Camera View Planning for Structure from Motion: Achieving Targeted Inspection Through More Intelligent View Planning Methods

Trent James Okeson
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Chemical Engineering Commons

BYU ScholarsArchive Citation

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Camera View Planning for Structure from Motion:
Achieving Targeted Inspection Through More Intelligent View Planning Methods

Trent James Okeson

A thesis submitted to the faculty of Brigham Young University in partial fulfillment of the requirements for the degree of Master of Science

John D. Hedengren, Chair
  Kevin W. Franke
  Stella D. Nickerson

Department of Chemical Engineering Brigham Young University

Copyright © 2018 Trent James Okeson All Rights Reserved
ABSTRACT

Camera View Planning for Structure from Motion: Achieving Targeted Inspection Through More Intelligent View Planning Methods

Trent James Okeson
Department of Chemical Engineering, BYU
Master of Science

Remote sensors and unmanned aerial vehicles (UAVs) have the potential to dramatically improve infrastructure health monitoring in terms of accuracy of the information and frequency of data collection. UAV automation has made significant progress but that automation is also creating vast amounts of data that needs to be processed into actionable information. A key aspect of this work is the optimization (not just automation) of data collection from UAVs for targeted planning of mission objectives. This work investigates the use of camera planning for Structure from Motion for 3D modeling of infrastructure. Included in this thesis is a novel multi-scale view-planning algorithm for autonomous targeted inspection. The method presented reduced the number of photos needed and therefore reduced the processing time while maintaining desired accuracies across the test site. A second focus in this work investigates various set covering problem algorithms to use for selecting the optimal camera set. The trade-offs between solve time and quality of results are explored. The Carousel Greedy algorithm is found to be the best method for solving the problem due to its relatively fast solve speeds and the high quality of the solutions found. Finally, physical flight tests are used to demonstrate the quality of the method for determining coverage. Each of the set covering problem algorithms are used to create a camera set that achieves 95% coverage. The models from the different camera sets are comparable despite having a large amount of variability in the camera sets chosen. While this study focuses on multi-scale view planning for optical sensors, the methods could be extended to other remote sensors, such as aerial LiDAR.

Keywords: combinatorial optimization, camera planning, set covering problem, structure-from-motion, multi-scale modeling
ACKNOWLEDGMENTS

I would like to start by acknowledging the financial support and industrial perspective provided by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation Industry/University Cooperative Research Center (I/UCRC) under NSF Award No. IIP-1650547. Their support and input on the research helped focus the research on industrially relevant problems.

I would also like to express my appreciation for my advisor, Dr. John Hedengren. He has been very supportive of my work and goals, allowing me to shape the research to fit my interests. I would also like to express my appreciation for my committee, Dr. Kevin Franke and Dr. Stella Nickerson. Their feedback and input about my research and writing improved the quality of the research included in this thesis. I would also like to acknowledge the other students who have significantly contributed to this project. First, I would like to acknowledge Abe Martin for laying the groundwork for much of this work. I would also like to acknowledge Landen Blackburn, Joshua Pulsipher, Ben Horne, Cory Vernon, Matthew Gibbitas, and Sam Thorpe for their assistance coding up portions of this project. I would like to express my appreciation to the Civil Engineering graduate student on the project, Benjamin Barrett. It has been a pleasure to work with him over the past two years and much of the work included in this thesis has been impacted by his feedback and suggestions. I would also like to express my appreciation for the PRISM lab. I have learned from and gotten solution ideas from conversations with many of the students in the lab and appreciate the input I have received from them.

I would like to acknowledge my parents, who have supported me through my years of schooling. Their faith in me has helped me to achieve more than I could have on my own.
# TABLE OF CONTENTS

