft. This criteria narrows the potential transmission sources to those operating in the HF, VHF, UHF, or L-bands to maximize surveillance range and minimize weather effects. As mentioned earlier, the frequency is chosen based on the mission and the target. According to Barton [6], "Most aircraft produce peak forward lobes of some millions of square meters, even at L-band, with side lobes spreading over the entire forward hemisphere. Significant bistatic signal can be received even in lobes as much as 60 dB below the main lobe." Thus an L-band transmitter would be acceptable in a bistatic radar system for detecting aircraft.

3.2.3 Transmitter and Receiver Selection.

The target in this thesis is a small aircraft, and the L-band has been described as adequate to perform the bistatic forward-scatter radar surveillance function [6]. Combining this information and the desire that the transmitter constellation is reliably available with a large set of transmitters in view of a target at all times reduces the list of satellite systems under consideration to the Global Positioning System (GPS) constellation. According to the official US Government information site about GPS

[1], GPS broadcasts three individual transmissions in the L-band, is guaranteed by the United States Air Force to have 95% availability, and currently boasts a minimum of five and an average of eight satellites overhead at any given time.

3.3 Properties of GPS as a Radar Transmission Source

Table 2 shows the properties for three signals broadcast by GPS Block II obtained from the GPS technical documents [22, 23]. The three signals, all operating in the L band are referred to as Link 1 (L1), Link 2 (L2), and Link 5 (L5). A comparison of each GPS link as the transmission source for a bistatic radar was performed by Behar [7] and Sakhawat[51]. They show that the L5 signal is the superior choice for a bistatic radar configuration because the L5 signal is stronger than the L1 signal and the bandwidth is wider than the bandwidth of the L1 signal.

Table 2. Useful GPS signal properties used to determine the capability of a GPS-based bistatic radar system.

Channel	Wavelength, λ (m)	Frequency (MHz)	Bandwidth ΔF (MHz)	Transmitter Power at Surface (dBW)
L1	0.1905	1575.42	20.46	-161.5
L2	0.2445	1227.60	20.46	-164.5
L5	0.2548	1176.45	24	-157.9

As seen later in Equation (3), the bistatic forward-scatter RCS is inversely proportional to the square of the signal wavelength, meaning a smaller wavelength, higher frequency signal produces a larger RCS. Also described later in Equation (13), the maximum detectable range is directly proportional to the root of the transmission power and inversely proportional to the wavelength. Thus, a higher power and shorter wavelength provides the maximum range. A disadvantage of using L5 instead of L1 or L2 is that, because the cone angle is directly proportional to the wavelength, shown in Equation (4), having a smaller wavelength reduces the spread of the detectable region. This thesis uses Behar [7] and Sakhawat's [51] recommendation and analyzes only L5

as the transmission source to model the detection region. Finally, the bandwidth is directly proportional to the signal processing gain which has the same relationship with the maximum range as transmission power as seen in Equation (9). So a higher bandwidth increases the range of the detection cone.

3.3.1 Derivation of the Detection Region.

The objective function of the optimization algorithm developed in this thesis tallies the number of aircraft that are able to complete their entire flight unobserved by a sensor. An aircraft is considered to be observed when it creates a measurable difference in signal amplitude at the receiver. This signal amplitude difference occurs when the target aircraft is near enough to the line-of-sight vector connecting any GPS satellite to a receiver that the target aircraft reflects a 'sufficient' amount of electromagnetic energy that the GPS receiver detects the energy difference. This concept of sufficiency is examined with more rigor later. To compute whether or not an aircraft has been observed, a detection region within which sufficient energy delta occurs is defined for each transmitter-receiver combination. If an aircraft enters this detection region, it is labeled as detected.

I initially define this detection region from the perspective of an aircraft. In Section 3.3 the detection region is redefined from the perspective of the receiver to reduce model complexity. This region models the forward-scatter bistatic radar effect and is analogous to a shadow being cast by the target aircraft. This shadow is simulated mathematically as a cone originating at the target. The shadow cone expands away from the target along the directional vector from the transmitter to the target. The parameters for the shadow region are defined so that there is sufficient change in signal to ensure detection. This is achieved via changing the parameters of the cone of shadow to limit its size in such a fashion that any receiver positioned inside of the

shadow cone has a mathematical probability approaching 100% of detecting the target based solely on the change in transmitter signal amplitude measured at that receiver. Two parameters are used to model each detection cone in this thesis: the cone angle, θ, which is the angle of expansion of the cone away from the target, and maximum distance to the receiver $D_{r_{\max}}$, which is determined by the signal reduction as the cone expands and the shadow diffuses. The next few paragraphs provide calculations for the cone angle and the maximum distance that the cone extends.

3.3.1.1 Detection Region Diffraction Pattern.

The wavelength of a transmitted signal, the size of the target, and the distances that separate the target from the transmitter or receiver are used to determine what kind of scattering effect the target has on the signal. According to Hecht [28], there are two significant types of scattering that may occur.

The first type of scattering is Fresnel diffraction, which occurs in the near-field, and is characterized by constructive and destructive interference patterns caused by obstacles in the signal path. Ufimtsev [56] shows that this type of diffraction is more difficult to model because the wave front must be modeled as spherical rather than planar. This diffraction region is even more complicated in the forward-scatter bistatic configuration because of the interaction of shadow radiation with the original waves. Also, the phase differences are non-constant for a curved wave front causing the amplitudes of each wave front to vary from point to point, increasing the complexity of the model.

The second type of scattering is Fraunhofer diffraction, or far-field diffraction, which occurs when the distances from the target to the transmitter or receiver differ much less than the wavelength of the signal. In Fraunhofer diffraction, the propagation patterns for each wavelet can be treated as parallel.

The Fresnel Number, F, as defined by Hecht and Peebles [42], is used to determine in which diffraction region a target resides. A Fresnel number greater than 1 indicates target is in the Fresnel region and less than 1 indicates target is in the Fraunhofer region. The Fresnel number is calculated using Equation (1) where a is the largest single dimension of the target and λ is the wavelength of the signal. The targets under consideration are Cessna-172 aircraft with height $h = 2.72$ m and length $l = 8.3$ m.

$$F = \frac{a^2}{D_r \lambda} \tag{1}$$

The radar equations used in this thesis assume the target is always flying in the Fraunhofer diffraction zone. To prove this, the boundary between the near- and far-field is found by setting the Fraunhofer number in Equation (1) to 1 and solving for the distance. Since the transmitter is located in a Medium Earth Orbit and the target is in the Earth's atmosphere, it is assumed that the receiver is always closer to the target than the transmitter. Thus, only the distance between the target and the receiver, D_r, need to be considered. Equation (2) is used to solve for the boundary between Fresnel and Fraunhofer diffraction regions. The targets are in the Fraunhofer diffraction zone if their distances to each receiver are greater than $D_{r,\text{boundary}}$

$$D_{r,\text{boundary}} = \frac{a^2}{\lambda} \tag{2}$$

$$D_{r,\text{boundary}} = \frac{8.3^2}{0.2548} \approx 270 \text{ m}$$

According to Equation (2), the boundary between the near and far-field is approximately 270m. The targets in this model are aircraft being flown at an altitude of 3050m and the receivers are located at sea level; thus, the forward-scatter radar equations for the Fraunhofer diffraction zone are appropriate.

3.3.1.2 Radar Cross Section Within the Detection Region.

Griffiths [25], Hecht[28], and Willis [60] define the bistatic metrics required for this model, the bistatic forward-scatter radar cross section (RCS) and the cone angle of the detection zone. The bistatic forward-scatter RCS, σ_f, as measured in the Fraunhofer diffraction zone is shown in Equation (3), where A is the cross sectional area of the target. The forward-scatter bistatic RCS is one of the parameters required to determine the change in signal amplitude caused by the target.

$$\sigma_f = \frac{4\pi A^2}{\lambda^2} \tag{3}$$

$$\sigma_f = \frac{4 \times \pi \times (8.3 \times 2.72)^2}{0.2548^2} = 98651.8 \text{ m}^2$$

Evaluating Equation (3) shows that the aircraft in a forward-scatter bistatic radar configuration create an incredibly large RCS. This is beneficial because the ability to detect a target with radar increases with RCS.

One of the advantage of this large RCS is that small targets are easier to detect in a forward-scatter configuration. Another advantage is that a target may be detected at farther distances from the receiver than in a back-scatter configuration.

Among the disadvantages is that other small objects, such as birds, may also have a large RCS, causing a high false positive detection rate. Another disadvantage is that the forward-scatter effect is only detectable in a narrow area extending out from the target. The spread of this area is directly related to the wavelength of the signal. According to Davis [18], GPS L5 is operating at a much smaller wavelength than traditional forward-scatter bistatic radar systems, restricting the detectable region even further. Equation (4) is used to calculate the cone angle, or spread, of this detectable region.

3.3.1.3 Angular Width of the Detection Region.

The cone angle of the detection zone is equal to the total angular width of the main diffraction lobe plus two side lobes of the signal scattered by the target. The cone angle is calculated using Equation (4). Solving Equation (4) shows that the forward-scatter radar effect casts an approximately 8.8 degree conical shadow along the pointing vector from the transmitter to the target.

$$\theta_f = \frac{5\lambda}{a} \tag{4}$$

$$\theta_f = \frac{5 \times 0.2548}{8.3} = 0.153494 \text{ rad}$$

$$\theta_f = 0.153494 \times \frac{180}{\pi} \approx 8.8 \deg$$

3.3.1.4 Maximum Detectable Distance of the Forward-Scatter Effect.

As this cone expands away from a target, the forward-scatter radar effect diffuses. Thus, the strength of the signal amplitude drop diminishes with distance from the target. According to Behar [7], the bistatic radar equation shown in Equation (5), as provided by Willis [60], can be simplified and rewritten for the forward-scatter case.

$$(D_r D_t)_{max} = \left[\frac{P_t G_t G_r \lambda^2 \sigma_b F_t^2 F_r^2}{(4\pi)^3 K T_s B_n (SNR)_{min} L_t L_r} \right]^{\frac{1}{2}} \tag{5}$$

Behar and Sakhawat [7, 51] state that for the bistatic forward-scatter geometry rather than using Equation (5) one should walk through the logic to independently derive the equation via its constituent parameters.

Towards that end working through Behar's logic, and modeling each target as an transmission source of shadow radiation, the power spectral density (PSD) at the

output of an omni-directional antenna near the Earth's surface can be determined using Equation (6).

$$PSD = D_1 = \frac{4\pi P_t}{\lambda^2} \tag{6}$$

Where P_t is the GPS L5 signal power found in Table 2. The power of the signal reflected from the target, P_{tg} is found by multiplying the PSD by σ_b, the bistatic RCS as in Equation (7).

$$P_{tg} = \frac{4\pi P_t \sigma_b}{\lambda^2} \tag{7}$$

The signal power at the output of the receiver antenna to the signal processor depends on the antenna gain per Equation (8).

$$P_{rec} = \frac{P_t G_r \sigma_b}{4\pi D_r^2} \tag{8}$$

Sakhawat [51] and Behar calculate the receiver noise, N_r to be approximately -131 dB, using Equation (9).

$$N_r = kT\Delta F = KTB_n \approx -131dB \tag{9}$$

Combining Equation (8) and Equation (9) to determine the Signal to Noise Ratio (SNR) at the receiver, SNR_{rec} produces Equation (10).

$$SNR_{\text{rec}} = \frac{P_{\text{rec}}}{N_r} = \frac{P_t G_r \sigma_b}{4\pi N_r D_r^2} \tag{10}$$

Both authors further postulate that the signal could be improved through circular cross-correlation by multiplying by a signal processing gain, G_{SP}, calculated using Equation (11) from Richards [49], where T_{Q5} is the period of the Q component of the

GPS L5 signal. GPS L5 has two components that could be used, an I component and a Q component. The I component is modulated by a 10-bit Neuman-Hoffman code and the Q component is modulated by a 20-bit Neuman-Hoffman code. Each bit takes 1 ms, resulting in a 20 ms period of the Q component [22, 23].

$$G_{SP} = \Delta F T_{Q5} \tag{11}$$

Substituting Equation (11) and Equation (3) into Equation (10) yields Equation (12).

$$SNR_{\text{det}} = \frac{P_t G_r (hl)^2 G_{SP}}{\lambda^2 N_r D_r^2} \tag{12}$$

Solving Equation (12) for D_r produces Equation (13).

$$D_r = \frac{hl}{\lambda} \sqrt{\frac{P_t G_r G_{SP}}{N_r SNR_{\text{det}}}} \tag{13}$$

Both authors conclude that a minimum signal-to-noise ratio to detect aircraft with a forward-scatter bistatic radar configuration using GPS L5 as the transmission source is 20 dB. Solving Equation (13), Behar calculates that for a Cessna-172 with $h = 2.72$ m, $l = 8.3$ m, $G_r = 25$ dB, and $SNR_{min} = 20$ dB, the maximal range of detection is, $D_{r,max} = 7112$ m.

3.3.2 Summary of GPS-Based Forward-Scatter Radar Properties.

In summary, for each GPS satellite in view of a target aircraft, a detectability zone can be represented as a cone emanating from the aircraft with an angle of 8.8 degrees and extending for 7112 m.

3.4 Sensor and Aircraft Spacing Model

The goal of the scenario model is to produce a detection matrix that has potential sensor locations as columns and potential target aircraft as rows. The size of the detection matrix determines the computational complexity of both the model used to produce it and the algorithms that solve it. Therefore, the model fidelity is reduced in this section to be as small as possible while still providing a pseudo-continuous representation.

Since the location and time any target aircraft may attempt a crossing is unknown, I develop a model to account for aircraft travelling at all possible locations and crossing a fence at all times of day. To simplify the model I assume all aircraft travel in straight paths from South to North, however; even with this assumption the computational complexity of building such a model with continuous inputs is beyond the capability of the software being used and the computational methods being developed. Thus, the aircraft are represented as discrete events spaced in such a way that the resolution of these events is indistinguishable in fidelity versus a continuous allowable spacing. i.e., they are pseudo-continuous.

Because the detection cones cover the smallest area at the target altitude when pointing straight up, each reduction in scope is determined assuming one satellite is directly overhead. This section begins by defining the minimum longitudinal spacing for sensors and aircraft such that the detection cones emanating from adjacent sensors touch but do not overlap. Next, the temporal spacing of aircraft is determined so that the detection cone has moved a maximum distance between consecutive aircraft while still guaranteeing all aircraft are detected. Finally, the latitudinal spacing of sensors is calculated using the distance travelled by an aircraft on a single flight path before the next aircraft on that path launches.

Figure 5 illustrates the area under consideration, highlighted in red, given the data

that was able to be produced. For the rest of this thesis, each map representation is constrained to this area of interest.

Figure 5. Map of Caribbean Sea with area of interest highlighted in red

3.4.1 Initial Sensor Spacing.

The first step in building such a model is to determine the maximum distance any sensor can be located from its neighbors without creating a gap in coverage. The minimum width of a sensor cone at aircraft altitude is calculated using the detection cone angle from Equation (4) and assuming the target aircraft fly at an altitude of 3048 meters. Equation (14) shows the width of a detection cone at target altitude when a satellite is directly above the sensor. Figure 6 illustrates this minimum

spacing geometry. Therefore, each sensor must be spaced at most 469 meters apart longitudinally.

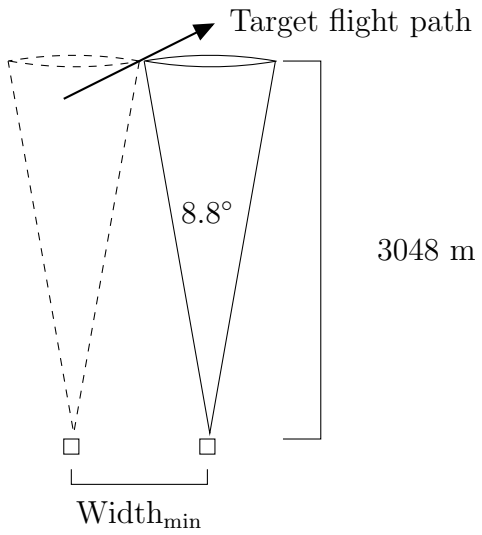


Figure 6. Minimum longitudinal sensor spacing is determined by detection cone width at target altitude. The minimum width creates adjacent but non-overlapping cones at target altitude when a single satellite is directly overhead.

$$\text{Width}_{\text{min}} = 2 * 3048 \tan 4.4^\circ = 469\ m \tag{14}$$

The radius of the Earth, R_e, at the equator is approximately 6,378,137 meters [57]. Moving away from the equator, the distance of one degree longitude must be multiplied by the cosine of the latitude as seen in Equation (15).

$$\text{Dist}_{\text{long}} = \frac{2\pi R_e}{360} \cos \text{lat} \tag{15}$$

Thus, at 14 degrees latitude, 1 longitudinal degree equates to about 108,013 meters and 1 latitudinal degree to about 110,644 meters. To guarantee coverage from longitudes between 75 and 74 degrees West and latitudes between 14 and 16 degrees North given a single satellite directly overhead, which represents minimal detection cone coverage, there would need to be 230 sensors per degree longitude and 236 sen-

sors per degree latitude, totalling 108,560 sensor locations in the model. Accounting for the 31 individual GPS satellites in the STK almanac, each being modeled as a separate observation cone extending from each sensor site, there would be a total of 3,365,360 individual observation cones in the model.

Assuming each aircraft follows a South to North trajectory, a single row of 230 sensors at a latitude of 14 degrees North guarantees that a complete 'fence' is modeled and all aircraft passing through it are detected. One goal of this thesis is to reduce the number of sensor locations by taking advantage of the rotation of the earth and the orbital motion of the satellites. Placing sensors at different latitudes allows for the satellites to change relative position to the sensors as an aircraft makes its way north. The fidelity of this motion is discussed in Section 3.4.3 and Section 3.4.4.

At a latitude of 16 degrees North the length of 1 degree longitude is 107,007 meters, less than 1 kilometer shorter than at a latitude of 14 degrees North. This distance would require 2 fewer sensors to provide complete coverage based on the single satellite always directly overhead scenario proposed in this section. To simplify the computations in the model and detection matrix, the worst case scenario of 230 sensors per degree latitude was continued at each latitude evaluated even though 228 is sufficient at 16 degrees latitude. To achieve this, the first sensor on each line of latitude was placed at a longitude of 75 degrees West and each additional sensor was place 1/230th of a degree to the East along the same line of latitude.

3.4.2 Aircraft Longitudinal Spacing.

Figure 7 illustrates the coverage at a fixed point in time for a single sensor site. The site location is denoted by the yellow dot on the ocean surface. Each yellow cone is directed towards a different GPS satellite, expands at 8.8 degrees, and extends for 7,112 meters as concluded in Section 3.2.4.5. A target aircraft is marked by a red

dot. It follows the red flight path from South to North. As the scenario progresses in time, the cones track their respective satellites in an easterly fashion and the aircraft to travel north along their flight paths.

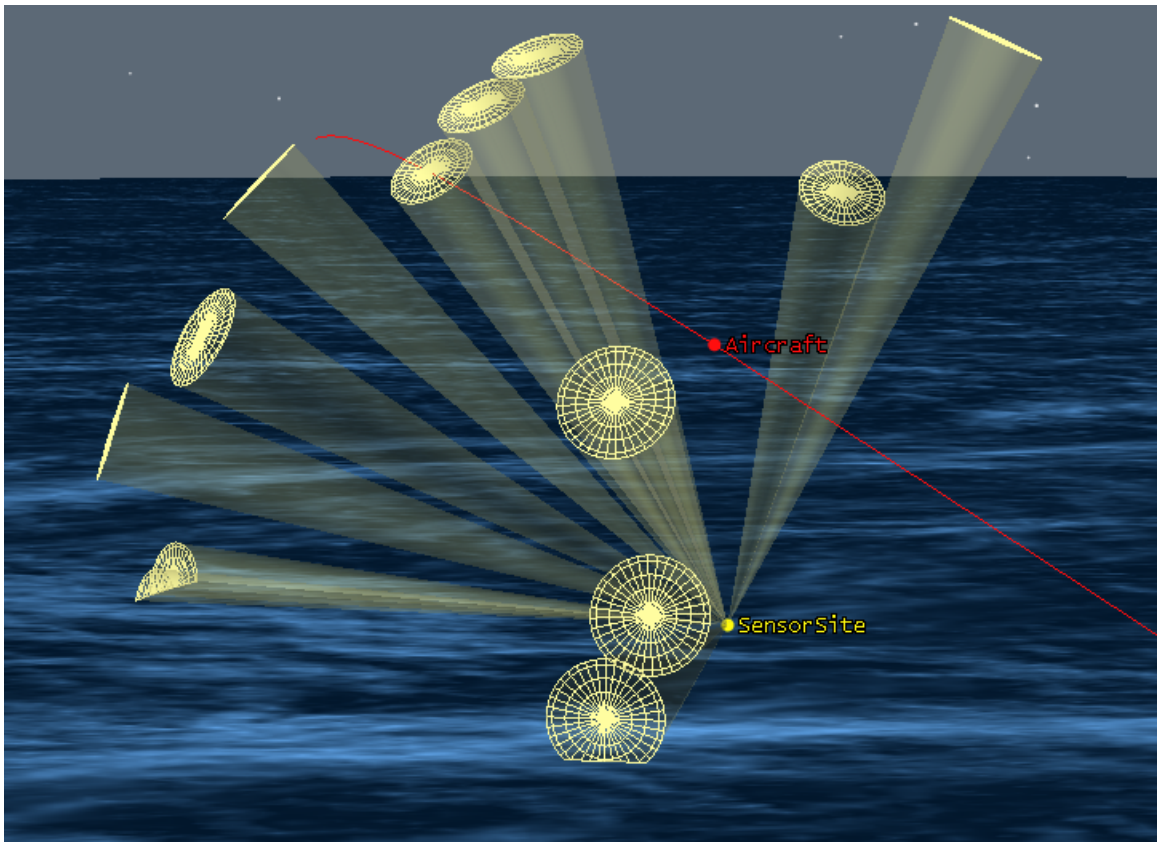


Figure 7. Example sensor site with observation cones in yellow pointing to GPS satellites in view with target aircraft path illustrated by a red line and current target location by a red dot.

The aircraft in this model follow South to North flight paths spaced the same longitudinal distance apart as the sensors. The flight paths start and end at the farthest southern and northern extents that a sensor in the model could detect them. The Pythagorean Theorem is used in Equation (16) to calculate the maximum horizontal range of a sensor knowing each sensor has a maximum line of sight range of 7112 meters and the targets are flying at 3048 meters.

$$\text{Max Horizontal Distance} = \sqrt{7112^2 - 3048^2} = 6425\text{m} \tag{16}$$

The maximum horizontal range of a sensor is 6425 meters and is illustrated in Figure 8.

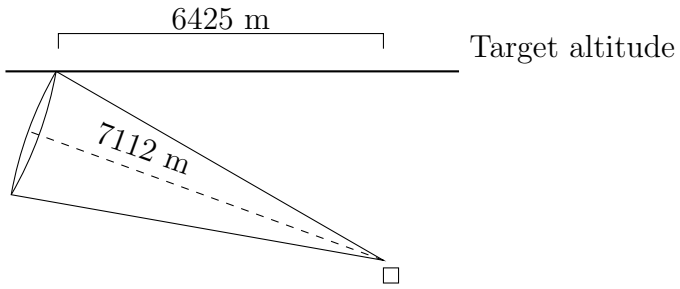


Figure 8. Maximum horizontal distance from sensor location an aircraft flying at 3048 meters altitude can be detected from a sensor site.

Using Equation (15) at a latitude of 14 degrees North the maximum horizontal range is 0.058 degrees. I add the maximum horizontal range to the northern and southern edges of the sensor field so that any potential detection north or south of the sensor field is collected. Thus, in the model there are 230 South to North flight paths per degree longitude from 75 to 74 degrees West, each separated by 469 meters, that start at 13.942 degrees North and end at 16.058 degrees North.

Figure 9 shows the current state of the model. With a maximum horizontal range of 6425 meters and a separation of 469 meters I calculate that 13 consecutive sensors placed East or West of an aircraft may detect that aircraft. Thus, for each target, there are 13 sensors on either side and one directly beneath aligned in 236 rows per latitude for a total of 6,372 sensor locations that could potentially detect that target along its path.

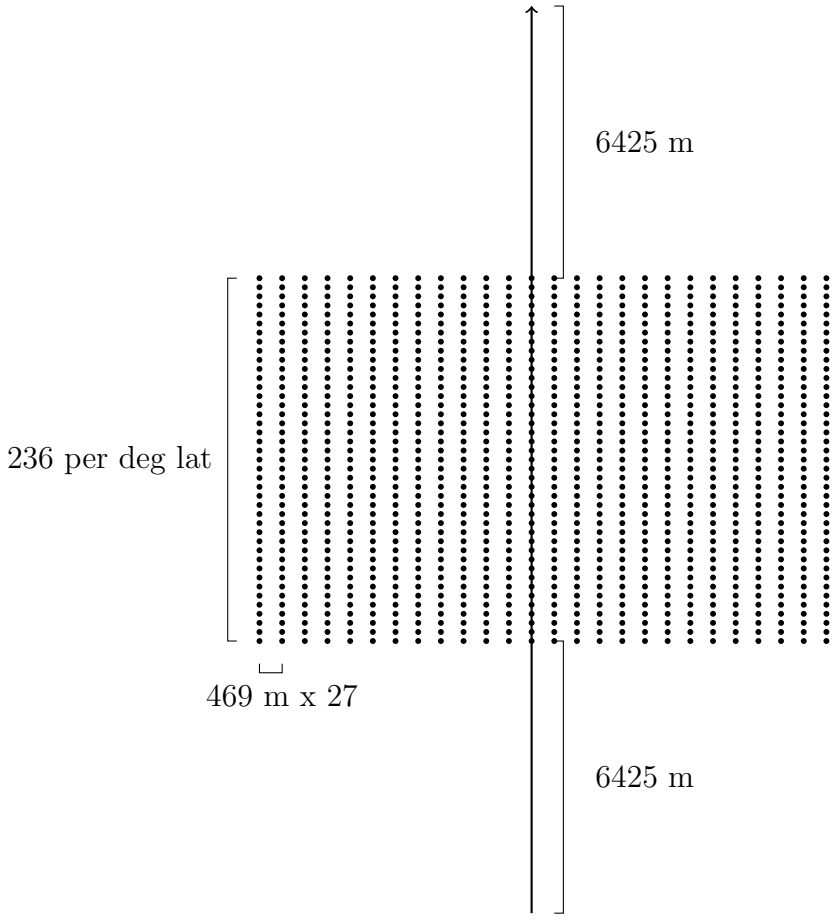


Figure 9. Model description for one flight path, shown as a solid line, with apparent sensor locations shown as dots.

3.4.3 Aircraft Temporal Spacing.

To model the discrete aircraft as near continuous over the time frame being analysed, there would need to be 462 meters, the minimum width of a sensor cone at aircraft altitude, between consecutive aircraft on each flight path. Assuming the aircraft are flying at a constant 188 miles per hour (or 84 meters per second) a 5.5 second delay between aircraft launches is required. The total number of aircraft in the model then depends on the duration of the model and the number of individual flight paths under consideration.

The duration of the model is determined by the time it takes each GPS satellite

to return to a relative point in the sky from the perspective of a viewer at sea level. GPS satellites orbit the Earth twice per sidereal day [1], not to be confused with a solar day. The Earth's rotation with respect to the equinox is called sidereal time while the Earth's rotation with respect to the sun is called solar time. A sidereal day is measured relative to ostensibly fixed stars and consists of approximately 23 hours 56 minutes and 4 seconds or 86,164 seconds [57]. Since the satellites are moving at twice the rotational speed of the Earth, from an observational point on the surface of the Earth they appear to move at 1 full revolution per sidereal day.

Therefore the observational period is one sidereal day. Having aircraft fly due north along 230 evenly spaced trajectories per degree longitude, each spaced 462 meters or 5.5 seconds apart equates to 15,666 aircraft modeled per flight path, totalling 3,603,180 aircraft per degree longitude in the model. To reduce this number, I take into consideration the fact that the observational cones change position throughout the day.

As the satellites fly overhead, the observational cones sweep West to East with some motion north, south, or both depending on the location of a satellite in its orbit. Since the aircraft are flying South to North, it is the West to East motion of the observation cones that is of concern. Given any arbitrary point in time, as a cone moves East its western trailing edge eventually lies along its original eastern leading edge as shown in Figure 10. This new cone position lies adjacent to the original cone position but has no overlap.

In a worst case scenario, the observation cone is moving due East and moves across a flight path with the shortest amount of time possible. I redefine the time between aircraft launches to be this minimum time required for an observation cone from a sensor to sweep eastward to an adjacent location, thus, the minimum time that a cone may spend observing a flight path.

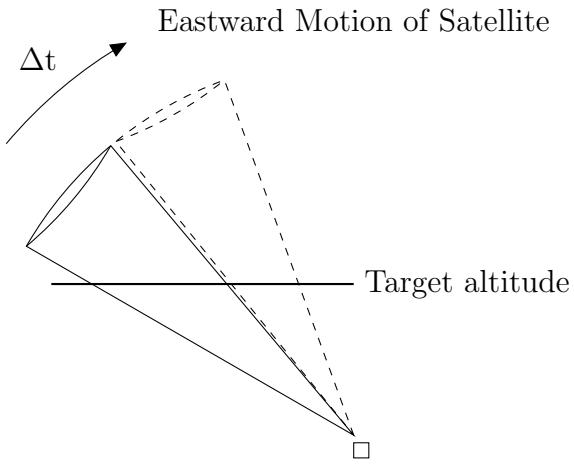


Figure 10. Eastward motion of an observation cone as satellite moves across a sensor field of view. An aircraft flying at the target altitude on a path that extends directly out of the page would only be in view of the sensor for time Δt.

A fixed Earth reference frame is used to calculate the minimum time it would take for an observation cone to move from West to East by 8.8 degrees, the width of the cone. The orbital radius of GPS satellites, R_{GPS}, is approximately 20,222 km [52] and the radius of the Earth, R_e, is approximately 6,378 km [57]. The time to achieve an 8.8 degree sweep from the perspective of a sensor located at sea level is calculated by considering right triangles measured from the center of the Earth and the observation point out to a satellite moving in a purely west to East manner, as shown in Figure 11.

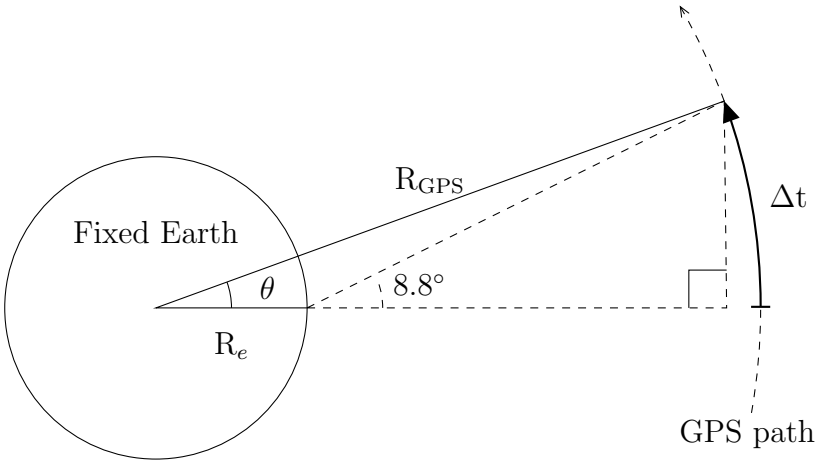


Figure 11. Relative GPS motion from sensor location about fixed Earth as viewed from North Pole

The orbital angular progression is calculated using Equation (17) to be approximately 6.01 degrees.

$$\theta = \sin^{-1} \frac{(R_{GPS} - R_e) \sin 8.8^\circ}{R_{GPS}} = 6.01^\circ \tag{17}$$

Since it was derived using a fixed Earth reference frame, the time required for a GPS satellite to travel that far can be easily be determined using an angular rate of 1 revolution per sidereal day. This elapsed time, Δt, is calculated using Equation (18) to be about 24 minutes.

$$\Delta t = \frac{86,164\ \theta}{360} = 1438.9 \text{ s} \approx 24 \text{ min} \tag{18}$$

Therefore, along each flight path, aircraft can be launched every 24 minutes, as opposed to every 5.5 seconds, and still achieve the same coverage. To provide clear overlap of potential aircraft flights, the temporal aircraft spacing in this model is set to be 20 minutes along each flight path, reducing the total number of aircraft in the model from over 3.6 million to 23 thousand per degree longitude.

3.4.4 Sensor Latitudinal Spacing.

At this point the model still consists of 236 sensors per degree latitude. Employing a similar methodology to that used to reduce the number of aircraft in the model, I am able to limit the number of required sensors South to North. I do this again by considering the time necessary for a sensor cone to move to an adjacent eastward location and the minimum distance travelled by an aircraft during that time.

Assuming the aircraft are travelling at 84 meters per second and launching at 20 minute intervals, the latitudinal spacing of the sensor sites can be modeled to fall within this aircraft spacing window. That is to say, by the time an aircraft passes directly over a second sensor site, another aircraft on that same flight path has already entered the system. In 20 minutes, at 84 meters per second, a target travels 100,800 meters or 0.91 degrees latitude at an original latitude of 14 degrees North.

Again, to provide some margin of overlap of the targets while still reducing the number of sensors, a latitudinal spacing of 0.5 degrees latitude is used in this model as opposed to 469 meters. The sensors are located at latitudes of 14, 14.5, 15, 15.5, and 16 degrees North, thereby reducing the total number of sensor sites from 180,560 down to just to 1,150 per degree longitude. The reduced model is demonstrated in Figure 12.

Thus, the output from the reduced model is a detection matrix with 1,150 potential sensor locations and 16,560 aircraft per degree longitude. There are now only 675 million sensor-aircraft-satellite interactions per degree longitude that need to be determined in the model as opposed to 31 billion from the unreduced system.

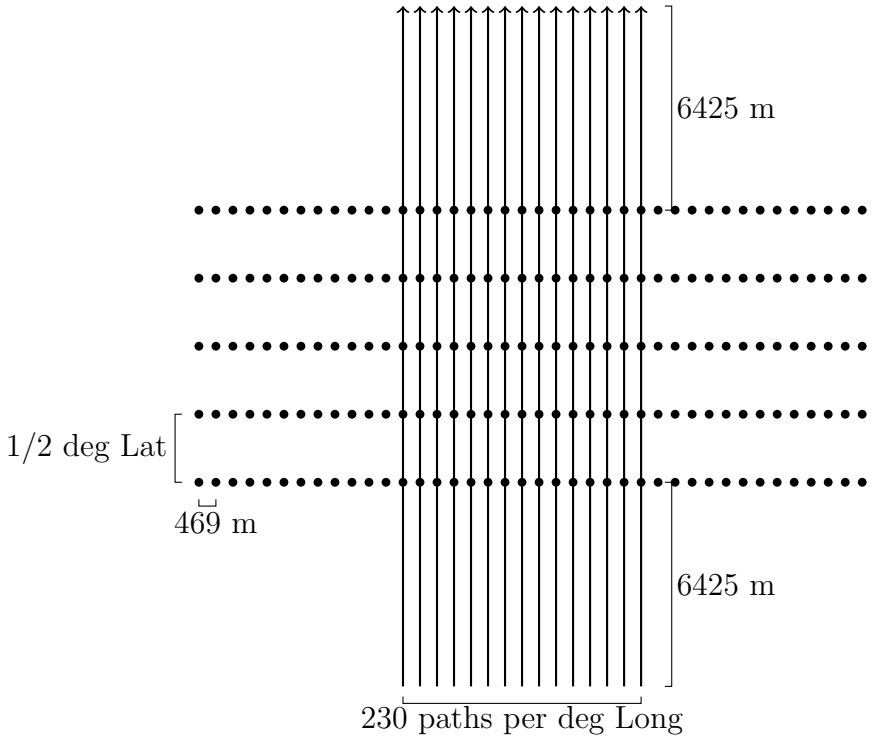


Figure 12. Final model configuration with flight paths shown as solid lines and sensor locations shown as dots.

3.5 Optimization

Two optimization problems are explored in this thesis using only the detection matrix as a data source. The first is a minimal set covering problem which solves for the fewest number of sensors required to detect all aircraft. The answer to this problem also provides the locations where the sensors should be placed to achieve the goal of detecting all aircraft. The second is a maximal set covering problem which solves for the maximum number of aircraft that can be detected given a finite number of sensors. This section discusses the numeric and meta-heuristic solution methods applied to each of the set covering problems.

The minimal set covering problem is modeled as an integer optimization problem and solved using the IBM CPLEX OPLIDE. Optimization problems are defined by parameters, decision variables, an optimization function, and constraints.

The minimal set covering problem is solved first and the solutions are used to tailor the inputs of the maximal set covering problem. On very small subsets of the data, a numeric approach is capable of finding the optimal solution or solutions to each of the set covering problems under consideration. However, due to the combinatorial nature of the solution space, the computational complexity grows at a nonlinear rate and quickly becomes intractable with modern methods and technology. I show in Section 4.3.2 how the computational time needed to solve a set covering problem using numerical methods on this data would exceed the age of the universe.

In the instances where an exact solution was unobtainable, the numeric approaches were allowed to run for a finite amount of time after their solutions stopped improving. This method of limiting the computational duration of a numeric solution produced interesting results when compared to the results from the meta-heuristic.

A genetic algorithm meta-heuristic search method was tailored to solve the set covering problems on the full data set. Because there is currently no way to prove that a solution from a meta-heuristic is optimal without using a numeric method, I use known optimal solutions from subsets of the data to validate the solutions generated by the meta-heuristic.

Finally, the solutions found by aligning the sensors in geometric patterns were evaluated against those found by the numeric and meta-heuristic approaches. These solutions can be used to identify potential heuristics for placing sensors outside of the area of interest studied in this thesis.

3.5.1 Numeric Solution Method for Minimal Set Covering Problem.

The parameters of the minimal set covering problem are the aircraft, the potential sensor sites, and the detection matrix. The decision variables are the choices to include any sensor in the solution. The objective function in Equation (19) minimizes the

total number of potential sensor sites chosen. The constraints shown in Equation (20) guarantee that every aircraft is detected by at least one sensor.

Parameters

$N_a \in \mathbb{Z}_+$	Number of aircraft
$N_s \in \mathbb{Z}_+$	Number of potential sensor sites
$i \in I = \{1, \ldots, N_a\}$	Aircraft
$j \in J = \{1, \ldots, N_s\}$	Sensor Sites
$d_{ij} \in \{0, 1\}$	1 if a sensor at site j can detect aircraft i, 0 otherwise

Decision Variables

x_j	Decision to place a sensor at site j

Objective Function

$$\text{minimize} \quad \sum_{j \in J} x_j \tag{19}$$

Constraints

$$\text{subject to} \quad \sum_{j \in J} d_{ij} x_j \geq 1 \qquad \forall\, i \in I \tag{20}$$

$$x_j \in \{0, 1\} \qquad \forall\, j \in J$$

3.5.2 Numeric Solution Method for Maximal Set Covering Problem.

The parameters of the maximal set covering problem are the aircraft, the potential sensor sites, the detection matrix, and the exact number of sensors that must be in the solution. The decision variables are the choices to include any sensor in the solution. The objective function in Equation (19) minimizes the total number of undetected aircraft. The constraints shown in Equation (22) and Equation (23) guarantee that

the solution contains the exact number of sensors required.

Parameters

$N_a \in \mathbb{Z}_+$	Number of aircraft
$N_s \in \mathbb{Z}_+$	Number of potential sensor sites
$M_s \in \mathbb{Z}_+$	Number of sensors allowed in solution
$i \in I = \{1, \ldots, N_a\}$	Aircraft
$j \in J = \{1, \ldots, N_s\}$	Sensor Sites
$d_{ij} \in \{0, 1\}$	1 if a sensor at site j can detect aircraft i, 0 otherwise

Decision Variables

x_j	Decision to place a sensor at site j
g_i	1 if aircraft i undetected by all sensors in solution, 0 otherwise

Objective Function

$$\text{minimize} \quad \sum_{i \in I} g_i \tag{21}$$

Constraints

$$\text{subject to} \quad g_i + \sum_{j \in J} d_{ij} x_j \geq 1 \qquad \forall\, i \in I \tag{22}$$

$$\sum_{j \in J} x_j = M_s \tag{23}$$

$$x_j \in \{0, 1\} \qquad \forall\, j \in J$$

$$g_i \in \{0, 1\} \qquad \forall\, i \in I$$

3.5.3 Meta-heuristic Search Algorithm.

A genetic algorithm is the meta-heuristic used in this thesis to optimize the minimal and maximal set covering problems. This section describes how the parameters of the genetic algorithm are tuned and their impact to solution convergence and run time. The tunable parameters are population size, initialization, fitness functions, elitism, selection, crossover, and mutation. The R package *GA* from Scrucca [54] provides a convenient framework for creating and tuning a genetic algorithm. Each parameter described in this section is coded and used by the *GA* package. Reeves [48] outlines the general pseudo-code for a genetic algorithm in Algorithm 1.

Algorithm 1 Genetic Algorithm [48]

```
Choose an initial population of chromosomes
while termination condition not satisfied do
  if crossover condition satisfied then
    {select parent chromosomes; choose crossover parameters; perform crossover};
  end if
  if mutation condition satisfied then
    {choose mutation points; perform mutation};
    evaluate fitness of offspring
  end if
  if sufficient offspring created then
    select new population
  end if
end while
```

As described by Reeves [48], a genetic algorithm consists of a population of solutions called *chromosomes*. The individual entries in a chromosome are referred to as *genes* and the possible values that can fill a gene are known as *alleles*. The chro-

mosomes in this thesis consists of genes representing each potential sensor site and alleles being the binary decision whether or not to place a sensor at that site. In an iteration of the genetic algorithm each member of the population is evaluated for fitness using some fitness function. The members are then recombined using simple analogies for genetic crossover and mutation. The members being combined to make a new population are chosen by some method involving their fitness with competing goals of continuing to explore the solution space while improving the average fitness of the population.

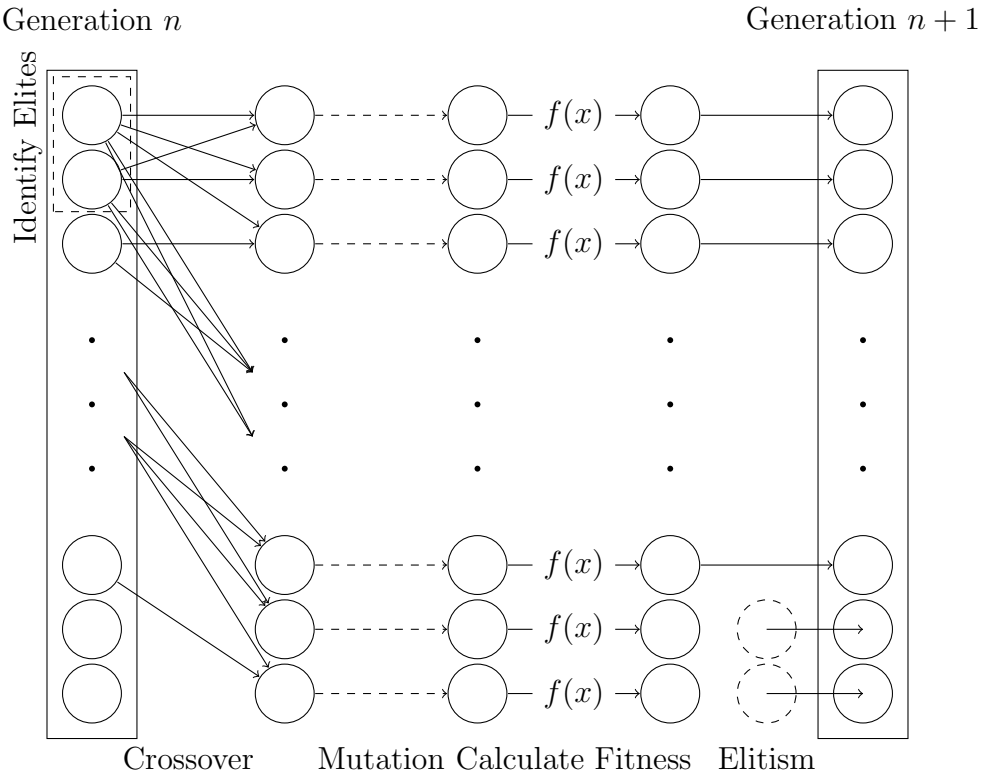


Figure 13. Single iteration of a genetic algorithm with chromosomes represented by circles

Figure 13 illustrates the steps involved in a single iteration of a genetic algorithm. In the first step, the chromosomes of the population are ranked by fitness with the fittest members represented at the top. Elite member are identified to be carried over into the next generation. Children are created from two randomly chosen parent

chromosomes via a crossover method. The parents are selected using a linear rank weighting in which fitter chromosomes have a higher probability of being selected to mate. It is possible for some members to never be selected for mating. Once enough children are created to replace the population, their genes are mutated and their final fitness is calculated. Finally, elite members from the previous generation replace the least fit children.

3.5.3.1 Population Size.

The population size, N_{pop}, trades solution quality for computations required. If the population is too small, it runs the risk of failing to adequately search the solution space and converging too early, whereas a large population may increase computational time so greatly that a different meta-heuristic, such as simulated annealing or tabu search, may be more appropriate [48]. In Reeves concluded in earlier work [47] the minimum population size of a q-nary string to be the amount sufficient such that every possible point in the search space is reachable from the initial population via crossover alone given a uniform distribution . The minimum population size required to achieve a probability P_q^* that at least one instance of every allele at each locus is found within the initial population can be found using Equation (24) where q is 2 for a binary problem, L is the length of a solution string, and $S(N_{\text{pop}}, q)$ is a Stirling number of the second kind.

$$P_q^* = \left\{ \frac{q! S(N_{\text{pop}}, q)}{q^{N_{\text{pop}}}} \right\}^L \tag{24}$$

From Equation (24), a minimum of 20 members are required to reach 99% confidence and 22 members are required for 99.9% confidence of fully covering the solution space with the data contained within the initial population. A rough value for these numbers was determined using the mathematical software Mathematica, and the

nearby integer solutions were evaluated for accuracy. This contradicts findings from Alander [2] which state that $N_{\text{pop}} = 2L$ is ideal, suggesting 2,560 members would be required in the initial population.

With the ability to compute the fitness of each member of the population in parallel, given in an add-on to the *GA* package [53], the processing time for the genetic algorithm does not increase by more than a small overhead margin until the number of processors in parallel is exceeded. The computer used in this thesis has 4,400 processors available, thus any population size smaller than or equal to 4,400 members does not adversely affect the computational time. A larger population size is advantageous for this thesis because the fitness functions include the sum of all values in a chromosome, meaning several solutions may have the same objective value based solely on their genes but are differentiated by the count of genes with an allele value of 1. This is particularly difficult in the maximal set covering problem where the sum across a chromosome must equal a predetermined value. I address this issue by considering the genetic makeup of the initial population.

3.5.3.2 Initial Population.

Reeves [48] defines a *schema* as a subset of genes in which all chromosomes share a particular set of defined values. Reeves discusses using an experimental design to cover the solution space effectively by maximizing the number of schema present in the initial population [47] rather than just ensuring that every possible gene expression is present.

For the minimal set covering problem, considering I can have up to 4,400 members in the initial population and the genes can be arranged in groups of five, each sharing the same longitude, I can repeat every permutation for five longitudinal groups of five sensors each. This produces $5^5 = 3125$ patterns from which to create unique

initial members however it excludes schema containing multiple sensors in a single longitudinal row.

For the maximal set covering problem, since the number of sensors, M_s, in a feasible solution is fixed, it is beneficial to seed the initial population with chromosomes containing exactly k activated genes. Using the criteria defined in Section 3.5.3.1, that every allele for each gene should be represented at least once in the initial population, M_s determines the absolute minimum population size via Equation (25). A minimum population seeded in this way, while guaranteeing that all solutions are reachable from the initial population, does not provide much diversity in how the genes are combined.

$$N_{\text{pop, min}} = \frac{\text{length}_{\text{chromosome}}}{M_s} \tag{25}$$

To encourage diversity, the initial population for the maximal set covering problem was originally seeded so that each chromosome has M_s activated genes and each gene is activated at least $N_{\text{pop}}/N_{\text{pop, min}}$ in the population. In subsequent tests, seeding the initial population with geometric patterns and randomly deactivating genes until a desired gene count was reached yielded superior results. The patterns used created a saw-tooth shape by choosing a different latitudinal gene index for each adjacent collection of longitudinal genes, and a linear pattern by choosing the same latitudinal gene index for each adjacent collection of longitudinal genes.

3.5.3.3 Parent Selection.

Whitley [4] describes a rank based approach to determining which chromosomes are selected for reproduction. Utilizing rank, as opposed to fitness, reduces premature convergence in the presence of *super individuals*. Super individuals are chromosomes whose fitness is so much higher than their peers that they are chosen to produce a

large number of offspring, preventing others from contributing to the next generations. After only a few generations, a super individual may completely eliminate a desirable schema causing convergence on a local optimum.