**LIST OF TABLES**  ............................................. vi

**LIST OF FIGURES** ........................................... vii

**Chapter 1  Introduction** ................................ 1

1.1 Literature Review ....................................... 2

1.1.1 Camera Planning ....................................... 3

1.1.2 Iterative Modeling .................................... 5

1.1.3 Multi-Scale Modeling ................................. 5

1.1.4 Early View Covering Research and Solutions .......... 6

1.1.5 Combinatorial Optimization .......................... 6

1.1.6 Set Covering Problem Solutions ...................... 7

1.1.7 Set Covering for View Planning ....................... 7

1.2 Summary of Novel Contributions ........................ 7

1.3 Thesis Outline ........................................... 8

**Chapter 2  Overall Problem Formulation** ............... 10

2.1 Problem Formulation .................................... 10

2.2 Overall SfM Camera Placement Algorithm Improvements .... 13

2.2.1 Vectorization: ........................................ 13

2.2.2 Sparse Matrices ....................................... 14

2.2.3 Improved Graphing .................................... 14

2.2.4 Google Maps API ....................................... 16

2.2.5 Mobile Application Development .................... 16

**Chapter 3  Multi-scale Modeling Extension** ............ 18

3.1 Introduction ............................................. 18

3.1.1 Iterative Modeling ................................... 18

3.2 Methods .................................................. 19

3.2.1 Platform .............................................. 20

3.2.2 Tibble Fork Dam Test Case ........................... 20

3.3 Results .................................................... 21

3.3.1 Flight Planning Results ............................... 21

3.3.2 Model Quality Results ................................. 24

3.4 Conclusions .............................................. 24

**Chapter 4  Set Covering Algorithms for SfM Camera Planning** .... 26

4.1 Introduction ............................................. 26

4.1.1 Combinatorial Optimization .......................... 27

4.2 Set Covering Problem Solutions ........................ 27

4.2.1 Greedy Heuristics .................................... 28

4.2.2 Linear Programing .................................... 29
4.2.3 Particle Swarm Optimization ............................................. 30
4.2.4 Simulated Annealing ....................................................... 30
4.2.5 Genetic Algorithms ....................................................... 30
4.2.6 Ant Colony Optimization .................................................. 31
4.2.7 Novel Contributions ....................................................... 31
4.3 Methods ............................................................................ 31
  4.3.1 Site Selection ............................................................... 32
  4.3.2 Site Characterization ..................................................... 32
  4.3.3 Greedy Heuristics ......................................................... 33
  4.3.4 Linear Programming ..................................................... 34
  4.3.5 Ant Colony Optimization ................................................ 35
  4.3.6 Particle Swarm Optimization ........................................... 36
  4.3.7 Genetic Algorithms ....................................................... 36
  4.3.8 Simulated Annealing ..................................................... 37
4.4 Results .............................................................................. 37
  4.4.1 Randomly Selected Sites ................................................ 37
  4.4.2 Handpicked Sites .......................................................... 43
4.5 Conclusions ................................................................. 47

Chapter 5  Physical Flight Tests ................................................. 49
  5.1 Introduction ...................................................................... 49
  5.2 Literature Review ........................................................... 49
  5.3 Methods .......................................................................... 50
  5.4 Results and Discussion ..................................................... 52
  5.5 Conclusions .................................................................... 57

Chapter 6  Conclusions ........................................................... 59
  6.1 Future Work ..................................................................... 60

REFERENCES ........................................................................ 63

Appendix A  Set Covering Solvers ............................................ 69

Appendix B  Physical Flight Test Models ................................... 84
# LIST OF TABLES

3.1 The number of cameras and the computational timing results for each of the optimized sections. .................................................. 22

3.2 Computational time and number of photos needed to model the entire TF Dam at varying priority levels. .................................................. 23

4.1 Site characterization values for three of the 20 hand selected sites. Plots for these three sites are shown in Figure 4.3 .................................................. 33

4.2 The linear fit parameters for the number of cameras and the size of the area to be modeled. 39

4.3 Two-sided p-value for the paired t-test on the average number of cameras needed. . . . 40

4.4 Fitting parameters with the 95% confidence interval for the linear fit relating number of cameras selected and the inverse of density. ............................ 42

4.5 Fitting parameters with the 95% confidence interval for the density fits. ....................... 46

5.1 Solve time and number of camera positions and poses chosen for the SCP for RCP test site. The approximate flight time to collect the photos is also included. ........... 51

5.2 Several of the camera specifications for the DJI Phantom 4’s camera. .......................... 51

5.3 Percent common cameras. ....................................................................... 53

5.4 The probability of randomly drawing at least as many in common between the two camera sets. ............................................................... 54
### LIST OF FIGURES