The linear rank selection method is used to select chromosomes to pass information to the next generation. The rank of each member is determined by sorting them from highest fitness, having a rank of 1, to lowest fitness, having a rank of N_{pop}. The linear function in Equation (27) has the user defined parameter q.

$$\text{rank} = (N_{\text{pop}} + 1) - \text{order}_{(\text{fitness, descending})} \tag{26}$$

$$p = q - (\text{rank} - 1)r \tag{27}$$

Noting that Equation (28) must be true implies that q can be defined in terms of r as in Equation (29).

$$\sum_{i=1}^{N_{\text{pop}}} p_i = 1 \tag{28}$$

$$q = \frac{r(N_{\text{pop}} - 1)}{2} + \frac{1}{N_{\text{pop}}} \tag{29}$$

If $r = 0$ then $q = 1/N_{\text{pop}}$ and there is no selection pressure. Conversely, if $q - (N_{\text{pop}} - 1)r = 0$ then $r = 2/(N_{\text{pop}}(N_{\text{pop}} - 1))$ and $q = 2/N_{\text{pop}}$, providing the maximum selection pressure. For this thesis application, I chose to use the maximum selection pressure, creating the probability of selection curve in Figure 14 when $N_{\text{pop}} = 500$. An subeset of the selection probabilities are enumerated in Table 3.

Table 3. Probability of selection by chromosome fitness rank with $N_{\mathbf{pop}} = 500$

Rank	Probability of Selection
1	0.00400
2	0.00399
3	0.00398
⋮	⋮
499	0.00000802
500	0

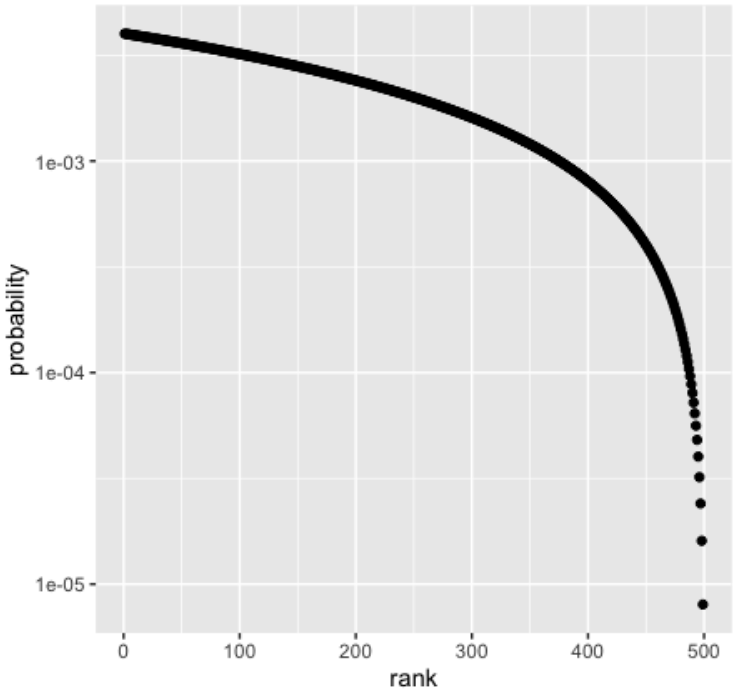


Figure 14. Log probability of selection by chromosome fitness rank with $N_{\mathbf{pop}} = 500$.

The output of the selection process is a selection vector of chromosomes to potentially be mated.

3.5.3.4 Crossover.

A single point crossover method is used in this thesis. When two chromosomes are chosen to be mated, a uniform random point between two genes is generated. Children are created by exchanging all of the genes after the randomly generated point from each parent to the other. The single point crossover method is illustrated in Figure 15.

$$
\begin{array}{lc}
\text{Parent 1} & \left[\begin{array}{cccc:cccccc} A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 & A_{10} \end{array}\right] \\
\text{Parent 2} & \left[\begin{array}{cccc:cccccc} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_8 & B_9 & B_{10} \end{array}\right] \\
 & \text{Crossover Point} \\
\text{Child 1} & \left[\begin{array}{cccccccccc} A_1 & A_2 & A_3 & A_4 & B_5 & B_6 & B_7 & B_8 & B_9 & B_{10} \end{array}\right] \\
\text{Child 2} & \left[\begin{array}{cccccccccc} B_1 & B_2 & B_3 & B_4 & A_5 & A_6 & A_7 & A_8 & A_9 & A_{10} \end{array}\right]
\end{array}
$$

Figure 15. Example child creation using the single point crossover method

A mating matrix of size $N_{\text{pop}}/2 \times 2$ is created and the index numbers 1 through N_{pop} are each assigned to a random position in the matrix. Each row of the matrix represents a pairing of two chromosomes to potentially be mated. The numbers in the matrix correspond to an index in the selection vector. Next, a uniform random number is compared to the mating probability, $p_{\text{crossover}}$, to determine if a row successfully mates. Figure 16 shows a mating matrix for a population with ten members.

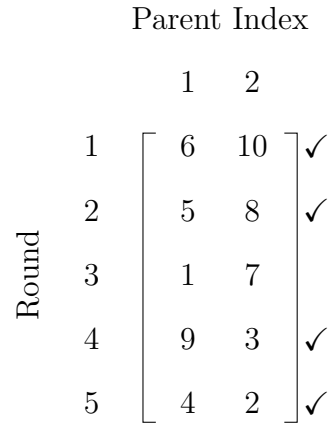


Figure 16. Mating matrix with check marks next to pairs chosen to successfully mate

Figure 17 shows the mating matrix on the left and the crossover overview on the right of an instance with a population size of ten. In the mating matrix rows 1, 2, 4, and 5 are check marked as successfully mating pairs. The chromosomes in the indexes of the selection vector corresponding to a mating pair are then combined to create offspring.

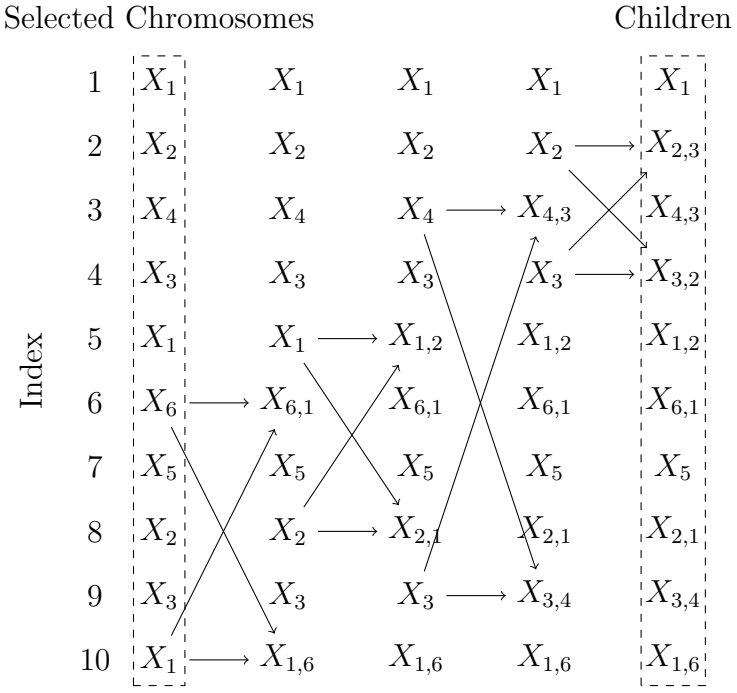


Figure 17. Crossover of index pairs 6-10, 5-8, 3-9, and 2-4

3.5.3.5 Mutation.

After selection and crossover, every chromosome has an opportunity for mutation. A uniform random number is compared to the probability of mutation, p_{mutate}, to determine if a chromosome undergos mutation. If selected for mutation, a random uniform index value is generated and the corresponding gene in that index position takes on the value of the opposite allele. In a binary system, gene mutation is accomplished using Equation (30).

$$\text{gene}_{\text{mutated}} = |\text{gene}_{\text{value}} - 1| \tag{30}$$

3.5.3.6 Elitism.

Before selection, crossover, and mutation occur, a copy of the population is created to propagate elitism. The copied population is sorted by fitness and all duplicate chromosomes, chromosomes with identical genetic makeups, are filtered out. Finally, some number of elite members are chosen, typically a proportion of the population, n_{elite}. Five percent of the population is used here for elitism.

After selection, crossover, and mutation are finished, the fitness of the new population is calculated. The new population is then sorted by fitness value. The lowest $N_{\text{pop}} * n_{\text{elite}}$ ranking members are then replaced with the elite members from the previous generation.

3.5.3.7 Fitness Functions.

The goal of the fitness function is to reward a desired behavior among the population and penalize undesired behaviors. I use the same notation from Section 3.5.1 and Section 3.5.2 where aircraft are indexed over i and sensor sites are indexed over j. In this thesis the two behaviors that are available for analysis are the number

of sensors in a solution, $\sum_j x_j$ and the number of undetected aircraft in a solution $[N_a - \sum_i \max_j x_j d_{ij}]$, where N_a is the total number of aircraft in the scenario. To get the fitness, I multiply number of sensors and undetected aircraft by some weight and sum the values.

For the minimal set covering problem, the objective is to detect all aircraft while using as few sensors as possible. If all aircraft are detected, the undetected aircraft term is zero and the weight on the number of sensors is arbitrary so long as it is strictly positive. I can set the sensor weight to one so the scaling for the undetected aircraft weight is uncomplicated. Since missing even a single aircraft is worse than using every sensor in this problem, the weight applied to undetected aircraft must be at least as large as the total number of sensors available, N_s. Thus, the GA minimized the fitness function for the minimal set covering problem, shown in Equation (31).

$$f(\text{chromosome}) = \left(\sum_{j \in J} x_j \right) + N_s \left(N_a - \sum_{i \in I} \max_{j \in J} x_j d_{ij} \right) \tag{31}$$

For the maximal set covering problem, the objective is to detect as many aircraft as possible while using a fixed number of sensors, M_s. If the correct number of sensors is used, the sensor term goes to zero and the weight on the number of undetected aircraft is arbitrary so long as it is strictly positive. I can set the undetected aircraft weight to one so the scaling for the sensor count weight is uncomplicated. To encourage the algorithm to move towards the correct number of sensors, the weight should reflect the distance from the desired value and scaled for the number of aircraft in the scenario. Thus, the GA maximized the fitness function for the maximal set covering problem, shown in Equation (32).

$$f(\text{chromosome}) = \left(\sum_{i \in I} \max_{j \in J} x_j d_{ij} \right) - N_a \left| M_s - \sum_{j \in J} x_j \right| \tag{32}$$

3.5.4 Time-Limited Deterministic Approach.

The numeric method described in Section 3.5.2 is unable to determine an optimal solution to larger instances of the problem in a tractable amount of time but it can be halted early and the best-so-far solution can be extracted. Thus, several of the problem instances, which are too large to be solved to optimality, were given up to an hour of computer time using the method of Section 3.5.2 and the best solutions were extracted and compared to the solutions produced by the genetic algorithm.

3.5.5 Mimetic Pattern Approach.

The results of all other approaches failed to yield any discernible pattern to the optimal solutions. In this absence, the mimetic patterns used to seed the genetic algorithms were input into the scenario and evaluated for fitness. Using the idea of a narrow 'fence' as described in Section 3.3.1, many of these patterns did not include more than 2 lines of latitude and were simplified to suggest reproduce-ability in other geographic regions. The first examined pattern produced a zigzag where each consecutive longitudinal index contained only one entry which did not match that of an adjacent longitudinal row. The second pattern allowed all sensors along a single line of longitude and removed sensors uniformly until a sensor limit was reached. Figure 18 illustrates some of the patterns selected for analysis.

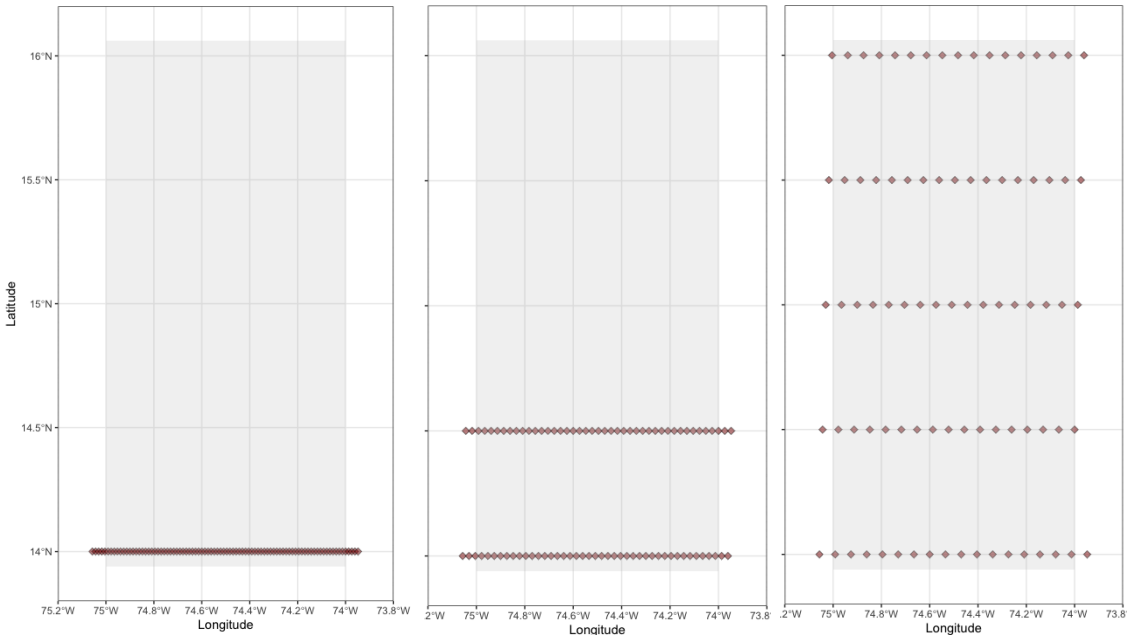


Figure 18. Sample of mimetic patterns input into the genetic algorithm where each pattern contains an equal quantity of sensors

IV. Analysis

4.1 Overview

The goal of this chapter is to provide a framework for deploying a GPS-based multistatic radar architecture. This goal is achieved by validating the data produced by the physics-based model, calculating known optimal solutions to subsets of the data, applying deterministic and heuristic search strategies to find optimal, or near optimal, solutions to the full data set, and testing extrapolation strategies. The extrapolation strategies empirically demonstrate that simple geometric patterns can achieve greater than 98 percent coverage without the need for thousands of hours of simulation, data generation, and optimization.

4.2 Data

This section describes the steps taken to ensure high quality data being fed into the models and algorithms. In an effort to minimize analysis time while maximizing thoroughness, a combination of detailed examinations and broad approaches were used to assess the validity of the data. A full evaluation of a subset of the data, an ocular analysis of the entire detection matrix, and a data correlation analysis were performed.

4.2.1 Data Generation.

The data was created by splitting the overall scenario into 256 instances, each responsible for generating the sensor-aircraft relationships for five sensors at the same longitude. Using a combination of shell scripting, bash commands, connect commands, python, and STK, each instance of the model simulated all GPS satellites in the STK almanac, five sensors located at a single longitude, and all aircraft that are

in the field of view of those sensors. Each instance then created an output log for every sensor-satellite-aircraft combination. Every output log was immediately read, aggregated into a single data file, and deleted. The 256 individual data files were then ingested into R and processed to create a full data object containing all aircraft detections in a single binary matrix. Each of the 256 data files required approximately ten hours to create. Unfortunately, STK does not have additional ports to allow simultaneous communication with multiple scenarios. For this reason, each instance had to be created using a different CPU or in serial on a single CPU.

The original scope of this thesis covered ten degrees longitude. However, generating such a large amount of data was contingent on having an estimated sixty thousand hours of computer time available on the Air Force Research Lab (AFRL) High Power Computer (HPC). A facility anomaly left the required assets unavailable throughout the entirety of the time scheduled for data generation.

Fortunately, although greatly limited in scope, a concurrent effort to create the data was carried out by accessing sixty computers on the Air Force Institute of Technology (AFIT) intranet and running the requisite programs on a background profile. In total, two months were required to generate enough data to analyse a one degree longitude by two degree latitude area of interest. One assumption, given the repetitive satellite motion of the GPS constellation, is that the solutions provided by this thesis could be scaled to a larger region at the same latitude. A scaled version would provide only an idea of how many sensors would be required, but could not predict where those sensors should be placed. The pre-selected patterns described in Section 3.4.5 may be scaled up, at least at 14 degrees North, and provide an approximate coverage capability of a patterned architecture.

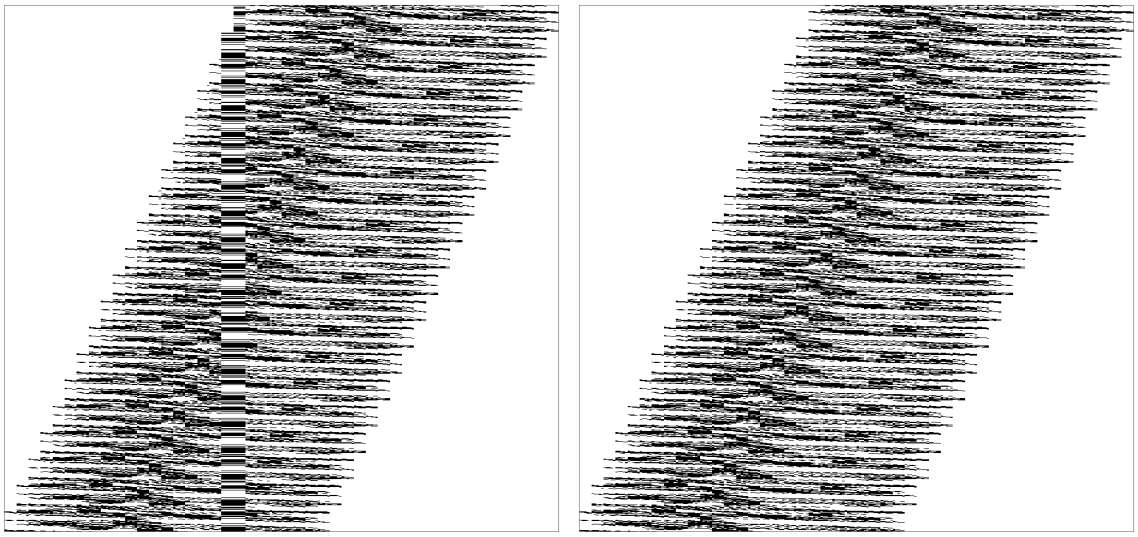
4.2.2 Random Sampling.

A randomly sampled subset of data files were examined in detail and the accuracy of every data point was validated against the model. In total, 10 of the 256 files were exhaustively compared to the STK model for each aircraft detection identified. No discrepancies were found during this phase of data validation.

4.2.3 Detection Matrix Visualization.

The entire detection matrix was visualized and reviewed for unexpected behavior. The close spacing of the sensors and the relatively slow motion of the satellites produces a pseudo repeating pattern within the detection matrix. There is very little difference between each individual set of five sensors, so scanning visually through the data provided an opportunity to detect anomalies.

Two such anomalies were found when reviewing the data. Figure 19a shows a portion of the detection matrix with twenty consecutive files worth of data. There is an obvious deviation from the repeating nature of the data. This unexpected behavior was further analysed and is discussed in Section 4.2.4. After regenerating the errant data files, the data visualization, shown in Figure 19b, no longer displayed any apparent deviations.



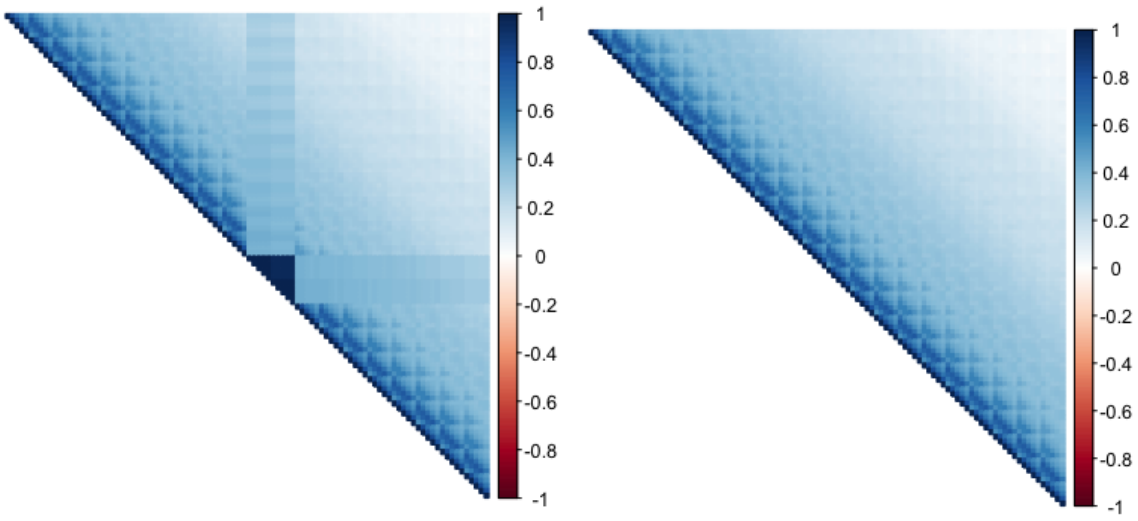
(a) Two Corrupted Files (b) No Corrupted Files

Figure 19. Visual representation of the combined detection matrix from twenty data files with aircraft on the vertical axis and sensor locations on horizontal axis. Each grid element pertaining to an aircraft detected by a particular sensor is filled in with black.

The exhaustive and occular data validation steps were improved upon in Section 4.2.3 using the statistical measurement of multi-collinearity.

4.2.4 Multi-collinearity.

A correlation between the detections made by adjacent sensors is expected, but the lengths taken to ensure the minimum required overlap of sensors and aircraft guarantee that no two sensors should exhibit the same behavior on the system. For this reason, I can measure the correlation among all of the sensors and quickly look for outliers such as sensors that are too highly correlated, or sensors that share any information with disparate sensors too far away to possibly detect any of the same aircraft.



(a) Two Corrupted Files (b) No Corrupted Files

Figure 20. Sensor observations correlation matrix of twenty data files

Figure 20a is a visualization of correlation values between each sensor location and the aircraft detectable at that location. There is a five-column repeating pattern due to the index grouping of five sensors per longitude. There is an obvious set of outliers in this figure. This technique revealed that four of the data files had different correlation profiles from the rest. Further investigating the meta-data of the raw data files uncovered discrepancies in the sizes of those files as well. All of the corrupt files came from the same computer and were rerun on another machine. The recreated data was used to produce the correlation matrix of Figure 20b.

4.3 Minimal Set Covering Problem Exact Results

The minimal set covering problem answers the question: What is the minimum number of sensors required to detect every target in the system? The mathematical model developed in Section 3.4.1 was used to determine the solution to the minimal set covering problem on tractable problem sizes and estimate the time for a deterministic solution to the full problem size.

4.3.1 Solutions to the Minimal Set Covering Problem.

IBM's CPLEX OPLIDE utilized a branch-and-bound approach to solve the problem instances. The optimization software was able to solve to optimality problems with fewer than 28 adjacent flight paths. Figure 21 is a plot of the minimum number of sensors required for total coverage given the number of adjacent flight paths in the scenario along with a linear regression line shown in blue.

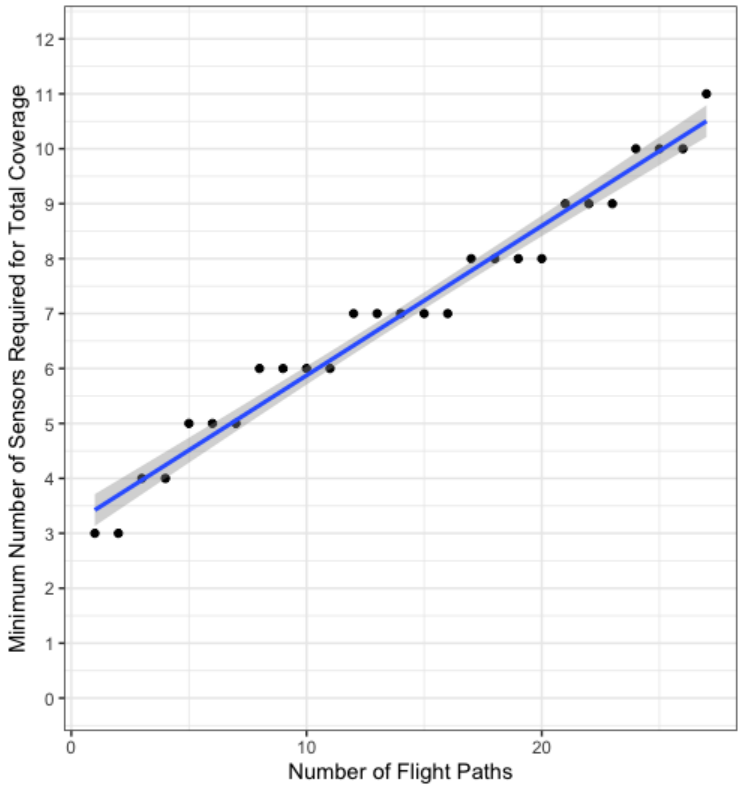


Figure 21. Minimum number of sensors required for total coverage vs number of adjacent flight paths with first order regression in blue.

The results from the first 27 flight paths are used to estimate the minimum number of sensors, S_{Min} that are required for total coverage in the one degree longitude scenario with 230 adjacent flight paths, AC_n. The linear model is described in Equation (33). By solving Equation (33) with $AC_n = 230$ I estimate that at least 66 sensors are required for total coverage of a one degree longitude area of interest.

$$S_{\text{Min}} = 3.15 + 0.27\ AC_n \tag{33}$$

Figure 22 illustrates one potential method for identifying alternative optimal solutions. An instance with only a single flight path, plotted as the solid vertical line in the center of the figure, was considered and multiple optimal sensor placements were found. To create these solutions, two of the three optimal sensor locations are held constant, shown as solid black points, and 28 unique potential locations for the third sensor were discovered, shown as light grey points.

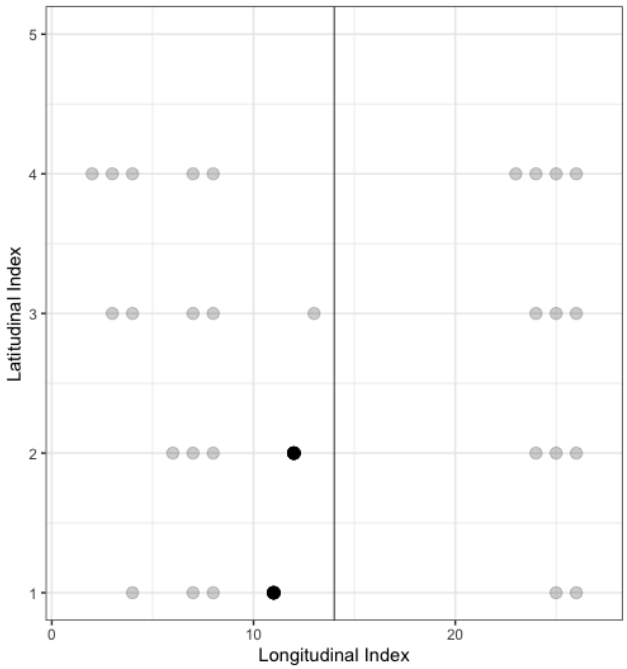


Figure 22. Eighteen unique optimal solutions for a single flight path, plotted as a black line. Each solution shares two sensor locations, marked as solid black points, and has one unique sensor location, marked as grey points.

Next I consider wholly unique optimal solutions to the minimal set covering problem with only one flight path. Figure 23 plots the six completely unique solutions, meaning no sensor site is used in more than one solution. These solutions were found by constraining the model to force previously used sensor locations to zero in con-

secutive instances. With the 18 total sensor sites from these six solutions removed from consideration, there are no remaining subsets of only three sensor sites that are capable of detecting every aircraft on the flight path.

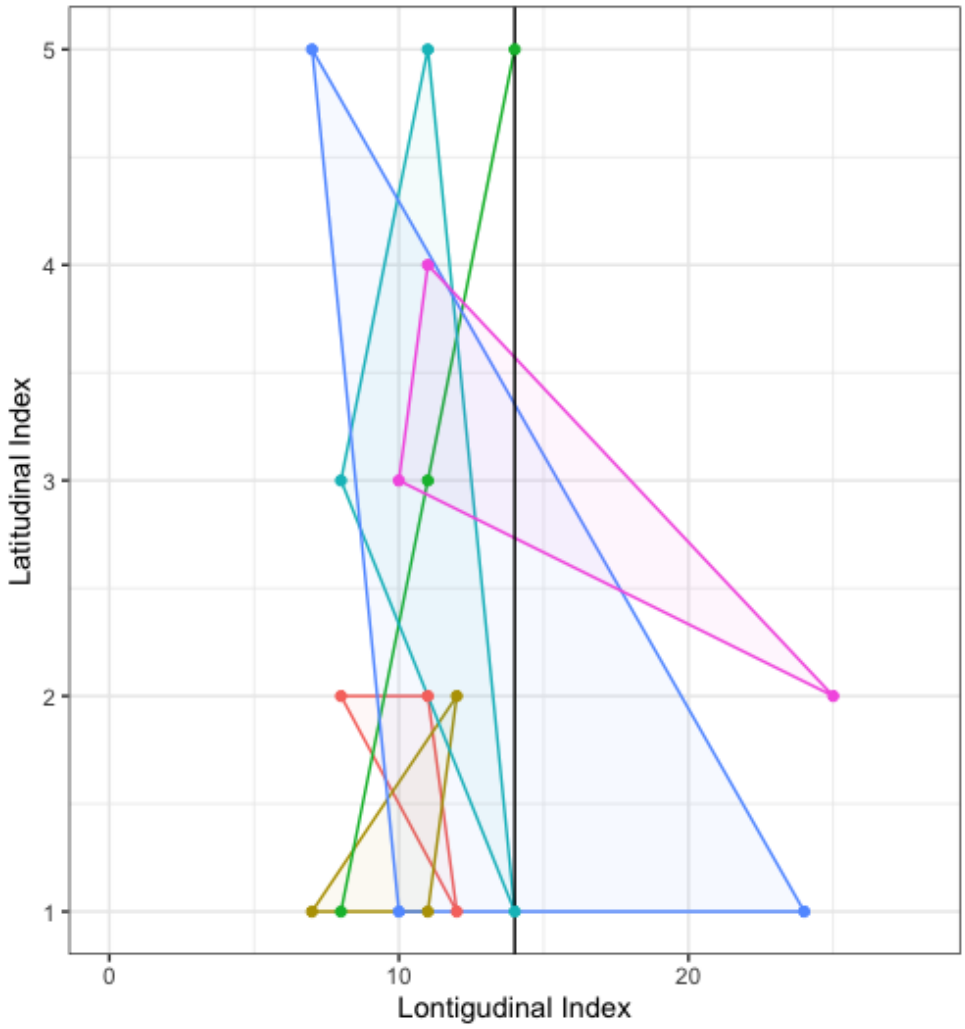


Figure 23. Unique optimal solutions for a single flight path, plotted as a black line, each marked by a colored triangle with vertices on the optimal sensor locations, none of which are shared by any two solutions.

No discernible pattern has emerged from the analysis of many variations of optimal solutions. Repeating this technique on a problem with 20 adjacent flight paths, shown in Figure 24 also revealed no clear pattern. In the case of 20 flight paths, there was only one optimal solution of eight sensors. When those eight sensor locations are

constrained to zero, nine other sensor locations are required to guarantee complete coverage.

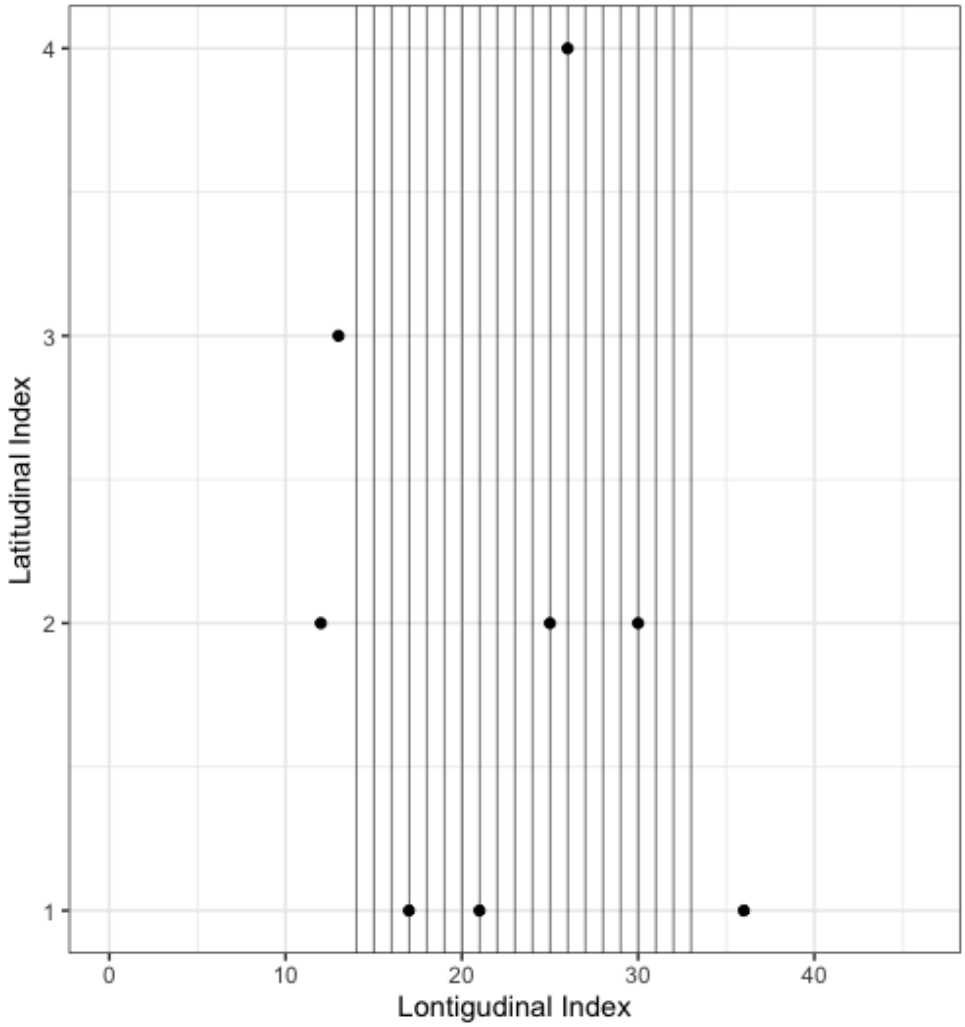


Figure 24. Optimal solution for 20 adjacent flight paths, plotted as black lines, with sensor locations marked as solid black points.

4.3.2 Temporal and Computational Constraints on Tractability.

The intractability of a deterministic approach for the full size problem was empirically demonstrated by extrapolating from the time required to generate each of the tractable problem sizes. Figure 25 plots the time required to compute the optimal solution to the minimum set covering problem for instance sized from one to twenty-

five flight paths. The time axis is logarithmic and shows the exponential growth in computation time with a linear growth in problem size.

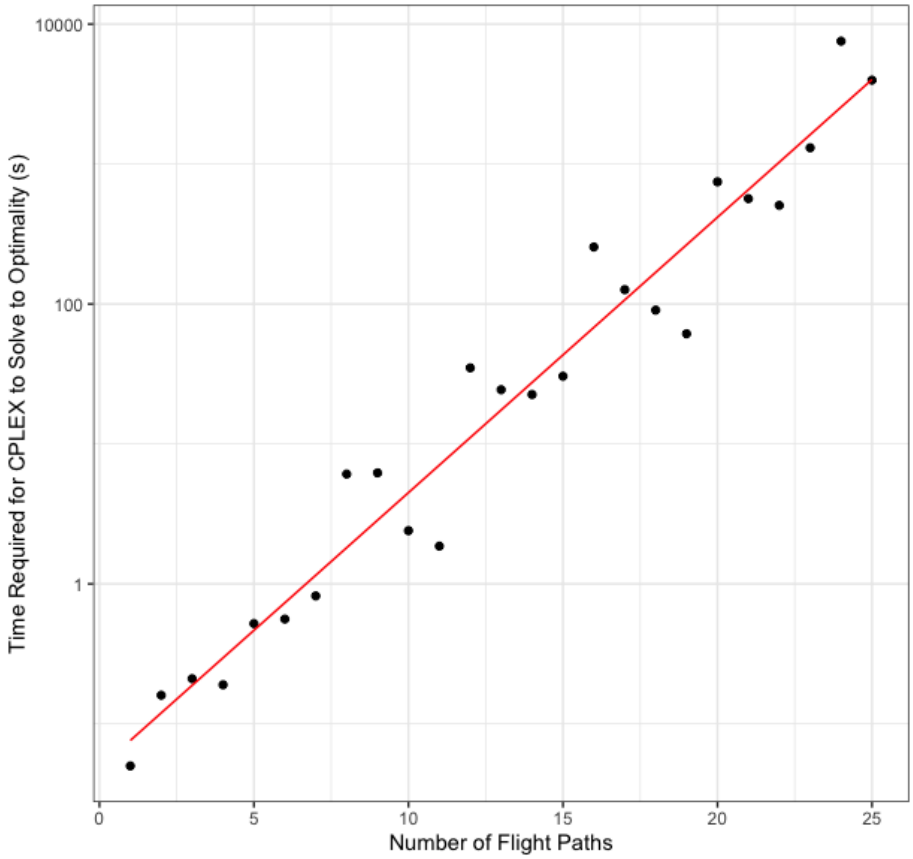


Figure 25. Plot of time required to compute optimal solution to minimum set covering problem for instance sizes from one to twenty-five flight paths. Time axis is logarithmic and best fit line is drawn in red.

The regression equation for the best fit line in Figure 25 is shown in Equation (34) where time is measured in seconds. For the full problem size with 230 flight paths this trend suggests that 8.80e43 seconds would be required. It is important to note at this point that the age of the universe is estimated to be around 4.35e17 seconds.

$$\text{time} = e^{-3.0302+0.4531(\text{Number of Flight Paths})} \tag{34}$$

4.4 Maximal Set Covering Problem Exact Results

The maximal set covering problem answers the question: Given a fixed number of sensors, what is the maximum number of targets that can be detected? The mathematical model developed in Section 3.4.2 was used to calculate solutions to the maximal set covering problem. For each of the tractable instances to which a solution to the minimal set covering problem was solved, a series of maximal set covering problems were solved in an effort to characterize the solution space. Starting with one sensor available, each instance was solved to optimality, then another sensor was added and the problem was solved again. I continued this process until the number of sensors in the problem equalled the minimum number of sensors required for total coverage.

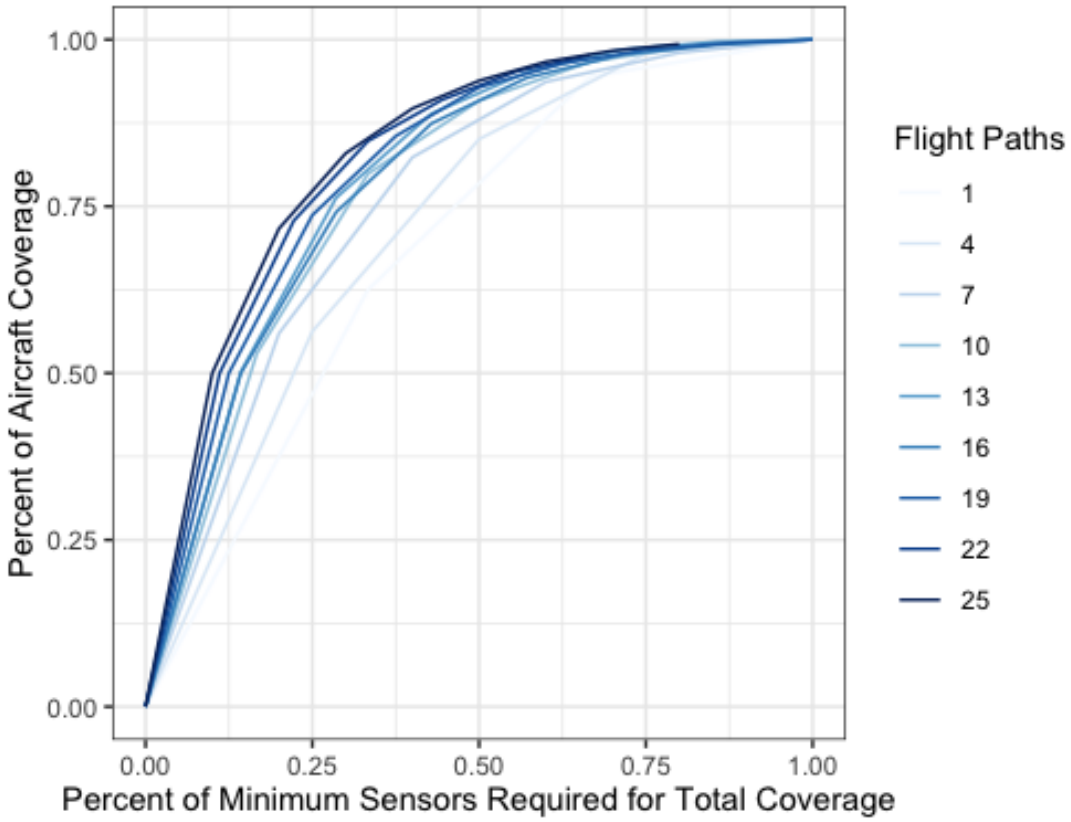


Figure 26. Percent of targets detected vs percent of minimal number of sensors required for complete coverage.

Figure 26 is a plot of the the percentage of targets that are detected using some percentage of the minimum number of sensors required for complete coverage. For tractable instances, the percentage of the minimum number of sensors required for complete coverage was found using the solutions found with the minimal set covering problem. The optimal solutions are enumerated in Table 4.

Table 4. Minimum number of undetected targets given fixed number of sensors, missing values correspond to intractable problem sizes of the maximum set covering problem. The zeros were all determined by the minimum set covering problem.

	Number of Undetected Aircraft Given Fixed Number of Sensors										
Flight Paths	1	2	3	4	5	6	7	8	9	10	Total Aircraft
1	27	4	0								72
2	59	15	0								144
3	92	28	4	0							216
4	126	43	6	0							288
5	159	60	13	1	0						360
6	191	74	23	4	0						432
7	222	89	32	8	0						504
8	258	107	44	13	1	0					576
9	297	142	56	18	2	0					648
10	339	145	66	24	5	0					720
11	380	169	76	30	6	0					792
12	423	198	88	38	11	1	0				864
13	466	222	104	46	14	1	0				936
14	511	246	117	52	19	5	0				1008
15	556	272	129	57	24	6	0				1080
16	596	301	145	66	26	9	0				1152
17	637	328	161	77	31	12	1	0			1224
18	678	351	178	85	38	17	2	0			1296
19	722	373	198	97	45	21	5	0			1368
20	766	402	215	108	54	24	8	0			1440
21	809	433	235	120	66	32	12	2	0		1512
22	851	468	259	138	76	39	17	6	0		1584
23	898	504	286	161	86	47	21		0		1656
24	951	541	315	182	103	54	26			0	1727
25	1005	584	348	202	116	64	30			0	1800
26	1066	631	382	225	132	74	39			0	1872
230	15741	14922	14105	13288	12471	11654	10839	10030			16560

An exponential function was fit to the data with a R-squared value of 0.999. The model was based on optimality information from the subset of data with 26 flight paths. The fit line is created using Equation 36 where low = 0, high = 1, and ρ ≈ -0.194. The value for ρ was determined using Equation 35 and actual data from the 26 flight paths instance of the maximal set covering problem using undetected aircraft = 132, total aircraft = 1872, and z = (10-5)/5 = 0.5.

$$\frac{\text{undetected aircraft}}{\text{total aircraft}} = \frac{1 - e^{-z(\text{undetected aircraft/total aircraft})/\rho}}{1 - e^{-1/\rho}} \tag{35}$$

$$y = 1 - \frac{1 - e^{-(\text{high}-x)/\rho}}{1 - e^{-(\text{high}-\text{low})/\rho}} \tag{36}$$

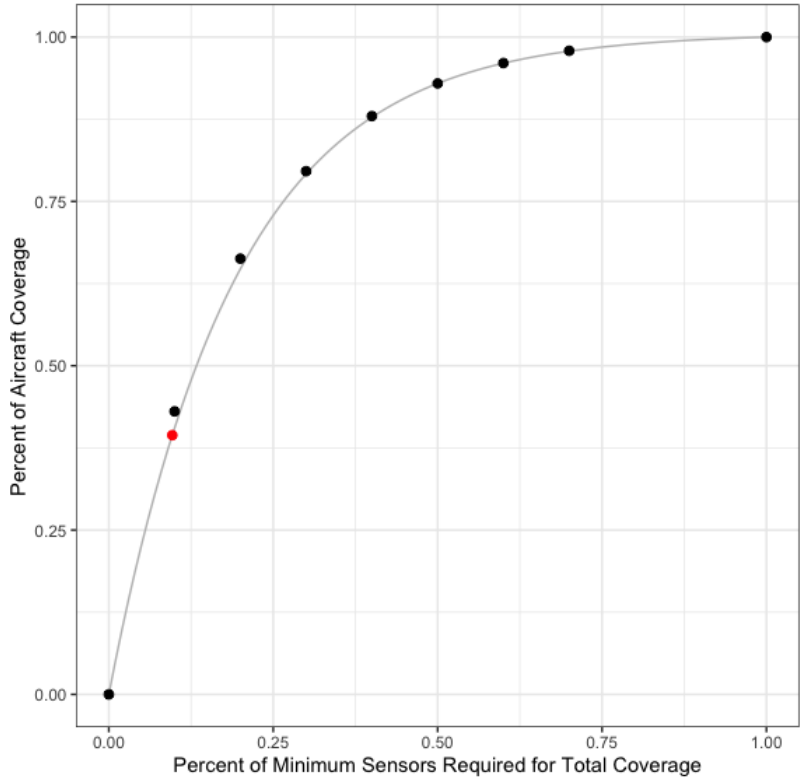


Figure 27. Exponential fit of maximal set covering data shown, as a solid line, compared against actual data from 26 flight paths, shown as black points. The red point represents the actual data from 8 sensors on the full scale problem and is used to estimate the number of sensors required for complete coverage.

The OPLIDE was able to generate some optimal solutions to the maximal set covering problem on the entire data set. I started with one sensor, then added more until the problem was no longer tractable with the software. This approach gave us optimal answers for up to eight sensors. I employed a numerical approximation technique to determine the value of x that would produce the y values I was able to determine with the OPLIDE. Having already calculated that, I can detect 6,530 of 16,560 targets in a one degree longitudinal space I used Equation 36 to estimate the required number of sensors in the full scale problem.

Solving Equation 36 for 8 sensors in the full scale problem gives an estimate that this is 9.65 percent of the sensors required for complete coverage. Thus an estimated $8/0.0965 = 83$ sensors are required to achieve total coverage using the maximal set covering problem solution space analysis.

This differs from the estimation given by the minimal set covering problem solutions, that at least 66 sensors are required for complete coverage. With these two approximations, I expect the meta-heuristic to produce a solution to the minimal set covering problem within the range of 66 to 83 sensors.

4.5 Minimal Set Covering Problem Heuristic Results

The meta-heuristic for the minimal set covering problem was validated against the known solutions, using the exact results from Section 4.3. The meta-heuristic was executed ten times for each instance size from 1 to 26 adjacent flight paths. The results are plotted in Figure 28, with the results from the meta-heuristic shown as a green box-and-whisker plot and the known optimal solutions as red dots. Each instance used a population of 480 chromosomes with 24 elite members, 250 iterations, an 80% crossover rate and a 20% chance of mutation.

For the full problem size, ten instances of the meta-heuristic search algorithm were

executed. The instances had a population of 48 chromosomes and 3 elite members. The best solution found by the genetic algorithm requires only 84 sensors to guarantee complete coverage. The time-limited optimization function was halted after ten minutes and found a solution requiring only 72 sensors.

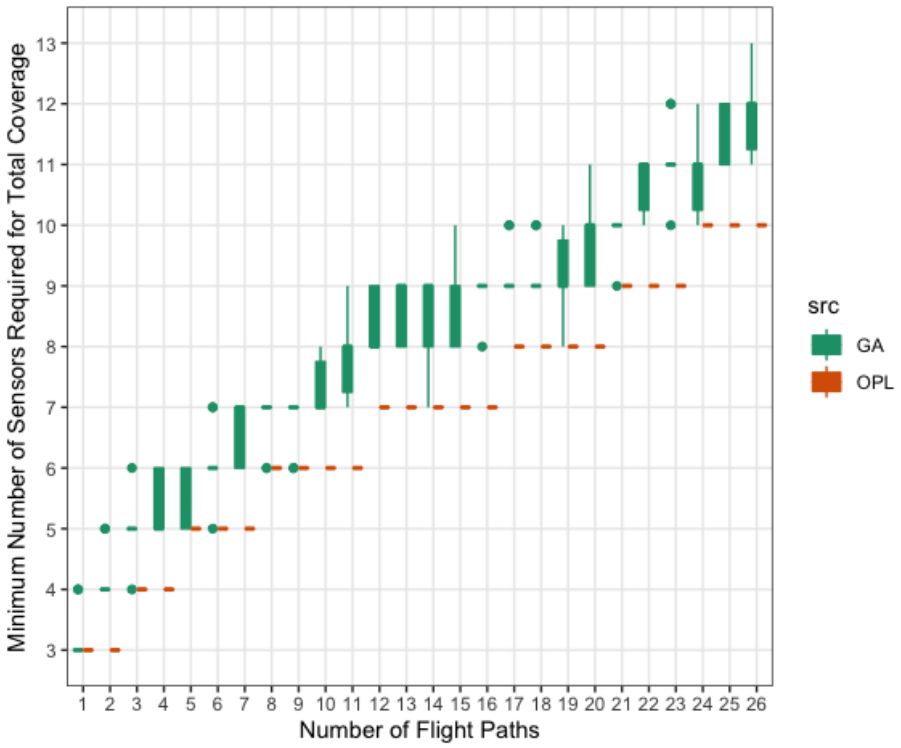


Figure 28. Ten genetic algorithm sample validation results per number of flight paths compared to known optimal solutions. In many cases the genetic algorithm is able to produce an optimal solution, given enough samples. In this plot I am trying to determine the minimum number of sensors required to detect all aircraft in each instance.