1.1 Visualization of how SfM works. The overlapping of the images is key. Images also need to be close enough together that common points between the photos appear similar. 3

2.1 Overall work flow for camera planning. 11
2.2 The different viewing angles for a single surface point. 12
2.3 The different categorizations of points for a single camera location and pose. 12
2.4 An example of a serial code and improved code utilizing Numpy’s vectorization to avoid serial computations. 14
2.5 The time to solve the set covering problem with the greedy algorithm based on the number of triangles in the mesh. 15
2.6 A example of the new plot output. The points are camera positions with lines showing their orientation. The red line is the optimized flight path. 15
2.7 A plot of Google Map’s elevation data against NED’s (USGS) elevation data when compared a georeferenced model (in green). 16

3.1 Overall work flow for multi-scale iterative camera planning. 20
3.2 Photo of the left half of the Tibble Fork Dam. 21
3.3 Models used for planning optimized flights. (a) Initially optimized camera locations and poses based on publicly available elevation data. (Elevation data in m). (b) Course subsampled model used for multi-scale modeling with priority points indicated. 22
3.4 Final optimized camera positions and poses for the multi-scale flight. 23
3.5 The final model processed in Agisoft from the multi-scale flight. 25

4.1 An example of using the coloring problem to solve the art gallery problem. 27
4.2 The sites chosen in the continental United States for testing. The points indicate the randomly selected locations and the stars represent the locations of the sites chosen by the author. 33
4.3 Elevation plots for three of the 20 hand selected sites. Site characterization information is included in Table 4.1. 34
4.4 Pareto front based on solve time and performance relative to the greedy algorithm for the randomly selected sites. 38
4.5 The number of cameras needed versus the area to be modeled for the randomly selected sites. 40
4.6 The number of cameras needed versus the inverse of the graph density. Linear fits were made for each method with a fixed intercept of zero. The slopes are given in the table. 41
4.7 Number of elements in to be covered (the graph’s size) with respect to the CBC termination condition. 43
4.8 The solve time based on the model area for the randomly selected sites. 44
4.9 The solve times according to site size and sorted by point density. 44
4.10 Pareto front based on solve time and performance relative to the greedy algorithm for the handpicked sites. 45
4.11 The number of cameras needed versus the area to be modeled for the handpicked sites. 46
4.12 The number of cameras needed versus the inverse of the graphs density. The fits are linear fits with a forced intercept of zero. .................................................. 47

5.1 Visual of the workflow for finding holes in a model. .............................................. 52
5.2 Point cloud for the model using the RG algorithm’s selected cameras. ................. 54
5.3 Models for three of the selection methods including the visualization of the holes (shown as red points) in the models ................................................................. 55
5.4 CDF plots of the distances from the sampled points to the original point cloud. ...... 56
5.5 A top down view of the overlap for several of the selected models. .................... 57
5.6 Illustrative example of the problem that buckets create for accurately determining if a set of cameras are sufficient to model a point .................................................. 58

B.1 Greedy .............................................................................................................. 84
B.2 CG ..................................................................................................................... 85
B.3 RG ...................................................................................................................... 85
B.4 CBC .................................................................................................................... 86
B.5 ACO .................................................................................................................... 86
B.6 GA ...................................................................................................................... 87
B.7 PS ...................................................................................................................... 87
B.8 SA ...................................................................................................................... 88
CHAPTER 1. INTRODUCTION

Infrastructure monitoring is a significant challenge for businesses and government agencies. With aging equipment and infrastructure, regular monitoring is important to ensure safety of existing structures. Recently there has been significant interest in using unmanned aerial vehicles (UAVs) to perform infrastructure monitoring inspections. UAVs have grown in popularity due to platform versatility, allowing for a wide variety of applications. UAVs can enter areas that are dangerous for people, perform routine surveillance, and complete difficult inspection tasks. Auxiliary sensors, including multi-spectral/thermal cameras, light detection and ranging (LiDAR), and chemical snifters, can be equipped to increase the functionality of a UAV. The wide variety of data collected can assist in the analysis of structural integrity.

Of the many different inspection methods used, an emerging method for UAV based infrastructure monitoring is to generate 3D models using structure from motion (SfM) [1, 2]. SfM uses photographs to make 3D models of objects of interest. This is done by using multiple photos of an object to triangulate its location. Once the model is created, it can be manually inspected or computer software can be used to compare to previous models or to detect changes. SfM is popular because of its portability, ease of data collection, and low cost [3].

Initially, UAVs were manually piloted to collect photos for model creation. This made UAV applications possible, but had several shortcomings. Human pilots are prone to make mistakes when flying, resulting in damage to capital equipment. Another shortcoming is the lack of repeatability of the models. For regular inspection, repeatable results are needed to improve comparisons between two models. In the best cases, manually piloted flights result in inconsistent photo locations and poses leading to inconsistent models [4].

The potential gains from automated UAV inspections over manually piloted inspections has sparked significant research in recent years [5]. One of the main research areas focuses on how to fly autonomous UAVs, leading to improved autopilots. Although UAVs are still unable to fly
reliably in some conditions (e.g., GPS denied environments), there has been significant research towards possible solutions [6]. For this research, automation for path following is treated as a solved problem.

A second, less studied, area of research focuses on where to fly instead of how to fly. This question is more application specific and is dependent on the stated objective of a mission. Where to fly is the focus of this research. For this research, the mission objective is to create accurate 3D models for inspection of infrastructure. The question for this research is what photos should be taken to create a complete model of the area of interest.

1.1 Literature Review

Structure from Motion (SfM) is the creation of 3D models from a set of 2D images. The basic principle is that two overlapping photographs from different positions are used to triangulate the location of points in space (see Fig. 1.1 for a visualization of this process). This process can be done without user intervention; however, user input can help to speed up or improve the reconstruction process [1]. One of the most common user inputs for SfM is ground control points (GCP). GCP are used to spatially reference a model. This referencing scales the model to the right size and puts it in a global coordinate system. If GCP’s are identified by the user prior to model creation, the SfM algorithms use a bundle adjustment. This adjustment speeds up the model creation as photos are pre-grouped and aligned based on the GCP’s. For some of the models in this study, GCP’s are used to reference and scale the models.

For successful SfM reconstruction, the photographs captured need to have sufficient overlap. Typically, optimal horizontal and vertical overlaps have been specified by commercially available SfM software. Some of the challenges for SfM include shiny or long narrow surfaces. Shiny objects are a challenge because bright spots (from reflected light) appear in different places depending on the camera location. This makes feature matching between photos difficult. Long narrow objects are also often difficult for SfM algorithms to match features on. The focus of this research is to address the first point: how to get sufficient overlap for model creation. To address the other challenges with SfM, sites are chosen that lend themselves to SfM reconstruction, avoiding these problems altogether. This allows for the research to focus on camera placement optimization.
Another challenge for SfM is the computational time needed to generate models. It is estimated that doubling the number of photos increases processing time for SfM models by four times [2]. There have been two approaches to overcoming this challenge. The first is to improve the SfM algorithms. There has been some success in this area, reducing portions of the problem to $O(n)$ without sacrificing accuracy [7]; however, the reconstruction process is still $O(n^2)$, limiting the speed at which SfM models are processed [8]. The second method to reduce the computational strain is to limit the number of photos needed for processing through a more intelligent selection of camera positions and poses. This method applies view planning to the SfM problem to reduce to processing load. When camera planning is not applied, excess photos are taken to ensure that the models do not have any holes in them. If there is insufficient coverage from multiple angles, model of the area of interest will have gaps in it. In the current work, camera planning is the approach taken to address the computation constraints.

1.1.1 Camera Planning

View planning for SfM has been an active area of research for a variety of applications and is also sometimes referred to as camera planning. As a general topic, camera planning includes planning camera positions, poses and sometimes includes specifying focal lengths and camera lenses. This problem has been applied to a variety of vision tasks, some examples include surveillance [9], target tracking [10], quality control [11], coverage [12] and SfM [13]. Although each
of these vision tasks have their own unique requirements, the general problem formulation and framework for each of these tasks are similar.