4.6 Maximal Set Covering Problem Heuristic Results

4.6.1 Maximal Set Covering Problem Genetic Algorithm Results.

Ten instances of the maximal set covering problem genetic algorithm were run for each scenario with a fixed number of sensors. The scenarios ranged from 2 to 8 sensors for validation and for 9 to 72 sensors for implementation. Each scenario with 9 to 71 sensors was also solved with the time-limited optimization function. The

results are plotted in Figure 29.

4.6.2 Maximal Set Covering Problem Pre-selected Pattern Results.

The best performing pre-selected pattern involved evenly spacing available sensors along a single line of latitude. Knowing that 72 sensors were capable of detecting all aircraft in the full scenario, the quality of each sensors ability to detect aircraft was examined. As expected, the sensors on the periphery of the area of interest tapered off in performance and there was a near uniform quality among sensors not near a longitudinal boundary. Thus, the 20 worst performing rows of sensor locations from both longitudinal bounds were removed and the remaining 216 rows of sensor locations were allocated sensors per the identified pattern. Up to 72 sensors, quantities were evaluated so that the sensors were evenly divided among the potential locations, meaning only solutions with multiples of prime factors of 216 were used. i.e., 12, 18, 24, 27, 36, 54, and 72.

Figure 29 plots the best results from each scenario. Similar to Figure 26, the y-axis is the percentage of aircraft detected and the x-axis is the number of sensors used in the scenario. The green points are known optimal solutions. The red points are solutions found by the time-limited optimization function. The grey points are the solutions found by the genetic algorithm. Finally, the blue points are the solutions from using a repeating geometric pattern on a single line of longitude. In the instance with 72 sensors, the repeating pattern simply meant placing a single sensor every 1,407 meters along a single line of latitude, resulting in over 98 percent of the aircraft being detected.

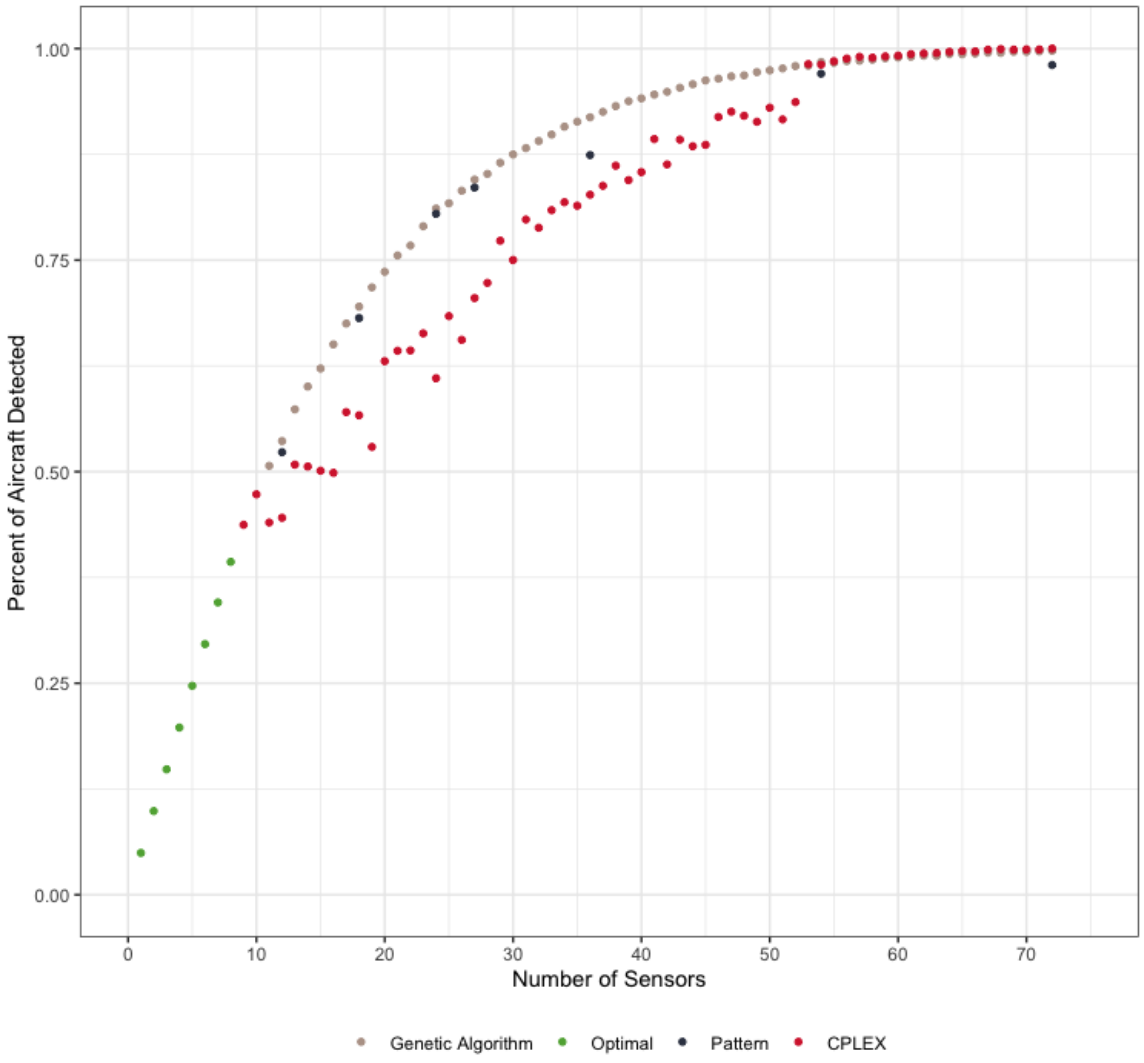


Figure 29. Percent of aircraft detected versus number of sensors used comparing results from the genetic algorithm, time-limited branch and bound, known optimal solutions, and sensors placed in geometric patterns. In this plot I am trying to determine the maximum number of aircraft that can be detected given a fixed number of sensors.

V. Conclusion and Future Research

5.1 Conclusion

The original intent of this thesis was to prove the feasibility of modeling the forward-scatter, GPS-based, transmitter/target/receiver problem over the geographic area of interest as a set covering problem, then develop a meta-heuristic capable of solving for a near optimal solution which maximizes target detection while minimizing costs. The answer to each of the research questions posed in this thesis is *yes*: It is possible to model the situation as a set covering problem and a meta-heuristic is capable of determining a near optimal solution.

The forward-scatter radar properties of a GPS-based architecture was shown to have a very limited coverage for each individual satellite; however, a large aggregate coverage is created when considering the entire GPS constellation. At the relatively low latitudes considered, 14 to 16 degrees North, there were nine satellites on average at any given time which could be used by the system.

The minimal set covering problem meta-heuristic developed for this thesis proved that any aircraft flying due north between 75 and 74 degrees West and 14 and 16 degrees North can be detected at least once given an architecture of at most 84 GPS receivers. The locations for the sensors are plotted in Figure 30. The time-limited optimization function found a solution that detects all aircraft but only requires 72 sensors. The solution requiring only 72 sensors is plotted in Figure 31

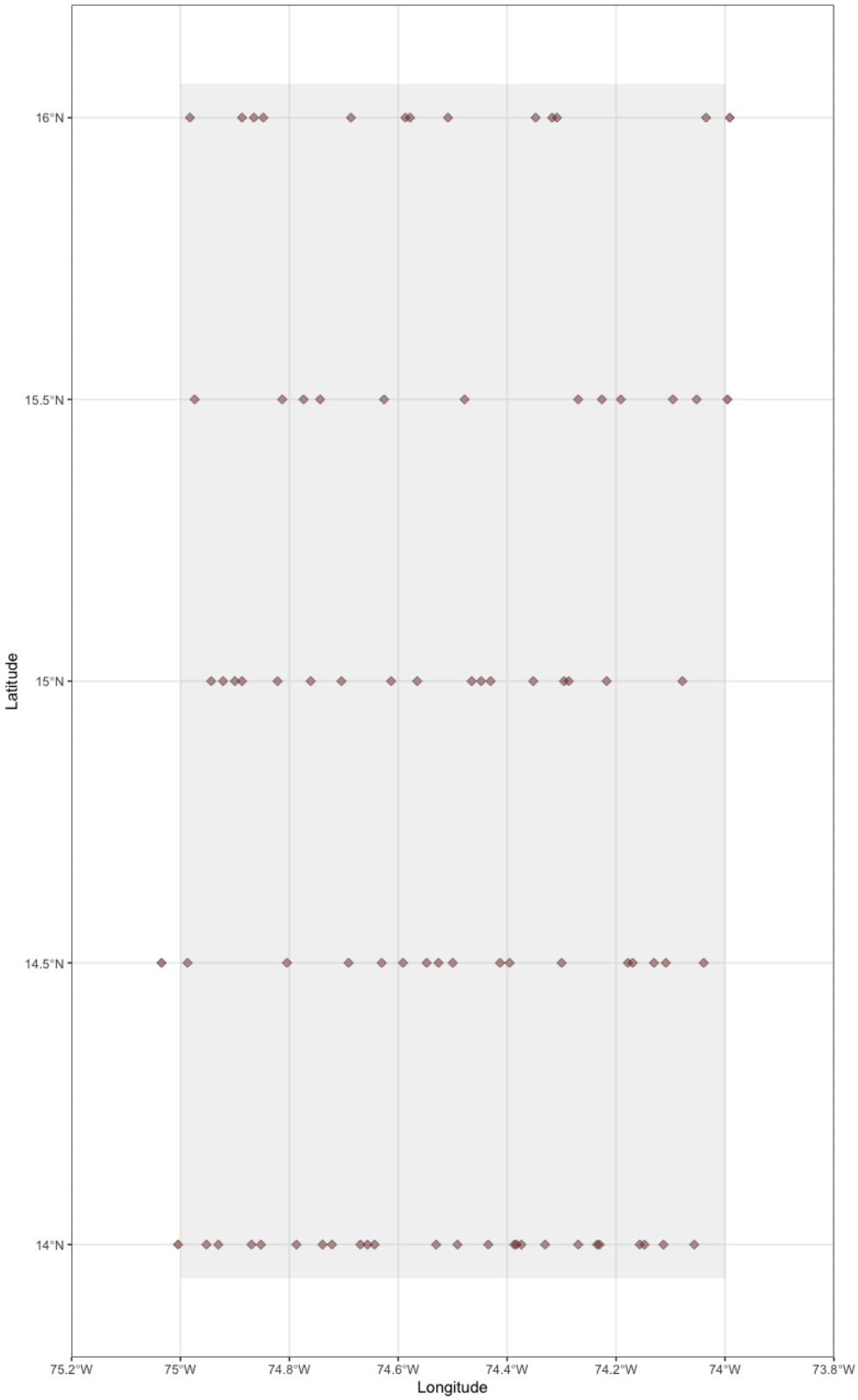


Figure 30. Best solution (84 sensors) found by the genetic algorithm

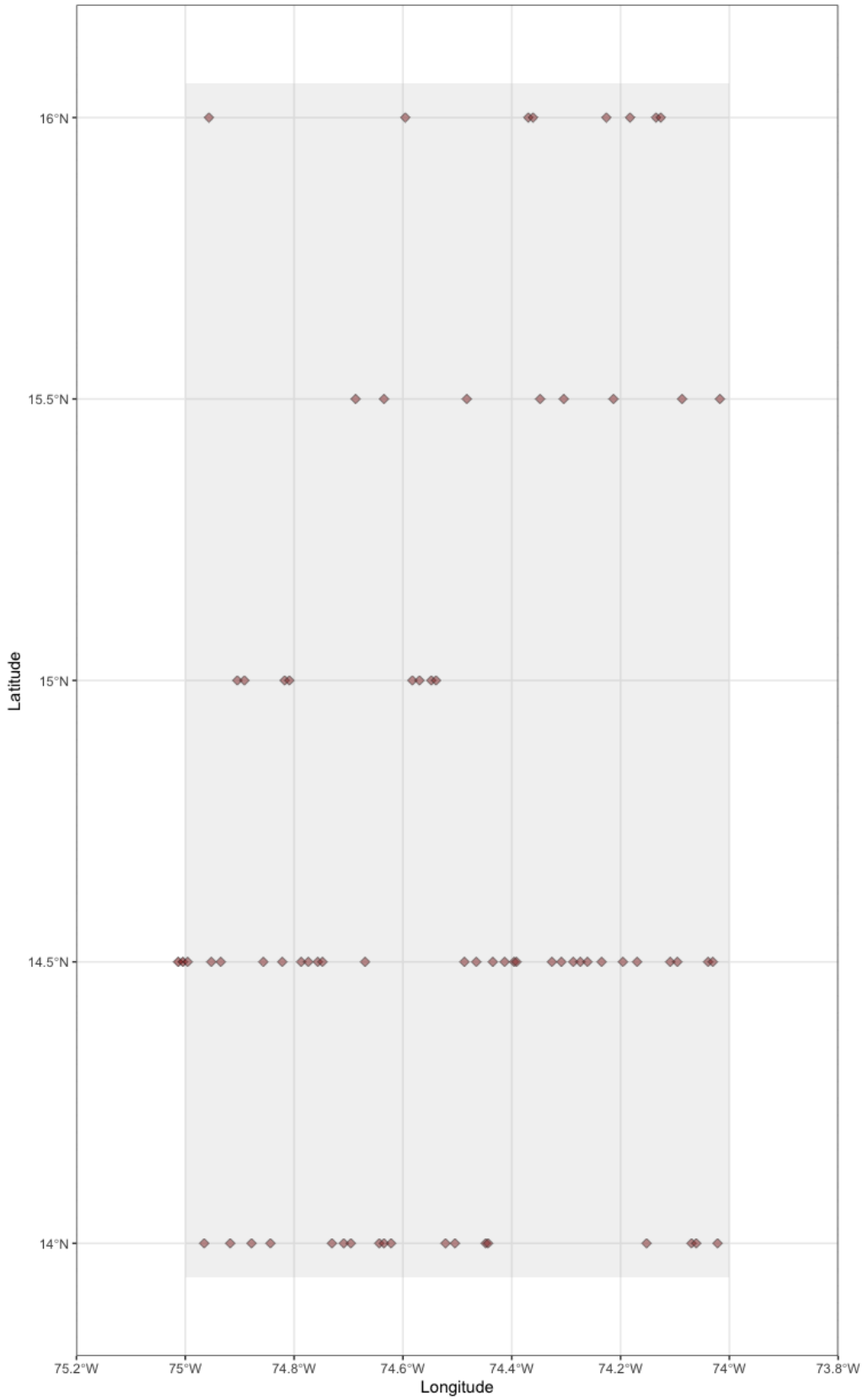


Figure 31. Best solution (72 sensors) found by the deterministic optimization algorithm

The Pareto frontier of the maximal set covering problem can be found as the best performing point per number of sensors plotted in Figure 29. Each of these points represents the known maximum number of aircraft that can be detected given a fixed number of sensors. The genetic algorithm produced better results between 11 and 56 sensors and the time-limited optimization function yielded better results between 57 and 72 sensors, each given about the same amount of time to run.

The shape of this frontier illustrates a trade-off between system cost and coverage. Each successive sensor added to the system provides a diminished return when compared to its predecessor. Some key points along this curve are earmarked in Table 5. Only 11 sensors are required to achieve 50 percent coverage of the area but and additional 10 sensors are required to add 25 percent more coverage. To increase from 99 to 100 percent coverage, an additional 18 sensors are required.

Table 5. Number of sensors required to achieve desired coverage

Coverage (%)	Sensors
50	11
75	21
90	32
95	40
99	54
100	72

The major discovery of this thesis is that simply placing sensors in a straight line can create a fence capable of detecting most of the aircraft that cross it. I proved that using 256 sensors per degree longitude at 14 degrees North yields 100 percent coverage, but using just 72 sensors spaced evenly along a straight line, without any data exploration, optimization, or tens of thousands of hours of computations,

reduces that coverage by only 2 percent. This patterned approach to setting up a forward-scatter radar fence should be repeatable in other locations that share a similar geo-spatial relationship with the GPS constellation. Also, because this patterned approach does not require multiple lines of latitude, the direction of travel of an aircraft does not affect the coverage. In fact, aircraft not moving due north or south spends more time in the detectable region, thereby increasing their chances of being detected.

5.2 Future Research

This thesis provides an optimal solution to provide coverage over a specific area given expected flight paths but stops short of describing a general solution for *any* geographic region and *any* flight patterns. The data created in this thesis can be used to explore some of the other facets of the problem space and the method developed to create the data may be reused on any area of interest.

Given the data already created and the optimal solutions presented, a user may determine coverage deficits which could be exploited by an adversary flying on a non-linear path. The raw data may also be used to develop new solutions with more robust coverage, by requiring that aircraft are detected more than once or simultaneously by multiple sensors.

Expanding on the data, a future researcher may optimize sensor placements in different areas of interest, especially over border crossings or no-fly zones using discrete sensor deployments. This low-bandwidth radar could also be integrated with an exquisite sensor in a tipping-and-queuing system.