Liu et al. summarized the general problem formulation and framework [14]. For each of the problems, user input of models for the environment, cameras, and required task are needed. The inputs are discretized in space and a pool of possible camera positions and poses are generated. The camera poses and terrain information is used to generate a graph, sometimes referred to as a histogram, that describes what information each camera can capture. Several methods to generate this graph are described in [15]. Using the graph, the optimization problem is formulated as the set covering problem and solved for the optimal camera placement strategy. When seeking to cover a specified region, the optimization problem is often formulated as the set covering problem. More information about the set covering problem, a standard problem in combinatorics, is discussed in Sections 1.1.4 to 1.1.7.

The general workflow outlined by Liu et al. has been applied to camera view planning for SfM modeling with some minor modifications to the traditional surveillance and coverage problems due to the unique requirements for model construction. The major modification is centered around ensuring that each point is seen from multiple angles to ensure accurate reconstruction. Essentially, a method to achieve needed overlap is required. Hoppe et al. solved the problem for range scanners by creating groupings based on the viewing angle relative to the surface normal [16]. This was later applied to SfM allowing for traditional camera planning formulation once the cameras are grouped [4]. Schmid et al. developed a view planning algorithm that compared a potential candidate against already chosen cameras. This allowed for finer control of the desired angles between cameras [17]; however, this method is relatively slow making optimization computationally infeasible for all but the smallest cases.

This idea has also been extended to slightly different objective functions. For example, rather than only optimizing the number of photographs taken, Bircher et al. set their cost function to minimize the distance traveled by a UAV to obtain complete coverage [18]. Papirichos et al. seek to optimize a flight with constraints on battery life. They optimized the best set of photos that can be collected in a single flight [19].
1.1.2 Iterative Modeling

In recent years there has been minimal work on iterative modeling for SfM. For iterative modeling a high-level flight based on old or less accurate data is planned. This first flight is used to create a more accurate model. Subsequent flights are then planned based on the new model, allowing for more reliable flight plans. For example, Schmid et al. first performed a high-level flight based on a low resolution digital elevation model (DEM). They then developed a more accurate model based on the high-level flight to plan subsequent flights from [17].

In a similar, but different study, Martin et al. explored iterative modeling when applied to long linear features [20]. In their paper, they explored the decoupled problem of a high-level inspection and low level (higher detailed) inspection of objects of interest discovered during the high-level flight. This unique approach allows for increased accuracy when objects of interest were discovered, while still maintaining coverage over the entire area for monitoring.

Iterative modeling allows for view plaining with greater confidence without the need to apply probabilistic planning due to uncertainty. Although there has been significant research on planning the next best view in an unknown environment [21–24], for many regular inspection tasks, it is not necessary to add this level of complexity due to an already accurate model of the infrastructure. A simple high-level flight, or a prior model can be sufficiently accurate to plan flights from. These models also allow for greater confidence when planning flights closer to the ground.

1.1.3 Multi-Scale Modeling

A less studied area for SfM is the use of targeted multi-scale modeling. For regular inspections of infrastructure, there are often areas of higher interest to the inspector due to the higher risk for failure. In some cases, more thorough inspection are mandated by national inspection standards. Martin et al. touched on multi-scale modeling with their approach to monitoring long linear features; however, camera positions and poses for the different scales were planned separately [20]. In a recent publication, Khaloo et al. reported a proof of concept study on multi-scale modeling demonstrating that multi-scale photos can be used to achieve varied desired point densities for a model [25]. Later the same group applied these findings to a test case in [26]. In that study the
specified overlap and distances were used to create flight paths that were then flown manually. While they achieved desired accuracies, almost 5,000 photographs of the 85 m long bridge were collected. In this work, the author applies traditional camera placement algorithms to multi-scale monitoring, reducing the number of photos needed while maintaining desired resolutions.

1.1.4 Early View Covering Research and Solutions

One of the earliest examples of a view covering problem was introduced by Victor Klee in 1973 and is known as the art gallery problem [27]. For this problem, an art gallery is trying to place guards in such a way as to minimize the number of guards needed to view all of the walls. The problem was solved by reformulating it as the coloring problem. The solutions relied on the geometry of the system to find an optimal set of locations. For the more complex SfM camera placement problem, this simple solution used for this problem is infeasible. This means that only general set covering algorithms have been applied to the camera placement problem for SfM. According to the author’s knowledge, there has been no work to connect the quality and speed of these algorithms to the geometry of the terrain.