Appendix A. Python Code to Create Scenario

```
while True:
    import datetime
    import sys
    import os
    #import glob
    import time
    timer_start = time.time()
    ####################
    # Instance Specific parameters
    ####################
    #inst_long = int(sys.argv[1]) # for execution
    homePath = 'L:\Research\ENS\\2020 STUDENT FOLDERS\MASTERS STUDENTS\OPER\Hufstetler, Brandon J. - Capt\WORKING FOLDER\HPC' # This is where reports are stored

    import re
    pattern = re.compile("[0-9]+")
    failures = 0
    restart = ""
    for i, line in enumerate(open(homePath + "\\Fails.txt")):
        for match in re.finditer(pattern, line):
            print("Failure number %s: %s" % (i+1, match.group()))
            failures += 1
            if failures == 1:
                start_inst = int(match.group())
                with open(homePath + "\\All_Access_" + "{0:0=4d}".format(start_inst) + ".txt") as f:
                    lines = f.read().splitlines()
                    last_line = lines[-1]
                    stop_inst = start_inst
```

```
                    restart_long = int(last_line[7])
                    restart_sat = int(last_line[11:13])
                    restart_aclong = int(last_line[17:21])
                    restart_time = int(last_line[22:24])
                    restart_aclong = restart_aclong - (start_inst -
    26)  #0
                    restart_time += 1
                    restart = "re"
     if failures == 0:
         i=0
         while i < 100:
             i += 1
             string = str(i) # + "\n"

             with open(homePath + "\\Starts.txt") as search:
                 exists = False
                 for line in search:
                     line = line.rstrip()  # remove '\n' at end of
    line
                     if string == line:
                         exists = True
                         break
                 if exists == False:
                     start_inst = i
                     stop_inst = i
                     restart_long = 1 #1
                     restart_sat = 1 #1
                     restart_aclong = 0 #0
                     restart_time = 1 #1
                     break

#             with open(homePath + "\\Starts.txt", "r") as f:
```

```
58 #                 if string in f.read():
59 #                     continue
60 #                 else:
61 #                     start_inst = i
62 #                     stop_inst = i
63 #                     restart_long = 1 #1
64 #                     restart_sat = 1 #1
65 #                     restart_aclong = 0 #0
66 #                     restart_time = 1 #1
67 #                     break
68     print("%sstarting F_%s_%s_S_%s_AC_%s_%s" % (restart, start_inst,
       restart_long, restart_sat, restart_aclong, restart_time))
69     #start_inst = 32
70     #stop_inst = 32
71     #restart_long = 1 #1
72     #restart_sat = 1 #1
73     #restart_aclong = 0 #0
74     #restart_time = 1 #1
75     #restart_aclong = restart_aclong - (start_inst - 26)  #0
76     #restart_time += 1
77     for inst_long in range(start_inst, stop_inst + 1):
78         timer_start = time.time()
79         longitude = -75.0 - (13/230.0) + (inst_long - 1) / 230.0 #
       deg
80         print("check0")
81         ####################
82         # Predefined parameters
83         ####################
84         homePath = 'L:\Research\ENS\\2020 STUDENT FOLDERS\MASTERS
       STUDENTS\OPER\Hufstetler, Brandon J. - Capt\WORKING FOLDER\HPC' #
        This is where reports are stored
85         #homePath = os.getcwd()
```

```
86
87          reportPath = homePath + "\\Reports"
88          #reportPath = homePath + "\\Reports_" + "{0:0=4d}".format(
    inst_long) #windows
89          #reportPath = homePath + "/Reports_" + "{0:0=4d}".format(
    inst_long) #linux
90
91          #if (inst_long != start_inst):
92          #os.mkdir(reportPath) # Make a directory for reports
93
94          sys.path.insert(0, homePath) # Point this to the folder
    containing the python functions module
95          sys.path.insert(0, '/p/home/bjhuf/run')
96
97          offset_time = 20*60 # seconds between aircraft launches
98          offset_long = 1/230.0 # meters between simultaneous aircraft
     (East to West)
99          offset_lat =  1/2.0 # meters between sites (North to South)
100         maxdist_horizontal = 13/230.0
101         ac_lat_start = '13.942' # deg
102         ac_lat_stop = '16.058' # deg
103         ac_altitude = '3048' # meters == 10,000 ft
104         ac_speed = '84' # 84 m/s == 188 mph
105         fac_lat_start = 14.0 # deg
106         fac_altitude = '0.0' # sea level
107         time_start = datetime.datetime(2019, 1, 1, 10, 0, 0) #
    Scenario start time (yyyy, mm, dd, hh, mm, ss)
108         time_end = datetime.datetime(2019, 1, 2, 10, 0, 0) #
    Scenario end time (yyyy, mm, dd, hh, mm, ss)
109
110         #%%
111         ####################
```

```
112         # Update the Starts and Fails file lists
113         ####################
114         with open(homePath + "\\All_Access_" + "{0:0=4d}".format(
    inst_long) + ".txt", "a+") as f:
115             f.close()
116         with open(homePath + "\\Starts.txt", "a") as f: # Update the
     tracker
117             f.write(str(start_inst) + "\n")
118         with open(homePath + "\\Fails.txt", 'r') as fin:
119             data = fin.read().splitlines(True)
120         with open(homePath + "\\Fails.txt", 'w') as fout:
121             fout.writelines(data[1:])
122 
123         #%%
124         ####################
125         # Add python functions module
126         ####################
127         import STK_FnsForPy3 as sf
128         data = [] # stores all errors returned from STK
129 
130         #%%
131         ####################
132         # Establish TCP/IP communication between this script and STK
133         ####################
134         print("check1")
135         import socket #imports a python class needed to establish
    TCP/IP
136         HOST = socket.gethostname()
137         PORT = 5001
138         s = None #s is a socket object used to pass info from Python
     to STK (must send bits not strings)
139         for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC,
```

```
    socket.SOCK_STREAM):
140             af, socktype, proto, cannonname, sa = res
141             try:
142                 s = socket.socket(af, socktype, proto)
143             except socket.error as msg:
144                 s = None
145                 continue
146             try:
147                 s.connect(sa)
148             except socket.error as msg:
149                 s.close()
150                 s = None
151                 continue
152             break
153         if s is None:
154             print ('could not open socket')
155             sys.exit(1)
156         print("check2")
157         #%%
158         ####################
159         # Create a new scenario
160         ####################
161         sf.CreateScenario(s, data, 'CALSx')
162         print("check3")
163         s.send(("Parallel / AutomaticallyComputeInParallel On \n").
    encode())
164         s.send(("Parallel / Configuration ParallelType Local \n").
    encode())
165         s.send(("Parallel / Configuration NumberOfLocalCores 24 \n")
    .encode())
166         #%%
167         print("check4")
```

```
168         ####################
169         # Set the time period
170         ####################
171         startdate = time_start.strftime("%d %b %Y")
172         starttime = time_start.strftime("%H:%M:%S.00")
173         enddate = time_end.strftime("%d %b %Y")
174         endtime = time_end.strftime("%H:%M:%S.00")
175         sf.ScenarioTime(s, data, startdate, starttime, enddate,
    endtime)
176
177         #%%
178         ####################
179         # Import GPS Constellation and rename to GPS##
180         ####################
181         sf.AddGPS(s, data, 'ID All', startdate, starttime, enddate,
    endtime)
182         for i in [x for x in range(1,33) if x !=4]: #There is no
    GSP04
183             sf.SendConCmd(s, data, 'Rename */Satellite/gps-' + "
    {0:0=2d}".format(i) + '* GPS' + "{0:0=2d}".format(i) + '\n')
184
185         #%%
186         ####################
187         # Create the aircraft
188         ####################
189         for x in range(0,27): #number of aircraft paths (14 on
    either side of sensor + 1 overhead)
190             ac_long = longitude - maxdist_horizontal + offset_long*x
     #Increment longitude
191             ac_long_str = "{:.6f}".format(ac_long)
192             ac_start = time_start - datetime.timedelta(seconds=
    offset_time) #reset the time
```

```
193             if ac_long >= -75 and ac_long <= -65:
194                 for y in range(1,73): #Aircraft leaving every 20 min
 (73)
195                     ac_start = ac_start + datetime.timedelta(seconds
=offset_time) #Increment the time and pull out the date/time
strings
196                     ac_startdate = ac_start.strftime("%d %b %Y")
197                     ac_starttime = ac_start.strftime("%H:%M:%S.00")
198                     ac_name = 'AC_' + "{0:0=4d}".format(inst_long -
26 + x) + '_' + "{0:0=2d}".format(y) #format name as "
AC_LongIter_TimeIter"
199                     sf.CreateAircraft(s, data, ac_name, ac_startdate
, ac_starttime, ac_lat_start, ac_long_str, ac_lat_stop,
ac_long_str, ac_altitude, ac_speed)
200
201         #%%
202         ####################
203         # Create Sensor Locations
204         ####################
205         fac_long_str = "{:.6f}".format(longitude)
206         for y in range(0,5): # latitude (5)
207             fac_lat = fac_lat_start + offset_lat * y
208             fac_lat_str = "{:.6f}".format(fac_lat)
209             fac_name = 'F_'"{0:0=4d}".format(inst_long) + '_' + "
{0:0=1d}".format(y + 1) #format name as "F_LongIter_LatIter"
210             sf.CreateFacility(s, data, fac_name, fac_lat_str,
fac_long_str, fac_altitude)
211
212             ####################
213             # Create a sensor for each GPS satellite
214             ####################
215             for i in [z for z in range(1,33) if z != 4]:
```

```
216                 GPS_name = 'GPS' + "{0:0=2d}".format(i)
217                 # Create a pointing vector from site to GPS
218                 vec_name = fac_name + '_to_' + GPS_name
219                 sf.SendConCmd(s, data, 'VectorTool * Facility/' +
    fac_name + ' Create Vector ' + vec_name + ' "Displacement" "/
    Facility/' + fac_name + ' Center" "/Satellite/GPS' + "{0:0=2d}".
    format(i) + ' Center"\n')
220
221                 # Add sensor to site
222                 sen_name = 'S_'"{0:0=2d}".format(i)
223                 sen_shape = 'SimpleCone'
224                 sen_dia = '4.4'
225                 #Create the sensor
226                 sf.SendConCmd(s, data, 'New / */Facility/' +
    fac_name + '/Sensor ' + sen_name + '\n')
227                 #Define sensor shape
228                 sf.SendConCmd(s, data, 'Define */Facility/' +
    fac_name + '/Sensor/' + sen_name + ' ' + sen_shape + ' ' +
    sen_dia + '\n')
229                 #Point the sensor
230                 sf.SendConCmd(s, data, 'Point */Facility/' +
    fac_name + '/Sensor/' + sen_name + ' AlongVector "Facility/' +
    fac_name + ' ' + vec_name + '" "Facility/' + fac_name + ' North"\
    n')
231                 #Restrict the sensor
232                 #sf.SendConCmd(s, data, 'SetConstraint */Facility/'
    + fname + '/Sensor/' + sname + ' ElevationAngle Min 26\n') #26
    deg horizon limit (not required if pointing instead of tracking)
233                 sf.SendConCmd(s, data, 'SetConstraint */Facility/' +
     fac_name + '/Sensor/' + sen_name + ' Range Max 7112\n') #7112 m
    range limit
234         timer_scenario = (time.time() - timer_start)/60.0
```

```
235         print(timer_scenario)
236         #%%
237         ####################
238         # Create Reports
239         ####################
240         s.send(("ExportConfig / Connection Headers Off
    KeepReportLines Off ShowStartStop Off WriteReportTitle Off
    WriteObjectNames Off WriteSectionTitles Off" + '\n').encode())
241         style = 'AER'
242         #facility, sensor, ac_longitude, launch instance
243         for i in range(restart_long,6):
244             for j in [x for x in range(restart_sat,33) if x !=4]:
245                 timer_runtime = (time.time() - timer_start)/60.0
246                 print("Latitude " + str(i) + "/5, Satellite " + str(
    j) + "/32, elapsed time: " + str(timer_runtime) + " minutes")
247                 obj1 = 'Facility/F_' + "{0:0=4d}".format(inst_long)
    + '_' + "{0:0=1d}".format(i) + '/Sensor/S_' + "{0:0=2d}".format(j
    )
248                 name1 = 'F_' + "{0:0=4d}".format(inst_long) + '_' +
    "{0:0=1d}".format(i) + '_S_' + "{0:0=2d}".format(j)
249                 for k in range(restart_aclong,27):
250                     for l in range(restart_time,73):
251                         restart_long = 1
252                         restart_sat = 1
253                         restart_aclong = 0
254                         restart_time = 1
255                         ac_long = longitude - maxdist_horizontal +
    offset_long*k #Increment longitude
256                         if ac_long >= -75 and ac_long <= -65:
257                             obj2 = 'Aircraft/AC_' + "{0:0=4d}".
    format(inst_long - 26 + k) + '_' + "{0:0=2d}".format(l)
258                             name2 = 'AC_' + "{0:0=4d}".format(
```

```
    inst_long - 26 + k) + '_' + "{0:0=2d}".format(l)
259                             title = 'Report_' + name1 + '_' + name2
260                             sf.CreateGenReports(s, data, obj1, obj2
    , reportPath, title, style)
261                             while True:
262                                 try:
263                                     with open(reportPath + "\\" +
    title + ".txt") as r, open(homePath + "\\All_Access_" + "{0:0=4d}
    ".format(inst_long) + ".txt", "a") as f:
264                                         f.write(title[7:31])
265                                         f.write(r.read())
266                                         time.sleep(0.1)
267                                     break
268                                 except (FileNotFoundError, OSError,
     PermissionError) as z:
269                                     pass
270                             while True:
271                                 try:
272                                     os.remove(reportPath + "\\" +
    title + ".txt")
273                                     break
274                                 except (FileNotFoundError, OSError,
    PermissionError) as z:
275                                     print("PermissionError")
276                                     continue
277
278         #%%
279         ####################
280         # Consolidate Data
281         ####################
282      #    read_files = glob.glob(reportPath + "\\*.txt") # windows
283      #    #read_files = glob.glob(reportPath + "/*.txt") # linux
```

```
284     #
285     #     # Create one file which consolidates all the rest of the
    data
286     #     with open(homePath + "\\All_Access_" + "{0:0=4d}".format(
    inst_long) + ".txt", "wb") as outfile: # windows
287     #     #with open(reportPath + "/All_Access_" + "{0:0=4d}".format(
    inst_long) + ".txt", "wb") as outfile: # linux
288     #         for f in read_files:
289     #             with open(f, "rb") as infile:
290     #                 outfile.write((f[-28:-4] +' \n').encode())
291     #                 outfile.write(infile.read())
292     #     timer_runtime = (time.time() - timer_start)/60.0
293     #     print("Inst",inst_long,"setup",timer_scenario,"runtime",
    timer_runtime)
294     # a = s.send(("Access */Aircraft/AC_0013_08 */Facility/F_0031_1/
    Sensor/S_05 TimePeriod UseScenarioInterval \n").encode())
```

Appendix B. R Code to Ingest Data

```
root <- rprojroot::find_root(rprojroot::is_rstudio_project)
data.path.raw <- base::file.path(root, "Data_Raw")
data.path.parsed <- base::file.path(root, "Data_Parsed")
opl.path <- base::file.path(root, "OPL")

# Collect all All_Access_Reports (from data_raw) in 'data.list'
data.list <- base::list.files(path = data.path.raw,
                              pattern = "All_Access",
                              full.names = TRUE)
# Collect all All_Access_Reports (from data_parsed) in 'data.list.
    parsed'
data.list.parsed <- base::list.files(path = data.path.parsed,
                              pattern = "All_Access",
                              full.names = TRUE)
# Discover which files have not yet been parsed
data.list.incomplete <- stringr::str_extract(data.list, "[0-9]{4}")
    %in% stringr::str_extract(data.list.parsed, "[0-9]{4}")==F
data.list.to_do <- data.list[data.list.incomplete]

# Define function to parse the raw data
parse.data <- function(data.raw)
  {
  # Create empty data.frame of appropriate size
  num.ac_per_long <- 72 # Number of aircraft that follow each flight
     path
  num.long_vis <- 27 # Maximum number of flight paths visible to a
    single sensor
  num.sen <- 5 # Number of sensors in a single data report
  inst <- base::as.integer(stringr::str_extract(data.raw, "[0-9]{4}"
    )) # Determine the instance number
```

```
26  if(inst < num.long_vis)
27    {
28    ac_long <- 1:inst # List the longitude indexes for each flight
     path in view
29    num.ac <- num.ac_per_long * inst # Total number of aircraft in
     the instance
30    }
31  else
32    {
33    ac_long <- (inst - num.long_vis + 1):inst
34    num.ac <- num.ac_per_long * num.long_vis
35    }
36  data.parsed <- base::as.data.frame(base::matrix(nrow = num.ac,
37                                                  ncol = num.sen))
38
39  # Create column names
40  names.col <- base::vector()
41  for(sen in 1:num.sen)
42    {
43    names.col <- c(names.col,
44                   base::paste("F",
45                               stringr::str_pad(base::as.character(
     inst), 4, pad = "0"),
46                               base::as.character(sen), sep = "_"))
47    }
48  colnames(data.parsed) <- names.col
49
50  # Create row names
51  names.row <- base::vector()
52  for(long in ac_long)
53    {
54    for(ac in 1:num.ac_per_long)
```

```
55      {
56      names.row <- c(names.row,
57                     base::paste("AC",
58                                 stringr::str_pad(base::as.character
    (long), 4, pad = "0"),
59                                 stringr::str_pad(base::as.character
    (ac), 2, pad = "0"), sep = "_"))
60      }
61    }
62  rownames(data.parsed) <- names.row
63
64  # Read in the data line by line, updating the data.frame with
     observations
65  con = base::file(base::file.path(data.raw), "r") # Open the file
66  while ( TRUE )
67    {
68    contents = base::readLines(con, n = 1) # Read line by line
     through the data file
69    if ( base::length(contents) == 0 )
70      { # Break after the last line or if the file is empty
71      break
72      }
73    if ( base::substr(contents, 1, 1) == "F" )
74      {
75      sensor.long <- base::strtoi(base::substr(contents, 3, 6), 10)
     # Retrieve sensor longitude index
76      sensor.lat <- base::strtoi(base::substr(contents, 8, 8), 10) #
      Retrieve sensor latitude index
77      sensor.index <- sensor.lat
78      sensor.sat <- base::strtoi(base::substr(contents, 12, 13), 10)
     # Retrieve sensor satellite index
79      aircraft.long <- base::strtoi(base::substr(contents, 18, 21),
```

```
  10) # Retrieve aircraft longitude index
80      aircraft.time <- base::strtoi(base::substr(contents, 23, 24),
  10) # Retrieve aircraft launch time index
81      aircraft.index <- base::match(paste("AC",
82                                          stringr::str_pad(base::as.
  character(aircraft.long), 4, pad = "0"),
83                                          stringr::str_pad(base::as.
  character(aircraft.time), 2, pad = "0"), sep = "_"),
84                                    names.row)
85      }
86    if ( !base::is.na(base::strtoi(base::substr(contents, 1, 1))) )
87      { # An integer here means that the aircraft was detected
88      data.parsed[aircraft.index, sensor.index] <- 1 # Store the
  detection information
89      }
90    }
91  base::close(con)
92
93  # Store the data.frame with .RData extension (to data_parsed)
94  name.data.parsed <- base::paste0("All_Access_",
95                                   stringr::str_pad(base::as.
  character(inst), 4, pad = "0"),
96                                   "_Parsed.RData")
97  base::assign(base::paste0("All_Access_",
98                            stringr::str_pad(base::as.character(inst
  ), 4, pad = "0")),
99               data.parsed)
100 base::save(list = base::paste0("All_Access_",
101                                stringr::str_pad(base::as.character
  (inst), 4, pad = "0")),
102           file = base::file.path(data.path.parsed, name.data.
  parsed))
```

```
  }

#   For each report in data.list_to_do, apply parse.data()
count.completed_files <- 0
for(item in data.list.to_do)
  {
  parse.data(item)
  count.completed_files = count.completed_files + 1
  base::print(base::paste0(count.completed_files,
                           "/",
                           base::length(data.list.to_do),
                           " complete."))
  }

# Merge all parsed data together
# Get a list of all .RData files
data.list.parsed <- base::list.files(path = data.path.parsed,
                                     pattern = "All_Access",
                                     full.names = TRUE)

# Create an environment to load all the parsed df into
ex <- base::new.env()

# Load the parsed files
for( item in data.list.parsed )
  {
  base::load(item, ex)
  }
base::rm(item)

# Get a list of the parsed df's
data.list.str <- base::objects(ex)
```

```
135
136 # Create an empty tibble to hold the merged data (tibbles merge more
      easily)
137 data.merged <- tibble::tibble(rowname = character())
138 # Open each df as a tibble and merge with data.merged
139 for( df in data.list.str )
140   {
141   data.merged <- dplyr::full_join(data.merged,
142                                   tibble::as_tibble(tibble::rownames
      _to_column(base::get(df, envir = ex))),
143                                   by = "rowname")
144   base::print(df)
145   }
146 base::rm(ex, data.list.str, data.list.parsed, df)
147
148 # Convert back to a data.frame with row names
149 data.row.names <- base::unlist(stringr::str_split(data.merged$
      rowname,
150                                                   pattern = " "))
151 data.merged.df <- base::as.data.frame(data.merged[, -1])
152 base::row.names(data.merged.df) = data.row.names
153 base::rm(data.row.names, data.merged)
154
155 # Convert all NA's to 0's
156 data.merged.df[base::is.na(data.merged.df)] <- 0
157
158 # Write to an OPL data file (.dat)
159 data.file = base::file.path(opl.path, "data_fixed.dat")
160 base::cat("d = [",
161           file=data.file,
162           append=FALSE,
163           sep = "\n")
```

```
for( i in 1:dim(data.merged.df)[1] )
  {
  base::cat(base::paste("[",
                        base::paste(data.merged.df[i,],
                                    collapse = " "),
                        "]",
                        collapse = " "),
            file=data.file,
            append=TRUE,
            sep = "\n")
  base::print(i)
  }
base::cat("];",
          file=data.file,
          append=TRUE,
          sep = "\n")

# Save data to a .RData file
base::save(data.merged.df,
           file = file.path(data.path.parsed,
                            "Data256.RData"))
```

Appendix C. Python Code to Run CPLEX

```
# Setup the environment with necessary dependencies
import subprocess
import re
import os

# homePath is where the outputs are stored on the network
homePath = 'L:\Research\ENS\\2020 STUDENT FOLDERS\MASTERS STUDENTS\
    OPER\Hufstetler, Brandon J. - Capt\WORKING FOLDER\OPL' # This is
    where reports are stored

# oplPath is where the data and ops file are stored on the local
    machine
oplPath = r'C:\Users\bhufstet\Documents\Thesis'

# all txt_ variables are OPL code
txt_first = """
 /*********************************************
 * OPL 12.8.0.0 Model
 * Author: Capt Brandon J. Hufstetler
 * Creation Date: Oct 27, 2019 at 5:58:13 PM
 *
 * THIS MODEL CALCULATES THE MAX AIRCRAFT DETECTABLE
 * FROM AIRCRAFT ROWS 1-3 GIVEN A FIXED NUMBER OF SENSORS
 * Over 1 degree of longitude, there are:
 * 18432 Aircraft and 1280 Sensor Sites
 * Maximum number of paths is 230
 **********/

 /* Parameters */
 """
```

```
28
29 txt_last = """
30  range A = 1..16560;
31  range S = 1..1280;
32  int d[A][S] = ...;
33  int numAC = paths * 72;
34  int numS = (paths + 26) * 5;
35
36  /* Decision Variables */
37  dvar boolean x[S];
38  dvar boolean g[A];
39  dexpr int aircraft_und =
40   sum(a in 1..numAC) g[a];
41
42  /* Model */
43  minimize aircraft_und;
44
45  /* Subject To */
46  subject to{
47   forall(a in 1..numAC)
48     /* Calculate aircraft detection */
49     ac_det: g[a]+ sum(s in 1..numS) x[s]*d[a][s] >= 1;
50   /* Max sensors allowed */
51   maxSen: sum(s in 1..numS) x[s] == sensors;
52  }
53  """
54
55 # minsen determined from minimal set covering problem
56 minsen = [3, 3, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8, 8,
      8, 9, 9, 9, 10, 10, 10]
57
58 # run to optimal solution for all tractable cases
```

```
for sen in range(7, 9):
    for paths in range(23,27):
        # fill in the remaining OPL code specific to the instance
        txt_paths = "int paths = " + str(paths) + ";"
        txt_sen = "int sensors = " + str(sen) + ";"

        # create a .mod file with the OPL text
        with open(oplPath + "\\max_ac_iter.mod", "w+") as f:
            f.write(txt_first + txt_paths + "\n " + txt_sen +
    txt_last)

        # command is sent to the command prompt
        command = r'''oplrun -p ""C:\Users\bhufstet\Documents\Thesis
    "" ""max_ac_iter""'''

        # send command to command prompt and pull all outputs
        pipe = subprocess.Popen(command, shell=True, stdout=
    subprocess.PIPE, stderr=subprocess.PIPE)

        # Find the objective output and store to output file
        while True:
            line = pipe.stdout.readline()
            if line:
                print(line)
                objective = re.match(b"OBJECTIVE: [0-9]+", line)
                if objective:
                    print(objective[0])
                    first = 11
                    last = len(objective[0]) + 2
                    obj_val = (objective[0][first:last]).decode('UTF
    -8')
                    with open(homePath + "\\max_ac_outputs_all.txt",
```

```
    "a") as f:
87                          f.write(str(paths) + "," + str(sen) + "," +
    obj_val + "\n")
88              if not line:
89                  break
90          os.remove(oplPath + "\\max_ac_iter.mod")
```

Appendix D. R Code to Run Genetic Algorithm

```
root <- rprojroot::find_root(rprojroot::is_rstudio_project)
data.path.raw <- base::file.path(root, "Data_Raw")
data.path.parsed <- base::file.path(root, "Data_Parsed")
opl.path <- base::file.path(root, "OPL")

# Collect all All_Access_Reports (from data_raw) in 'data.list'
data.list <- base::list.files(path = data.path.raw,
                              pattern = "All_Access",
                              full.names = TRUE)
# Collect all All_Access_Reports (from data_parsed) in 'data.list.
    parsed'
data.list.parsed <- base::list.files(path = data.path.parsed,
                              pattern = "All_Access",
                              full.names = TRUE)
# Discover which files have not yet been parsed
data.list.incomplete <- stringr::str_extract(data.list, "[0-9]{4}")
    %in% stringr::str_extract(data.list.parsed, "[0-9]{4}")==F
data.list.to_do <- data.list[data.list.incomplete]

# Define function to parse the raw data
parse.data <- function(data.raw)
  {
  # Create empty data.frame of appropriate size
  num.ac_per_long <- 72 # Number of aircraft that follow each flight
     path
  num.long_vis <- 27 # Maximum number of flight paths visible to a
    single sensor
  num.sen <- 5 # Number of sensors in a single data report
  inst <- base::as.integer(stringr::str_extract(data.raw, "[0-9]{4}"
    )) # Determine the instance number
```

```
26  if(inst < num.long_vis)
27    {
28    ac_long <- 1:inst # List the longitude indexes for each flight
     path in view
29    num.ac <- num.ac_per_long * inst # Total number of aircraft in
     the instance
30    }
31  else
32    {
33    ac_long <- (inst - num.long_vis + 1):inst
34    num.ac <- num.ac_per_long * num.long_vis
35    }
36  data.parsed <- base::as.data.frame(base::matrix(nrow = num.ac,
37                                                  ncol = num.sen))
38
39  # Create column names
40  names.col <- base::vector()
41  for(sen in 1:num.sen)
42    {
43    names.col <- c(names.col,
44                   base::paste("F",
45                               stringr::str_pad(base::as.character(
     inst), 4, pad = "0"),
46                               base::as.character(sen), sep = "_"))
47    }
48  colnames(data.parsed) <- names.col
49
50  # Create row names
51  names.row <- base::vector()
52  for(long in ac_long)
53    {
54    for(ac in 1:num.ac_per_long)
```

```
55      {
56      names.row <- c(names.row,
57                     base::paste("AC",
58                                 stringr::str_pad(base::as.character
    (long), 4, pad = "0"),
59                                 stringr::str_pad(base::as.character
    (ac), 2, pad = "0"), sep = "_"))
60      }
61    }
62  rownames(data.parsed) <- names.row
63
64  # Read in the data line by line, updating the data.frame with
     observations
65  con = base::file(base::file.path(data.raw), "r") # Open the file
66  while ( TRUE )
67    {
68    contents = base::readLines(con, n = 1) # Read line by line
     through the data file
69    if ( base::length(contents) == 0 )
70      { # Break after the last line or if the file is empty
71      break
72      }
73    if ( base::substr(contents, 1, 1) == "F" )
74      {
75      sensor.long <- base::strtoi(base::substr(contents, 3, 6), 10)
     # Retrieve sensor longitude index
76      sensor.lat <- base::strtoi(base::substr(contents, 8, 8), 10) #
      Retrieve sensor latitude index
77      sensor.index <- sensor.lat
78      sensor.sat <- base::strtoi(base::substr(contents, 12, 13), 10)
     # Retrieve sensor satellite index
79      aircraft.long <- base::strtoi(base::substr(contents, 18, 21),
```

```
  10) # Retrieve aircraft longitude index
80      aircraft.time <- base::strtoi(base::substr(contents, 23, 24),
  10) # Retrieve aircraft launch time index
81      aircraft.index <- base::match(paste("AC",
82                                          stringr::str_pad(base::as.
  character(aircraft.long), 4, pad = "0"),
83                                          stringr::str_pad(base::as.
  character(aircraft.time), 2, pad = "0"), sep = "_"),
84                                    names.row)
85      }
86     if ( !base::is.na(base::strtoi(base::substr(contents, 1, 1))) )
87      { # An integer here means that the aircraft was detected
88      data.parsed[aircraft.index, sensor.index] <- 1 # Store the
  detection information
89      }
90     }
91  base::close(con)
92
93  # Store the data.frame with .RData extension (to data_parsed)
94  name.data.parsed <- base::paste0("All_Access_",
95                                   stringr::str_pad(base::as.
  character(inst), 4, pad = "0"),
96                                   "_Parsed.RData")
97  base::assign(base::paste0("All_Access_",
98                            stringr::str_pad(base::as.character(inst
  ), 4, pad = "0")),
99               data.parsed)
100 base::save(list = base::paste0("All_Access_",
101                                stringr::str_pad(base::as.character
  (inst), 4, pad = "0")),
102           file = base::file.path(data.path.parsed, name.data.
  parsed))
```

```
  }

#   For each report in data.list_to_do, apply parse.data()
count.completed_files <- 0
for(item in data.list.to_do)
  {
  parse.data(item)
  count.completed_files = count.completed_files + 1
  base::print(base::paste0(count.completed_files,
                           "/",
                           base::length(data.list.to_do),
                           " complete."))
  }

# Merge all parsed data together
# Get a list of all .RData files
data.list.parsed <- base::list.files(path = data.path.parsed,
                                     pattern = "All_Access",
                                     full.names = TRUE)

# Create an environment to load all the parsed df into
ex <- base::new.env()

# Load the parsed files
for( item in data.list.parsed )
  {
  base::load(item, ex)
  }
base::rm(item)

# Get a list of the parsed df's
data.list.str <- base::objects(ex)
```

```
135
136 # Create an empty tibble to hold the merged data (tibbles merge more
      easily)
137 data.merged <- tibble::tibble(rowname = character())
138 # Open each df as a tibble and merge with data.merged
139 for( df in data.list.str )
140   {
141   data.merged <- dplyr::full_join(data.merged,
142                                   tibble::as_tibble(tibble::rownames
      _to_column(base::get(df, envir = ex))),
143                                   by = "rowname")
144   base::print(df)
145   }
146 base::rm(ex, data.list.str, data.list.parsed, df)
147
148 # Convert back to a data.frame with row names
149 data.row.names <- base::unlist(stringr::str_split(data.merged$
      rowname,
150                                                   pattern = " "))
151 data.merged.df <- base::as.data.frame(data.merged[, -1])
152 base::row.names(data.merged.df) = data.row.names
153 base::rm(data.row.names, data.merged)
154
155 # Convert all NA's to 0's
156 data.merged.df[base::is.na(data.merged.df)] <- 0
157
158 # Write to an OPL data file (.dat)
159 data.file = base::file.path(opl.path, "data_fixed.dat")
160 base::cat("d = [",
161           file=data.file,
162           append=FALSE,
163           sep = "\n")
```

```
for( i in 1:dim(data.merged.df)[1] )
  {
  base::cat(base::paste("[",
                        base::paste(data.merged.df[i,],
                                    collapse = " "),
                        "]",
                        collapse = " "),
            file=data.file,
            append=TRUE,
            sep = "\n")
  base::print(i)
  }
base::cat("];",
          file=data.file,
          append=TRUE,
          sep = "\n")

# Save data to a .RData file
base::save(data.merged.df,
           file = file.path(data.path.parsed,
                            "Data256.RData"))
```

Appendix E. STK Functions For Python

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 01 09:52:09 2017

The Air Force Institute of Technology
Center for Space Research and Assurance

This code is intended to be called by the Python script created to run the scenario
The Driver script extablishes a TCP/IP connection between Python and STK.  This
code takes commands intended for Connect, does minor error checking and most importantly
looks for an ACK or NACK back from Connect.  Upon completion of the Driver program,
the Driver program will print a list of all Connect commands for which a NACK was
received.  This facilitates debugging considerably!  Upon receipt of multiple failed
Connect commands, the coder should look to the first one in the last as the most likely
source of failure.

Common Connect commands have methods below that may be called to maximize error
checking and Intellisense capability

Use the function SendConCmd(s, data,cmdStr) to at least have the failed Connect
command identification capability
```

```
22
23 @author: Mark Bateman, Capt, USAF
24
25 Modified and tested by Broden Kelly and David Meyer
26
27 THIS IS MADE FOR PYTHON 3.0+!
28 """
29
30 #starting STK talking to python
31 from __future__ import (absolute_import, division,
32                         print_function, unicode_literals)
33 #import socket
34 #import sys
35 import time
36 '''
37 HOST = socket.gethostname()
38 PORT = 5001
39
40 s = None
41 for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.
      SOCK_STREAM):
42     af, socktype, proto, canonname, sa = res
43     try:
44         s = socket.socket(af, socktype, proto)
45     except socket.error as msg:
46         s = None
47         continue
48     try:
49         s.connect(sa)
50     except socket.error as msg:
51         s.close()
52         s = None
```

```
53          continue
54      break
55  if s is None:
56      print('Could not open socket - Please start STK first')
57      sys.exit(1)
58  '''
59  BUFFER_SIZE = 1024
60  data=[] #needed to receive ACKs or NACKs from STK for trouble
        shooting
61
62  """
63  Send commands to Connect, get ACK/NACK in return.  Use for any
        Connect call
64  for which a special method has not been created below
65  """
66  #%%
67  def SendConCmd(s, data,cmdStr):
68      BUFFER_SIZE = 2048
69      cmdFinal = str(cmdStr)
70      try:
71              s.send(cmdFinal)
72              time.sleep(1)
73              check = s.recv(BUFFER_SIZE)
74        #       print(cmdFinal + ":  " + data)
75      except TypeError:
76              cmdBytes = cmdFinal.encode()
77              s.send(cmdBytes)
78              time.sleep(1)
79              check = s.recv(BUFFER_SIZE)
80            #  finalData = check.decode("utf-8")
81            #  print(cmdFinal + ":  " + finalData)
82              if str(check.decode("utf-8")) != 'ACK':
```

```
                data.append(cmdFinal)
#%%
# Scenario Creation

def CreateScenario(s, data,name):
    """Connects to STK and creates a scenario with the given name
   the cannot contain any spaces"""
    s.send(('Unload / *\n').encode())
    time.sleep(0.1)
    check=s.recv(BUFFER_SIZE)
    if str(check.decode("utf-8")) != 'ACK':
        data.append('Old Scenario Unloaded')
    if type(name) != str or ' ' in name:
        raise TypeError('Create Scenario: Name must be a string with
    no spaces')
    else:
        s. send(('New / Scenario '+str(name)+'\n').encode())
        time.sleep(0.1)
        check1=s.recv(BUFFER_SIZE)
        if str(check1.decode("utf-8")) != 'ACK':
            data.append('Scenario Created')

#%%
#Date and Time

def ScenarioTime(s, data,startdate,starttime,enddate,endtime):
    """Establishes the start and stop time of the scenario using Day
    Month Year HH:MM:SS.SS"""
    if type(startdate) != str:
        raise TypeError('Scenario Time: Start Date needs to be a
   string in the form of Day Month Year')
    elif type(starttime) != str:
```

```
111         raise TypeError('Scenario Time: Start Time needs to be a
    string in the form of HH:MM:SS.SS')
112     elif type(enddate) != str:
113         raise TypeError('Scenario Time: End Date needs to be a
    string in the form of Day Month Year')
114     elif type(endtime) != str:
115         raise TypeError('Scenario Time: End Time needs to be a
    string in the form of HH:MM:SS.SS')
116     else:
117         s.send(('SetTimePeriod * \"' + startdate + ' ' + starttime +
     '\" \"' + enddate + ' ' + endtime + '\" \"' + '\n').encode())
118         time.sleep(0.1)
119         check=s.recv(BUFFER_SIZE)
120         if str(check.decode("utf-8")) != 'ACK':
121             data.append('Set Scenario Time')
122
123 #%%
124 #Facilities
125
126 def CreateFacility(s, data,name,latitude,longitude,altitude):
127     """Creates a Facility with the given name at the given location
    in Geodectic coordinates"""
128     if type(name) != str or ' ' in name:
129         raise TypeError('Create Facility: Facility name must be a
    string with no spaces')
130     else:
131         s.send(('New / */Facility/ ' +  name + '\n').encode())
132         time.sleep(0.1)
133         check=s.recv(BUFFER_SIZE)
134         if str(check.decode("utf-8")) != 'ACK':
135             data.append('Create Facility')
136     if type(latitude) != str or type(longitude) != str or type(
```

```
    altitude) != str:
137         raise TypeError('Create Facility: The Latitude, Longitude,
    and Altitude of the facility must be input as a string')
138     else:
139         s.send(("SetPosition */Facility/" + name + " Geodetic " +
    latitude +' ' + longitude + ' ' + altitude  + '\n').encode())
140         time.sleep(0.1)
141         check1=s.recv(BUFFER_SIZE)
142         if str(check1.decode("utf-8")) != 'ACK':
143             data.append('Set Facility Position')
144
145 #%%
146 #Targets
147 def CreateTarget(s, data,name,latitude,longitude,altitude):
148     """Creates a Target with the given name at the given location in
     Geodectic coordinates"""
149     if type(name) != str or ' ' in name:
150         raise TypeError('Create Target: Target name must be a string
     with no spaces')
151     else:
152         s.send(('New / */Target/ ' +  name + '\n').encode())
153         time.sleep(0.1)
154         check=s.recv(BUFFER_SIZE)
155         if str(check.decode("utf-8")) != 'ACK':
156             data.append('Create Target')
157     if type(latitude) != str or type(longitude) != str or type(
    altitude) != str:
158         raise TypeError('Create Target: The Latitude, Longitude, and
     Altitude of the facility must be input as a string')
159     else:
160         s.send(("SetPosition */Target/" + name + " Geodetic " +
    latitude +' ' + longitude + ' ' + altitude  + '\n').encode())
```

```
161         time.sleep(0.1)
162         check1=s.recv(BUFFER_SIZE)
163         if str(check1.decode("utf-8")) != 'ACK':
164             data.append('Set Target Position')
165 
166 #%%
167 #Satellites
168 
169 def CreateSatellite(s, data,name,epoch,SMA,eccentricity,inclination,
    argofperigee,raan,meananom):
170     """Creates a satellite with the given name and orbital elements
    (Epoch, semi-major axis(in meters), eccentricity, inclination,
    argument of perigee, RAAN, and mena anomaly) using classic J2
    Perturbations for the length of the scenario"""
171     if type(name) != str or ' ' in name:
172         raise TypeError('Create Satellite: Satellite name must be a
    string with no spaces')
173     else:
174         s.send(('New / */Satellite ' + name + '\n').encode())
175         time.sleep(0.1)
176         check=s.recv(BUFFER_SIZE)
177         if str(check.decode("utf-8")) != 'ACK':
178             data.append('Create Satellite')
179     if type(epoch) != str or type(SMA) != str or type(eccentricity)
    != str or type(inclination) != str or type(argofperigee) != str
    or type(raan) != str or type(meananom) != str:
180         raise TypeError('Create Satellite: The epoch, semi-major
    axis, eccentricity, inclination, argofperigee, raan, and mean
    anomoly must be input as strings')
181     else:
182         s.send(("SetState */Satellite/"+name+" Classical
    J2Perturbation UseScenarioInterval 5 J2000 \""+epoch+"\" " + SMA
```

```
    + " "+eccentricity+" "+inclination+" "+argofperigee+" "+raan+" "+
    meananom+" "  + '\n').encode())
183         time.sleep(0.1)
184         check1=s.recv(BUFFER_SIZE)
185         if str(check1.decode("utf-8")) != 'ACK':
186             data.append('Set Satellite State')
187 
188 def CreateWalker(s, data,name,numplane,numsat):
189     """Creates a constellation via the Walker command with number of
     planes and number of satellites per plane"""
190     if type(name) != str or ' ' in name or type(numplane) != str or
    type(numsat) != str:
191         raise TypeError('Create Constellation: The name must be a
    string with no spaces and the number of planes and number of
    satellites must be a string')
192     else:
193         s.send(("Walker */Satellite/" + name + ' Type Delta
    NumPlanes '+str(numplane)+" NumSatsPerPlane "+str(numsat)+"
    InterPlanePhaseIncrement "+str(int(numplane)-1)+" ColorByPlane
    Yes" + '\n').encode())
194         time.sleep(0.1)
195         check=s.recv(BUFFER_SIZE)
196         if str(check.decode("utf-8")) != 'ACK':
197             data.append('Create Constellation')
198 
199 #%%
200 #Sensors
201 
202 def CreateFixedSensor(s, data,parent_type,parent_name,sensor_name,
    shape,dim):
203     """Creates a fixed snesor and places the sensor on the given
    object, then specify the shape and dimensions of the sensor"""
```

```
204     if type(parent_name) != str or ' ' in parent_name or type(
    sensor_name) != str or ' ' in sensor_name or type(shape) != str
    or type(dim) != str or type(parent_type) != str or ' ' in
    parent_type:
205         raise TypeError('Create Fixed Sensor: The parent plaform
    type, name, and sensor name must be strings with no spaces. Shape
     and dimensions must be strings')
206     else:
207         s.send(("New / */"+str(parent_type)+"/"+parent_name+"/Sensor
     "+sensor_name + '\n').encode())
208         time.sleep(0.1)
209         check=s.recv(BUFFER_SIZE)
210         if str(check.decode("utf-8")) != 'ACK':
211             data.append('Create Fixed Sensor')
212         s.send(("Define */"+str(parent_type)+"/"+parent_name+"/
    Sensor/"+sensor_name+" "+shape+" "+dim+ '\n').encode())
213         time.sleep(0.1)
214         check1=s.recv(BUFFER_SIZE)
215         if str(check1.decode("utf-8")) != 'ACK':
216             data.append('Define Fixed Sensor')
217
218 def CreatePointingSensor(s, data,parent_type,parent_name,sensor_name
    ,shape,dim,targtype,targname):
219     """Creates a new pointing sensor on the given object, then
    specifies the shape and dimensions of the sensor, followed by the
     object(type and name) it is going to track"""
220     if type(parent_name) != str or ' ' in parent_name or type(
    sensor_name) != str or ' ' in sensor_name or type(shape) != str
    or type(dim) != str or type(parent_type) != str or ' ' in
    parent_type or type(targtype) != str or ' ' in targtype or type(
    targname) != str or ' ' in targname:
221         raise TypeError('Create Pointing Sensor: The parent plaform
```

```
    type, name, sensor name, target type and target name must be
    strings with no spaces. Shape and dimensions must be strings')
222     else:
223         s.send(("New / */"+str(parent_type)+"/"+parent_name+"/Sensor
     "+sensor_name +'\n').encode())
224         time.sleep(0.1)
225         check=s.recv(BUFFER_SIZE)
226         if str(check.decode("utf-8")) != 'ACK':
227             data.append('Create Pointing Sensor')
228         s.send(("Define */"+str(parent_type)+"/"+parent_name+"/
    Sensor/"+sensor_name+" "+shape+" "+dim+ '\n').encode())
229         time.sleep(0.1)
230         check1=s.recv(BUFFER_SIZE)
231         if str(check1.decode("utf-8")) != 'ACK':
232             data.append('Define Pointing Sensor')
233         s.send(("Point */"+str(parent_type)+"/"+parent_name+"/Sensor
    /"+sensor_name +" Targeted Tracking "+targtype+"/"+targname+"
    Rotate" + '\n').encode())
234         time.sleep(0.1)
235         check2=s.recv(BUFFER_SIZE)
236         if str(check2.decode("utf-8")) != 'ACK':
237             data.append('Assign Target to Pointing Snesor')
238
239 def AddTarget(s, data,parent_type,parent_name,sensor_name,targettype
    ,targetname):
240     """Adds any object to the list of targeted objects of a given
    pointing sensor"""
241     if type(parent_name) != str or ' ' in parent_name or type(
    sensor_name) != str or ' ' in sensor_name or type(targettype) !=
    str or ' ' in targettype or type(targetname) != str or ' ' in
    targetname or type(parent_type) != str or ' ' in parent_type:
242         raise TypeError('Add Taregt to Sensor: The parent plaform
```

```
    type, name, sensor name, target type and target name must be
    strings with no spaces.')
243     else:
244         s.send(("Point */"+str(parent_type)+"/"+parent_name+"/Sensor
    /"+sensor_name +" Targeted Add "+targettype+"/"+targetname+"
    Rotate" + '\n').encode())
245         time.sleep(0.1)
246         check=s.recv(BUFFER_SIZE)
247         if str(check.decode("utf-8")) != 'ACK':
248             data.append('Assign Additional Target to Pointing Sensor
     List')
249
250 #%%
251 #Missile
252
253 def NewMissile(s, data,name,launchtime,launchlat,launchlong,TOF,
    Implat,Implong,ImpRad):
254     """Creates a missile object and propogates the missiles flight
    based on the given parameters; Launch time, lat, and long, Time
    of Flight; Impact lat, long and radius(in meters)"""
255     if type(name) != str or ' ' in name or type(launchtime) != str
    or type(launchlat) != str or type(launchlong) != str or type(TOF)
     != str or type(Implat) != str or type(Implong) != str or type(
    ImpRad) != str:
256         raise TypeError('New Missile: The missile name must be
    strings with no spaces. All other inputs must be strings')
257     else:
258         s.send(('New / */Missile/ ' + name + '\n').encode())
259         time.sleep(0.1)
260         check=s.recv(BUFFER_SIZE)
261         if str(check.decode("utf-8")) != 'ACK':
262             data.append('Create Missile')
```

```
263         s.send(('Missile */Missile/'+name+' Trajectory "'+launchtime
    +'" 60.0 LnLatGeod '+launchlat+' '+launchlong+' 0.0 TOF '+TOF+'
    ImLatGeoc '+Implat+' '+Implong+' '+ImpRad+'\n').encode())
264         time.sleep(0.1)
265         check1=s.recv(BUFFER_SIZE)
266         if str(check1.decode("utf-8")) != 'ACK':
267             data.append('Set Missile Parameters')
268 
269 #%%
270 #Report Creation
271 
272 def CreateReports(s, data, satname, fileloc, title, numplane, numsat
    , style, targettype, targetname):
273     """Creates both Access and AER reports based given inputs"""
274     if type(satname) != str or ' ' in satname or type(title) != str
    or ' ' in title or type(fileloc) != str or ' ' in fileloc or type
    (numplane) != str or type(numsat) != str or type(style) != str or
     type(targettype) != str or ' ' in targettype or type(targetname)
     != str or ' ' in targetname:
275         raise TypeError('Create Reports: The sat name, file location
    , title, number of planes, number of satellites, style, target
    type and target name must be strings with no spaces.')
276     else:
277         for x in range(1,int(numplane)+1):
278             for y in range(1,int(numsat)+1):
279                 s.send(('ReportCreate */Satellite/'+satname + str(x)
     + str(y) +' Type Export Style '+style+' File \"'+fileloc+'\\\\'+
    title + str(x) + str(y) + '.txt\" AccessObject */'+targettype+'/'
     + targetname +'\n').encode())
280                 time.sleep(0.1)
281                 check=s.recv(BUFFER_SIZE)
282                 if str(check.decode("utf-8")) != 'ACK':
```

```
283                     data.append('Create Report')
284
285 #%%
286 #Save
287 def SaveScenario(s, data,fileloc,filename):
288     '''Saves the scenario to a designated lcoation'''
289     if type(fileloc) != str or ' ' in fileloc or type(filename) !=
    str or ' ' in filename:
290         raise TypeError('Save Scenario: The file location and file
    name must be strings with no spaces')
291     else:
292         s.send(('SaveAs / * \"'+fileloc+'\\\\'+filename+'"\n').
    encode())
293         time.sleep(0.1)
294         check=s.recv(BUFFER_SIZE)
295         if str(check.decode("utf-8")) != 'ACK':
296             data.append('Save Scenario')
297
298 #%%
299 #Export Ephemeris
300 def Export_Ephemeris(s, data,fileloc,sat_name):
301     if type(fileloc) != str or ' ' in fileloc or type(sat_name) !=
    str or ' ' in sat_name:
302         raise TypeError('Export Emphemris: The file location and
    satellite name must be strings with no spaces')
303     else:
304         s.send(('ExportDataFile */Satellite/'+str(sat_name)+'
    Ephemeris "'+fileloc+'\\'+str(sat_name)+'.e" Type STK CoordSys
    Inertial CentralBody Earth InterpBoundaries Include TimeSteps
    UseEphemerisSteps TimePeriod UseScenarioInterval' + '\n').encode
    ())
305         time.sleep(0.1)
```

```
306         check=s.recv(BUFFER_SIZE)
307         if str(check.decode("utf-8")) != 'ACK':
308             data.append('Export Ephemris')
309 
310 #%%
311 #Astrogator Commands Here
312 
313 def Astro_Create(s, data,sat_name):
314     if type(sat_name) != str or ' ' in sat_name:
315         raise TypeError('Astro Create: Satellite Name must be a
    string with no spaces')
316     else:
317         s.send(('New / */Satellite ' + str(sat_name) + '\n').encode
    ())
318         time.sleep(0.1)
319         check=s.recv(BUFFER_SIZE)
320         if str(check.decode("utf-8")) != 'ACK':
321             data.append('Astro Create Satellite')
322         s.send(("Astrogator */Satellite/"+str(sat_name)+"
    DeleteSegment Initial_State" +'\n').encode())
323         time.sleep(0.1)
324         check1=s.recv(BUFFER_SIZE)
325         if str(check1.decode("utf-8")) != 'ACK':
326             data.append('Astro Clear Segment')
327         s.send(("Astrogator */Satellite/"+str(sat_name)+"
    DeleteSegment Propagate" +'\n').encode())
328         time.sleep(0.1)
329         check2=s.recv(BUFFER_SIZE)
330         if str(check2.decode("utf-8")) != 'ACK':
331             data.append('Astro Clear Segment')
332 
333 def Astro_Create_Sequence(s, data,sat_name,seq_type,*man_type):
```

```
334     if type(sat_name) != str or ' ' in sat_name or type(seq_type) !=
        str or ' ' in seq_type:
335         raise TypeError('Astro Create Sequence: Satellite Name and
    Sequence Type must be strings with no spaces')
336     else:
337         s.send(("Astrogator */Satellite/"+str(sat_name)+"
    InsertSegment MainSequence.SegmentList.- "+str(seq_type) + '\n').
    encode())
338         time.sleep(0.1)
339         check=s.recv(BUFFER_SIZE)
340         if str(check.decode("utf-8")) != 'ACK':
341             data.append('Astro Create Sequence')
342         for man in man_type:
343             if type(man) != str or ' ' in man:
344                 raise TypeError('Astro Create Sequence: Manuever
    Types must be strings with no spaces')
345             else:
346                 s.send(("Astrogator */Satellite/"+str(sat_name)+"
    InsertSegment MainSequence.SegmentList."+str(seq_type)+".
    SegmentList.- "+str(man) + '\n').encode())
347                 time.sleep(0.1)
348                 check1=s.recv(BUFFER_SIZE)
349                 if str(check1.decode("utf-8")) != 'ACK':
350                     data.append('Astro Sequence Create Segment List'
    )
351
352 def Astro_Launch_Details(s, data,sat_name,tgt_seq,StepSize,Epoch,Alt
    ):
353     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq or type(StepSize) != str or type(Epoch) !=
    str or type(Alt) != str:
354         raise TypeError('Astro Launch Details: Satellite Name and
```

```
    Target Sequence must be strings with no spaces. Step Size, Epoch,
     and Altitude (in km) must be strings.')
355  else:
356      s.send(("Astrogator */Satellite/"+str(sat_name)+" SetValue
    MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.Launch.
    StepSize "+str(StepSize)+" sec" + '\n').encode())
357      time.sleep(0.1)
358      check=s.recv(BUFFER_SIZE)
359      if str(check.decode("utf-8")) != 'ACK':
360          data.append('Astro Launch Step Size')
361      s.send(("Astrogator */Satellite/"+str(sat_name)+" SetValue
    MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.Launch.
    Launch.Epoch " + str(Epoch)+ " UTCG" + '\n').encode())
362      time.sleep(0.1)
363      check1=s.recv(BUFFER_SIZE)
364      if str(check1.decode("utf-8")) != 'ACK':
365          data.append('Astro Launch Epoch')
366      s.send(("Astrogator */Satellite/"+str(sat_name)+" SetValue
    MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.Launch.
    Burnout.DisplaySystem Launch Az / Alt" + '\n').encode())
367      time.sleep(0.1)
368      check2=s.recv(BUFFER_SIZE)
369      if str(check2.decode("utf-8")) != 'ACK':
370          data.append('Astro Launch Burnout Az/Alt')
371      s.send(("Astrogator */Satellite/"+str(sat_name)+" SetValue
    MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.Launch.
    Burnout.LaunchAzDRDAlt.Altitude "+str(Alt)+" km" + '\n').encode()
    )
372      time.sleep(0.1)
373      check3=s.recv(BUFFER_SIZE)
374      if str(check3.decode("utf-8")) != 'ACK':
375          data.append('Astro Launch Alt')
```

```
376         s.send(("Astrogator */Satellite/"+str(sat_name)+" SetValue
    MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.Launch.
    Burnout.BurnoutOptions Use Fixed Velocity" + '\n').encode())
377         time.sleep(0.1)
378         check4=s.recv(BUFFER_SIZE)
379         if str(check4.decode("utf-8")) != 'ACK':
380             data.append('Astro Launch Fixed Velocity')
381
382 def Astro_Propagate_Details(s, data,sat_name,tgt_seq,prop_seq,
    condition_type,*condition_detail):
383     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq or type(prop_seq) != str or ' ' in prop_seq
     or type(condition_type) != str:
384         raise TypeError('Astro Propagate Details: Satellite Name,
    Target Sequence, and Propagate Segment must be strings with no
    spaces. Condition must be a string.')
385     else:
386         if condition_type != 'Duration':
387             s.send(("Astrogator */Satellite/"+str(sat_name)+"
    SetValue MainSequence.SegmentList."+str(tgt_seq)+".SegmentList."+
    str(prop_seq)+".StoppingConditions.Duration.Active false" + '\n')
    .encode())
388             time.sleep(0.1)
389             check=s.recv(BUFFER_SIZE)
390             if str(check.decode("utf-8")) != 'ACK':
391                 data.append('Astro Propagate Duration False')
392             s.send(("Astrogator */Satellite/"+str(sat_name)+"
    SetValue MainSequence.SegmentList."+str(tgt_seq)+".SegmentList."+
    str(prop_seq)+".StoppingConditions "+str(condition_type)+ '\n').
    encode())
393             time.sleep(0.1)
394             check1=s.recv(BUFFER_SIZE)
```