1.1.5 Combinatorial Optimization

Combinatorial optimization (CO) is a branch of optimization that focuses on choosing the best option from a finite set (or countably infinite set). There are many types of problems that fit into combinatorics. Some of the common problems studied include the traveling salesman problem and the set covering problem.

CO problems are typically categorized as being NP (non-deterministic polynomial-time), NP-hard or NP-complete. NP means that a given solution to the problem can be verified in polynomial time. NP-hard problems are problems that can be shown to be at least as hard as the hardest NP problems. NP-complete problems are problems that are both NP and NP-hard. The set covering problem in NP-complete [28]. For many NP-complete problems, heuristic algorithms are used to solve them in polynomial time.
1.1.6 Set Covering Problem Solutions

There are many common algorithms used to solve the set covering problem, within a threshold of accuracy. Some of the popular algorithms used to solve this problem include greedy heuristics, particle swarm optimization, and genetic algorithms. Greedy heuristics work by selecting the current best additional view and adding it to the list of selected views until the set is covered. Particle swarm optimization (PSO) takes advantage of swarm intelligence to solve the problem. Swarm intelligence leverages each of the solutions by sharing knowledge between independent particles (optimizations) to be used when planning for their next location to test. Finally, genetic algorithms (GA) treat solutions like genes and use natural selection principles to “evolve” a pool of solutions to converge on an optimal solution.

1.1.7 Set Covering for View Planning

Although there has been limited research on view planning for SfM, there has been some significant research on set covering for view planning applications. In many cases, the authors make minor adjustments to the heuristics of an algorithm to improve solution speed or quality. They then demonstrate for their application that their method performed better than the traditional one. For example, Al Hasan et al. developed the GRASP algorithm to solve the 2D stereo sensor planning problem [29]. This algorithm is essentially the greedy algorithm, except rather than selecting the best option each time, they probabilistically picked one of the best options, allowing for different solutions. The best solution from one of the iterations is then chosen. Several other examples of algorithms used for view planning are PSO [30], Binary Integer Programming (BIP) [31], GA [32] and simulated annealing (SA) [33]. For a more complete summary of algorithms used for camera view planning see [14].

1.2 Summary of Novel Contributions

The novel contributions included in this thesis are the following:

- Improved problem formulation and solving through vectorization of code and utilizing sparse matrices. These improvements in the code allow for the added complexities introduced in the rest of this thesis.
• Application of camera planning techniques to multi-scale inspections. This method is applied to a test site with promising results.

• Exploration of more rigorous optimizers to solve the set covering problem, including a comparison of the relative trade-offs based on algorithm used. A best method for selecting cameras is suggested.

• Physical verification of the set covering modeling method. This verification validates the findings about the best method for selecting cameras. Additionally, insight about the strengths and limitations of the overall camera placement method for SfM are made.

1.3 Thesis Outline

The topics covered in this thesis are as follows:

Chapter 2 describes the overall problem formulation method and includes significant improvements made to inherited code from previous graduate students. The problem is formulated as the set covering problem by grouping viewing angles. This formulation then allows the problem to be solved by traditional set covering problem techniques. To allow for more advanced optimization techniques, the problem was vectorized and sparse matrices were used in place of dense matrices. This led to almost an order of magnitude decrease in overall solve time.

Chapter 3 explores a multi-scale extension to the methods described in Chapter 2. This extension allows for targeted inspection based on need quality. This fits better with traditional inspection needs, as in many cases the government mandates different levels of inspection for a single site. This method was demonstrated to work well at Tibble Fork Dam in Utah, with improved model quality shown in the higher priority regions of the dam.

Chapter 4 investigates a variety of different algorithms for solving the set covering problem. 100 randomly selected sites and 20 handpicked sites were used to test 8 different algorithms to determine the best method. For all of the cases, publicly available terrain data was used to optimize the camera locations and poses. The methods were compared for solve time and the number of cameras selected. The best method is suggested for future use.

Chapter 5 verifies the results found in Chapter 4 with physical test flights. Each of the 8 algorithms were used to optimize a camera set for building a 3D model of rock canyon park. The
models were compared to demonstrate similar model quality for each of the methods despite the large difference in the number of cameras needed.

Chapter 6 discusses the overall conclusions and findings from this work. Future work suggestions are suggested, including additional work to verify the different algorithms on more detailed models.

There are also several appendices at the end of this work: all of the different set covering problem algorithms explored in Chapter 4 are included and additional plots from the flight tests in chapter five are found in Appendix B.
CHAPTER 2. OVERALL PROBLEM FORMULATION

For sensor planning, the formulation and set up used throughout the literature follows a similar path as outlined in [14]. Significant work on the problem formulation was already completed by prior graduate students [4, 20, 34]. This chapter overviews the general planning process and highlights the specific additional contributions to the basic formulation that was added by the author.

2.1 Problem Formulation

The problem formulation is outlined in Fig. 5.1. The first step is user inputs. For this step the user specifies the area to be modeled. This can be done using a kml format polygon object or can be passed in directly as a list of latitude and longitude points. The user also specifies other parameters, including the distance from the surface at which the photos should be taken (for example 50 m) and the desired percent coverage (typically 90-95%). The user can optionally choose to provide a model of the terrain as a meshed point cloud in a PLY file format. Other inputs include the minimum safety distance that the UAV must keep away from surfaces. Finally the user can optionally input priority points and distances from which those points should be modeled.

With all of the inputs from the user, the terrain information needed for camera planning is prepared. Elevation data is obtained for the area of interest using Google Maps API for the area of interest and a buffer of 20 m on all of the sides. If the user provided a PLY point cloud, the PLY data and the Google Maps data are combined to generate a single surface. This combination is done by removing the Google Maps points that are contained within the area modeled by the PLY data. The Google Maps data is then meshed using the Delaunay triangulation method [35], included in the python SciPy package [36]. The normals for each of the meshed triangles are generated for each of the meshed triangles, with the normal required to have a positive z component. If the user provides normals in their PLY file, the normal is not required to have a positive z component.
Next, potential camera positions and poses are generated. These are generated by placing a camera on each of the normals at the user specified distance from the surface. The cameras are oriented to look back down the normal, directly facing the surface triangle they were generated from. The potential camera locations are then checked to ensure that they are not underground or within the user specified safety distance from the surface. Cameras at invalid locations are removed from the camera set.

Once the cameras are generated, an optimal camera set is selected. To do this a graph of the cameras and which points they can see is generated. This is done using NumPy and many of the vectorization methods available in it. In addition to determining which cameras can see which points, the cameras are sorted by viewing angles Fig. 2.2. If the angle is more oblique than 45° then the camera is marked as not viewing the point. An example of the point categorization for one camera is shown in Figure 2.3. This sorting based on the viewing angle is the major difference between camera planning for structure from motion and the more traditional camera planning for coverage in surveillance applications. Combinatorial Optimization (CO) techniques are used to select a minimum number of cameras that view all the points with at least one camera for each viewing angle group. The CO methods and algorithms used to make this selection are explored in significant detail in Chapter 5.
Figure 2.2: The different viewing angles for a single surface point.

Figure 2.3: The different categorizations of points for a single camera location and pose.
Once the optimal set of cameras is chosen a minimum flight path is optimized for the UAV. This is done using the Christofides algorithm with a guaranteed solution within 1.5 times the global optimal value [37]. If the camera set is large, the cameras are clustered by location and the Christofides algorithm is applied to each of the clusters. This has a minimal effect on the overall performance because battery limitations mean that UAV’s are unable to collect all of the photos in a single flight. By pre-batching the photo groups, the computation time is greatly reduced without significant increasing flight time.

Finally, once everything is completed the python script outputs an ordered list of the optimized camera poses and positions for the flight. These outputs are formatted to work with both a in-house developed APP for flights as well as commercially available software.

2.2 Overall SfM Camera Placement Algorithm Improvements

To facilitate achieving the previously stated objectives of the work contained in this thesis, significant edits and improvements were needed for the algorithms. The key improvements made and the relative effect of the changes are detailed in the rest of this section.