```
395             if str(check1.decode("utf-8")) != 'ACK':
396                 data.append('Astro Propagate Set Stopping Condition'
    )
397             s.send(("Astrogator */Satellite/"+str(sat_name)+"
    SetValue MainSequence.SegmentList."+str(tgt_seq)+".SegmentList."+
    str(prop_seq)+".StoppingConditions."+str(condition_type)+".Active
     true"+ '\n').encode())
398             time.sleep(0.1)
399             check2=s.recv(BUFFER_SIZE)
400             if str(check2.decode("utf-8")) != 'ACK':
401                 data.append('Astro Propagate Stopping Condition
    Active')
402         else:
403             s.send(("Astrogator */Satellite/"+str(sat_name)+"
    SetValue MainSequence.SegmentList."+str(tgt_seq)+".SegmentList."+
    str(prop_seq)+".StoppingConditions.Duration.Active true" + '\n').
    encode())
404             time.sleep(0.1)
405             check3=s.recv(BUFFER_SIZE)
406             if str(check3.decode("utf-8")) != 'ACK':
407                 data.append('Astro Propagate Duration True')
408             for c in condition_detail:
409                 if type(c) != str:
410                     raise TypeError('Astro Propagate Details:
    Duration must be input as a String with the appropriate units')
411                 else:
412                     s.send(("Astrogator */Satellite/"+str(sat_name)+
    " SetValue MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.
    "+str(prop_seq)+".StoppingConditions.Duration.TripValue "+str(c)
    + '\n').encode())
413                     time.sleep(0.1)
414                     check4=s.recv(BUFFER_SIZE)
```

```
415                     if str(check4.decode("utf-8")) != 'ACK':
416                         data.append('Astro Propagate Duration Time
    Value')
417 
418 def Astro_Maneuver_Thrust_Axes(s, data,sat_name,tgt_seq,
    attitude_control,thrust_axes):
419     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq or type(attitude_control) != str or type(
    thrust_axes) != str:
420         raise TypeError('Astro Maneuver Thrust Axes: Satellite Name
    and Target Sequence must be strings with no spaces. Attitude
    Control and Thrust Axes must be strings.')
421     else:
422         s.send(("Astrogator */Satellite/"+str(sat_name)+" SetValue
    MainSequence.SegmentList."+str(tgt_seq)+".SegmentList.Maneuver.
    ImpulsiveMnvr.AttitudeControl "+str(attitude_control) + '\n').
    encode())
423         time.sleep(0.1)
424         check=s.recv(BUFFER_SIZE)
425         if str(check.decode("utf-8")) != 'ACK':
426             data.append('Astro Attitude COntrol')
427         s.send(('Astrogator */Satellite/'+str(sat_name)+' SetValue
    MainSequence.SegmentList.'+str(tgt_seq)+'.SegmentList.Maneuver.
    ImpulsiveMnvr.ThrustAxes "'+str(thrust_axes)+'"' + '\n').encode()
    )
428         time.sleep(0.1)
429         check1=s.recv(BUFFER_SIZE)
430         if str(check1.decode("utf-8")) != 'ACK':
431             data.append('Astro Maneuver Thrust Axes')
432 
433 def Astro_Maneuver_Cartesian(s, data,sat_name,tgt_seq,cart,*cart_det
    ):
```

```
434     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq or type(cart) != str:
435         raise TypeError('Astro Maneuver Cartesian: Satellite Name
    and Target Sequence must be strings with no spaces. Cartesian
    axis must be a string.')
436     else:
437         for c in cart_det:
438             if type(c) != str:
439                     raise TypeError('Astro Maneuver Cartesian:
    Cartesian Detail must be input as a string')
440             else:
441                 s.send(('Astrogator */Satellite/'+str(sat_name)+'
    SetValue MainSequence.SegmentList.'+str(tgt_seq)+'.SegmentList.
    Maneuver.ImpulsiveMnvr.Cartesian.'+str(cart)+' '+str(c) + '\n').
    encode())
442                 time.sleep(0.1)
443                 check=s.recv(BUFFER_SIZE)
444                 if str(check.decode("utf-8")) != 'ACK':
445                     data.append('Astro Maneuver Cartesian Value')
446         s.send(('Astrogator */Satellite/'+str(sat_name)+'
    AddMCSSegmentControl MainSequence.SegmentList.'+str(tgt_seq)+'.
    SegmentList.Maneuver ImpulsiveMnvr.Cartesian.'+str(cart) + '\n').
    encode())
447         time.sleep(0.1)
448         check1=s.recv(BUFFER_SIZE)
449         if str(check1.decode("utf-8")) != 'ACK':
450             data.append('Astro Maneuver Cartesian Variable')
451
452 def Astro_Results(s, data,sat_name,tgt_seq,prop_seq,result_type):
453     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq or type(prop_seq) != str or ' ' in prop_seq
     or type(result_type) != str:
```

```
454         raise TypeError('Astro Result: Satellite Name,Target
    Sequence, and Propagate Segment must be strings with no spaces.
    Result Type must be input as a string')
455     else:
456         s.send(('Astrogator */Satellite/'+str(sat_name)+' SetValue
    MainSequence.SegmentList.'+str(tgt_seq)+'.SegmentList.'+str(
    prop_seq)+'.Results '+str(result_type)+'\n').encode())
457         time.sleep(0.1)
458         check=s.recv(BUFFER_SIZE)
459         if str(check.decode("utf-8")) != 'ACK':
460             data.append('Astro Result')
461 
462 def Astro_Sequence_Update(s, data,sat_name,tgt_seq):
463     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq:
464         raise TypeError('Astro Sequence Update: Satellite Name and
    Target Sequence must be strings with no spaces')
465     else:
466         s.send(('Astrogator */Satellite/'+str(sat_name)+' SetValue
    MainSequence.SegmentList.'+str(tgt_seq)+'.Action Run active
    profiles' + '\n').encode())
467         time.sleep(0.1)
468         check=s.recv(BUFFER_SIZE)
469         if str(check.decode("utf-8")) != 'ACK':
470             data.append('Astro Sequence Run Action')
471         s.send(('Astrogator */Satellite/'+str(sat_name)+' SetValue
    AutomaticallyAddIndependentVariablesToDifferentialCorrectors true
    ' + '\n').encode())
472         time.sleep(0.1)
473         check1=s.recv(BUFFER_SIZE)
474         if str(check1.decode("utf-8")) != 'ACK':
475             data.append('Astro Sequence Independent Differential
```

```
    Correctors')
476         s.send(('Astrogator */Satellite/'+str(sat_name)+' SetValue
    AutomaticallyAddDependentVariablesToDifferentialCorrectors true'
    + '\n').encode())
477         time.sleep(0.1)
478         check2=s.recv(BUFFER_SIZE)
479         if str(check2.decode("utf-8")) != 'ACK':
480             data.append('Astro Sequence Dependent Differential
    Correctors')
481
482 def Astro_Set_Constraint(s, data,sat_name,tgt_seq,man,result,D_or_T,
    val):
483     if type(sat_name) != str or ' ' in sat_name or type(tgt_seq) !=
    str or ' ' in tgt_seq or type(man) != str or ' ' in man or type(
    D_or_T) != str or ' ' in D_or_T or type(val) != str:
484         raise TypeError('Astro Set Constraint: Satellite Name,Target
     Sequence, Maneuver, Result, and Desire/Tolerance must be strings
     with no spaces. Value must be input as a string')
485     else:
486         s.send(('Astrogator */Satellite/'+str(sat_name)+'
    SetMCSConstraintValue MainSequence.SegmentList.'+str(tgt_seq)+'.
    Profiles.Differential_Corrector '+str(man)+' '+str(result)+' '+
    str(D_or_T)+' '+str(val)+ '\n').encode())
487         time.sleep(0.1)
488         check=s.recv(BUFFER_SIZE)
489         if str(check.decode("utf-8")) != 'ACK':
490             data.append('Astro Set Constraints')
491
492 def Astro_Insert_Sequence(s, data,sat_name,seq_type,seq_num,*
    man_type):
493     if type(sat_name) != str or ' ' in sat_name or type(seq_type) !=
     str or ' ' in seq_type or type(seq_num) != str or ' ' in seq_num
```

```
    :
494         raise TypeError('Astro Insert Sequence: Satellite Name,
    Sequence Type, and Sequence Number must be strings with no spaces
    ')
495     else:
496         s.send(("Astrogator */Satellite/"+str(sat_name)+"
    InsertSegment MainSequence.SegmentList.- "+str(seq_type) + '\n').
    encode())
497         time.sleep(0.1)
498         check=s.recv(BUFFER_SIZE)
499         if str(check.decode("utf-8")) != 'ACK':
500             data.append('Astro Insert Sequence')
501         for man in man_type:
502             if type(man) != str or ' ' in man:
503                 raise TypeError('Astro Insert Sequence: Manuever
    Types must be strings with no spaces')
504             else:
505                 s.send(("Astrogator */Satellite/"+str(sat_name)+"
    InsertSegment MainSequence.SegmentList."+str(seq_type)+str(
    seq_num)+".SegmentList.- "+str(man) + '\n').encode())
506                 time.sleep(0.1)
507                 check1=s.recv(BUFFER_SIZE)
508                 if str(check1.decode("utf-8")) != 'ACK':
509                     data.append('Astro Insert Sequence Segment List'
    )
510
511 def Astro_Run(s, data,sat_name):
512     if type(sat_name) != str or ' ' in sat_name:
513         raise TypeError('Astro Run: Satellite Name must be a string
    with no spaces')
514     else:
515         s.send(('Astrogator */Satellite/'+str(sat_name)+' RunMCS' +
```

```
    '\n').encode())
516         time.sleep(0.1)
517         check=s.recv(BUFFER_SIZE)
518         if str(check.decode("utf-8")) != 'ACK':
519             data.append('Astro Run')
520 
521 #%%
522 # Add GPS from almanac
523 
524 def AddGPS(s, data, sat_ID, startdate, starttime, enddate, endtime):
525     """Propogates any number of GPS satellites from almanac"""
526     if type(startdate) != str:
527         raise TypeError('Scenario Time: Start Date needs to be a
    string in the form of Day Month Year')
528     elif type(starttime) != str:
529         raise TypeError('Scenario Time: Start Time needs to be a
    string in the form of HH:MM:SS.SS')
530     elif type(enddate) != str:
531         raise TypeError('Scenario Time: End Date needs to be a
    string in the form of Day Month Year')
532     elif type(endtime) != str:
533         raise TypeError('Scenario Time: End Time needs to be a
    string in the form of HH:MM:SS.SS')
534     elif type(sat_ID) != str:
535         raise TypeError('sat_ID: Must be a string in the form of \"
    ID ## ID ## ID ##\" or \"ID All\"')
536     else:
537         s.send(('ImportAlmanacFile * "C:\ProgramData\AGI\STK 11 (x64
    )\GPSAlmanacs\GPSAlmanac.al3" ' + sat_ID + ' StartStop \"' +
    startdate + ' ' + starttime + '\" \"' + enddate + ' ' + endtime +
     '\" WeekRefEpoch 22Aug1999' + '\n').encode())
538         time.sleep(0.1)
```

```
539         check=s.recv(BUFFER_SIZE)
540         if str(check.decode("utf-8")) != 'ACK':
541             data.append('AddGPS')
542 
543 #%%
544 # Add Aircraft with start, stop, altitude, speed
545 
546 def CreateAircraft(s, data, ac_name, startdate, starttime, startlat,
        startlon, endlat, endlon, alt, speed):
547     """Propogates any number of GPS satellites from almanac"""
548     if type(startdate) != str:
549         raise TypeError('Scenario Time: Start Date needs to be a
        string in the form of Day Month Year')
550     elif type(starttime) != str:
551         raise TypeError('Scenario Time: Start Time needs to be a
        string in the form of HH:MM:SS.SS')
552     elif type(ac_name) != str:
553         raise TypeError('sat_ID: Must be a string with no spaces')
554     else:
555         s.send(('New / */Aircraft ' + ac_name + '\n').encode())
556         time.sleep(0.1)
557         check=s.recv(BUFFER_SIZE)
558         if str(check.decode("utf-8")) != 'ACK':
559             data.append('AddAircraft')
560         s.send(('AddWaypoint */Aircraft/' + ac_name + '
        DetTimeAccFromVel ' + startlat + ' ' + startlon + ' ' + alt + ' '
        + speed + '\n').encode())
561         #time.sleep(0.1)
562         check=s.recv(BUFFER_SIZE)
563         if str(check.decode("utf-8")) != 'ACK':
564             data.append('AddFirstWaypoint')
565         s.send(('AddWaypoint */Aircraft/' + ac_name + '
```

```
    DetTimeAccFromVel ' + endlat + ' ' + endlon + ' ' + alt + ' ' +
    speed + '\n').encode())
566         #time.sleep(0.1)
567         check=s.recv(BUFFER_SIZE)
568         if str(check.decode("utf-8")) != 'ACK':
569             data.append('AddSecondWaypoint')
570         s.send(('SetGreatArcStart */Aircraft/' + ac_name + ' \"' +
    startdate + ' ' + starttime + '\"\n').encode())
571         #time.sleep(0.1)
572         check=s.recv(BUFFER_SIZE)
573         if str(check.decode("utf-8")) != 'ACK':
574             data.append('AddGreatArc')
575
576 #%%
577 #Generic Report Creation
578
579 def CreateGenReports(s, data, obj1, obj2, fileloc, title, style):
580     """Creates both Access and AER reports based given inputs"""
581     s.send(('ReportCreate */Aircraft/' + 'AC_0001_00001' + ' Type
    Export Style ' + 'AER' + ' File \"I:\Academic_Quarters\Thesis\
    Python\\' + 'TestReport' + '\\' + 'TestReport' + '.txt\"
    AccessObject */' + 'Facility/F_0001_001/Sensor/S_0001_001_01' + '
    /' + '' +'\n').encode() )
582     s.send(('ReportCreate */' + obj1 + ' Type Export Style ' + style
     + ' File \"' + fileloc + '\\' + title + '.txt\" AccessObject */'
     + obj2 + '/' + '' +'\n').encode() )
583     time.sleep(0.001)
584     check=s.recv(BUFFER_SIZE)
585     if str(check.decode("utf-8")) != 'ACK':
586         data.append('Create Report')
```

Bibliography

1. The global positioning system. `https://www.gps.gov/`. Accessed: 2019-03-01.

2. Jarmo T Alander. On optimal population size of genetic algorithms. In *CompEuro 1992 Proceedings computer systems and software engineering*, pages 65–70. IEEE, 1992.

3. Michael P. Atkinson, Moshe Kress, and Roberto Szechtman. Maritime transportation of illegal drugs from South America. *International Journal of Drug Policy*, 39:43–51, 2017.

4. author=Whitley, Darrell and Starkweather, Timothy. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In *International Conference on Genetic Algorithms*, volume 89, pages 116–123. Fairfax, VA, 1989.

5. Amir Azemati, Mahta Moghaddam, and Arvind Bhat. Relationship between bistatic radar scattering cross sections and GPS reflectometry delay-Doppler maps over vegetated land in support of soil moisture retrieval. *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2018-July(1):7480–7482, 2018.

6. David Barton. *Radar system analysis and modeling*. Artech House, Boston, MA, 2005.

7. Vera Behar and Christo Kabakchiev. Detectability of air targets using bistatic radar based on GPS L5 signals. *Radar Symposium (IRS), 2011 Proceedings International*, (January 2011):212–217, 2011.

8. Fabrizio Berizzi, Marco Martorella, and Elisa Giusti. *Radar imaging for maritime observation*. CRC, Boca Raton, FL, 2018.

9. Tellis A. Bethel. Caribbean narcotics trafficking: what is to be done. *DISAM journal*, pages 80–90, 2003.

10. Leonard Bolc. *Search methods for artificial intelligence*. Academic Press, London, 1992.

11. Eric Bonabeau. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, New York, NY, 1999.

12. S.L. Caldwell. GAO-14-527, Coast Guard: Resources Provided for Drug Interdiction Operations in the Transit Zone, Puerto Rico, and the U.S. Virgin Islands. (June), 2014.

13. Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371, 2000.

14. Hugo Carreno-Luengo, Guido Luzi, and Michele Crosetto. Sensitivity of CyGNSS Bistatic Reflectivity and SMAP Microwave Radiometry Brightness Temperature to Geophysical Parameters over Land Surfaces. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(1):107–122, 2019.

15. Raymond Chiong. *Variants of evolutionary algorithms for real-world applications*. Springer, Berlin, NY, 2012.

16. David Coley. *An introduction to genetic algorithms for scientists and engineers*. World Scientific, Singapore River Edge, NJ, 1999.

17. F. Daout, F. Schmitt, G. Ginolhac, and P. Fargette. Multistatic and multiple frequency imaging resolution analysis application to GPS-based multistatic radar. *IEEE Transactions on Aerospace and Electronic Systems*, 48(4):3042–3057, 2012.

18. Mark E. Davis. *Advances in Bistatic Radar*. SciTech Publishing, Raleigh, NC, 2013.

19. James Evans. *Optimization algorithms for networks and graphs*. Marcel Dekker, Inc, New York, NY, 1992.

20. Marina Gashinova, Liam Daniel, Edward Hoare, Vladimir Sizov, Kalin Kabakchiev, and Mikhail Cherniakov. Signal characterisation and processing in the forward scatter mode of bistatic passive coherent location. *Eurasip Journal on Advances in Signal Processing*, 2013(1):1–14, 2013.

21. Scott Gleason, Stephen Hodgart, Yiping Sun, Christine Gommenginger, Stephen Mackin, Mounir Adjrad, and Martin Unwin. Detection and processing of bistatically reflected GPS signals from low earth orbit for the purpose of ocean remote sensing. *IEEE Transactions on Geoscience and Remote Sensing*, 43(6):1229–1241, 2005.

22. Global Positioning Systems Directorate. NAVSTAR GPS Space Segment/Navigation User Segment Interfaces, 2018.

23. Global Positioning Systems Directorate. NAVSTAR GPS Space Segment/User Segment L5 Interfaces, 2018.

24. Teofilo Gonzalez. *Handbook of approximation algorithms and metaheuristics*. Chapman & Hall/CRC, Boca Raton, 2007.

25. Griffiths, Hugh D. and Christopher J. Baker. *An introduction to passive radar*. Artech House, Norwood, MA, 2017.

26. US Coast Guard. Coast Guard Publication 1, 2014.

27. Randy Haupt. *Practical genetic algorithms*. Wiley, New York, NY, 1998.

28. Eugene Hecht. *Optics, 4th ed.* Addison-Wesley, Reading, MA, 2002.

29. Hitoshi Iba. *New frontier in evolutionary algorithms : theory and applications*. Imperial College Press, London, 2012.

30. Michael Inggs. Passive coherent location as cognitive radar. *IEEE Aerospace and Electronic Systems Magazine*, 25(5):12–17, 2010.

31. Sean A. Kaiser, Andrew J. Christianson, and Ram M. Narayanan. Global positioning system processing methods for GPS passive coherent location. *IET Radar, Sonar & Navigation*, 11(9):1406–1416, 2017.

32. Hillol Kargupta. *Gene expression: The missing link in evolutionary computation*. Los Alamos National Lab., NM, 1997.

33. Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

34. Sanneke Kloppenburg. Mapping the Contours of Mobilities Regimes. Air Travel and Drug Smuggling Between the Caribbean and the Netherlands. *Mobilities*, 8(1):52–69, 2013.

35. Jing Li, Yongjun Zhao, and Donghai Li. Accurate single-observer passive coherent location estimation based on TDOA and DOA. *Chinese Journal of Aeronautics*, 27(4):913–923, 2014.

36. Dallas Masters, Penina Axelrad, and Stephen Katzberg. Initial results of land-reflected GPS bistatic radar measurements in SMEX02. *Remote Sensing of Environment*, 92(4):507–520, 2004.

37. Hamish Meikle. *Modern radar systems*. Artech House, Boston, MA, 2001.

38. Fred Nathanson. *Radar design principles : signal processing and the environment*. SciTech Pub, Raleigh, NC, 1999.

39. Office of National Drug Control Policy and United States of America. Cocaine smuggling in 2009. 2010.

40. United Nations Office on Drugs and Crime. *World drug report 2010*. United Nations Publications, 2010.

41. Debora Pastina, Fabrizio Santi, Federica Pieralice, Marta Bucciarelli, Hui Ma, Dimitrios Tzagkas, Michail Antoniou, and Mikhail Cherniakov. Maritime Moving Target Long Time Integration for GNSS-Based Passive Bistatic Radar. *IEEE Transactions on Aerospace and Electronic Systems*, 54(6):3060–3083, 2018.

42. Peyton Peebles. *Radar principles*. Wiley, New York, NY, 1998.

43. S. Powell, D. Akos, and V. Zavorotny. GPS SBAS L1/L5 bistatic radar altimeter. *International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 1544–1547, 2014.

44. Infrastructure Protection. Homeland Security Act of 2002, Public Law 107-296, 2002.

45. Chow Yii Pui and Matthew Trinkle. GPS bistatic radar using phased-array technique for aircraft detection. *2013 International Conference on Radar - Beyond Orthodoxy: New Paradigms in Radar, RADAR 2013*, pages 274–279, 2013.

46. Mojtaba Radmard, Mohammad Mahdi Nayebi, and Seyyed Mohammad Karbasi. Diversity-based geometry optimization in mimo passive coherent location. *Radioengineering*, 23(1):41–49, 2014.

47. Colin R Reeves. Using genetic algorithms with small populations. In *International Conference on Genetic Algorithms*, volume 590, page 92. Fairfax, VA, 1993.

48. Colin R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3):231–250, 1997.

49. M. A. Richards. *Fundamentals of radar signal processing*. McGraw-Hill Education, New York, NY, 2014.

50. Faozi Said, Zorana Jelenak, Paul S. Chang, and Seubson Soisuvarn. An Assessment of CYGNSS Normalized Bistatic Radar Cross Section Calibration. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(1):50–65, 2019.

51. S Sakhawat, M Usman, and A Hanif. Power budget analysis of reflected gps l1 and l5 frequency signals for passive microwave imaging. *Nucleus*, 51(1):87–92, 2014.

52. Nel Samama. *Global positioning: technologies and performance*. Wiley-Interscience, Hoboken, NJ, 2008.

53. Luca Scrucca. On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution. *arXiv preprint arXiv:1605.01931*, 2016.

54. Luca Scrucca et al. GA: a package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013.

55. El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, Hoboken, N.J, 2009.

56. Pyotr Ya Ufimtsev. *Fundamentals of the Physical Theory of Diffraction*. John Wiley & Sons, Hoboken, NY, 2006.

57. Sean Urban. *Explanatory supplement to the Astronomical almanac*. University Science Books, Mill Valley, Calif, 2013.

58. Alexander G. Voronovich and Valery U. Zavorotny. The transition from weak to strong diffuse radar bistatic scattering from rough ocean surface. *IEEE Transactions on Antennas and Propagation*, 65(11):6029–6034, 2017.

59. David Williamson. *The design of approximation algorithms*. Cambridge University Press, Cambridge, NY, 2011.

60. Nicholas J. Willis. *Bistatic Radar*. SciTech Publishing, Raleigh, NC, 2013.

61. Yang Xiao. *Bio-inspired computing and networking*. CRC Press, Boca Raton, FL, 2011.

62. Valery Zavorotny and Alexander Voronovich. GNSS-R Modeling Results Obtained with Improved Bistatic Radar Equation. *Proceedings of the 2018 20th International Conference on Electromagnetics in Advanced Applications, ICEAA 2018*, pages 35–38, 2018.

63. Heng Zhang, Jiangwen Tang, Robert Wang, Yunkai Deng, Wei Wang, and Ning Li. An accelerated backprojection algorithm for monostatic and bistatic SAR processing. *Remote Sensing*, 10(1), 2018.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	Sept 2018 — Mar 2020

4. TITLE AND SUBTITLE

Heuristic approaches for near-optimal placement of GPS-based multi-static radar receivers in American coastal waters

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hufstetler, Brandon J., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-20-M-154

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

Department of Operational Sciences
2950 Hobson Way
WPAFB OH 45433-7765
DSN 271-0690, COMM 937-255-3636
Email: brandon.hufstetler@afit.edu

10. SPONSOR/MONITOR'S ACRONYM(S)

AFIT

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

Narcotics smuggling across the Caribbean Sea is a growing concern for the United States Coast Guard. One vector for this illicit trafficking is via small aircraft. This thesis proposes a multi-static radar architecture using the Global Positioning System (GPS) constellation as a transmission source to detect these aircraft as they transit a detection fence. The system developed in this thesis relies on the forward-scatter phenomenon in which a radar shadow is cast by a target as it crosses in front of a transmitter, creating a measurable difference in the signal amplitude at the receiver. This thesis first develops a mathematical model parametrizing such a multi-static radar system. This model is then used to build a novel simulation, and output from the simulation is used as input in a vast set covering problem whose goal is both to determine the smallest number of sensors along with their locations in order to detect 100% of transiting aircraft, and to determine the near optimal location of a fixed number of sensors. The research proves the problem can be modeled, albeit at great computational expense. It further demonstrates that near optimal solutions can be generated with almost no computational expense using the geometric heuristic.

15. SUBJECT TERMS

Metaheuristic, Genetic Algorithm, Forward-Scatter Radar

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. Bruce A. Cox, AFIT/ENS
U	U	U	UU	158	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636; bruce.cox@afit.edu