2.2.1 Vectorization:

Much of the inherited code used serial processing. While this is not a problem for small areas, when trying to optimize large areas (for example, the Tibble Fork Dam test site used for the multi-scale work) the code takes a relatively long time. By vectorizing the code, a computer is able to make the calculations simultaneously instead of one after another, significantly speeding up solve time. This improved time allows for more complex camera selection techniques. One example of this is shown in Figure 2.4. In this example, nested for loops with an if statement were used to load in some variables. By using Numpy, these same variables were loaded without the for loops or the if statement, significantly reducing the computation time needed for large problems. This type of adjustment is done throughout the code minimizing the use of serial processing where possible.
vertex_indices = self.plydata['face']['vertex_indices']
for i in range(len(vertex_indices)):
    for v in range(3):
        vert_index = vertex_indices[i][v]
        vert = [
            x_verts[vert_index],
            y_verts[vert_index],
            z_verts[vert_index]
        ]
        point_cloud_triangle_list[i, v] = vert
        if vert not in point_cloud:
            point_cloud.append(vert)

(a) Serial Method

ply_point_cloud = np.vstack((x_verts, y_verts, z_verts)).T
ply_indices = np.concatenate(
    self.plydata['face']['vertex_indices']).reshape((-1,3))
ply_triangle_list = np.take(ply_point_cloud,
    ply_indices.ravel(), 0)
ply_triangle_list = np.reshape(ply_triangle_list,
    ply_indices.shape + (3,),
    order='C')

(b) Vectorized Method

Figure 2.4: An example of a serial code and improved code utilizing Numpy’s vectorization to avoid serial computations.

2.2.2 Sparse Matrices

This is an essential change to make using more computationally expensive optimization techniques possible for the set covering problem. Adjusting to use sparse matrices, reduced the size of large matrices from over a gigabyte to a couple megabytes. Additionally, matrix math is significantly faster because it only calculates the values where there are entries in the matrix. This speed increase and size reduction allows for more complicated combinatorial optimization techniques to be utilized with reasonable solve speeds. The improvement in speed for the greedy algorithm is shown in Fig. 2.5.

2.2.3 Improved Graphing

 Visualization of results is key for ensuring that results are reasonable. This included visualization of camera views for verification of histogram creation (see Fig. 2.3). Additionally, the
Figure 2.5: The time to solve the set covering problem with the greedy algorithm based on the number of triangles in the mesh.

final plots of camera selection and positioning have been improved. This is important because with multi-scale flights, the UAVs fly much closer to the ground, making visual verification that the drones will not crash important. An example plot is shown below (Fig. 2.6).

Figure 2.6: A example of the new plot output. The points are camera positions with lines showing their orientation. The red line is the optimized flight path.
2.2.4 Google Maps API

Previously Matlab’s mapping toolbox was used to obtain elevation data for the areas of interest [38]. The mapping toolbox could easily access data from the National Elevation Dataset (NED) maintained by the USGS. However, the motivation to distribute this package in an APP for public use, made finding an alternative method necessary. The author explored the option of using the Google Maps API to obtain elevation data. There have been some studies on the accuracy of Google Maps imagery and a single study by Wang et al. on the accuracy of the elevation data [39]. Wang found that for elevation data on roadways, Google Maps is more accurate than other sources.

To verify that similar results are true for other terrain, a georeferenced model of RCP is compared to elevation data from Google Maps and elevation data from the NED. A plot of the three data sets are shown in Fig. 2.7. Both methods lacked the detail to fit the entire section perfectly; however, the mean error for the Google Maps data and the NED data were 1.8 and 2.9 m respectively. This would indicate that the Google Maps data is on average, more accurate than the NED data for this site. This is consistent with the findings of Wang et al [39].

![Figure 2.7: A plot of Google Map’s elevation data against NED’s (USGS) elevation data when compared a georeferenced model (in green).](image)

2.2.5 Mobile Application Development

As part of the overall work for the cUAS project, work towards a final developed product is essential. The work has also progressed commercial APP development. This work has been
incorporated into a commercial APP for view planning optimization for infrastructure monitoring. The APP is an outcome of the Center for Unmanned Aircraft Systems, an Industrial/University Cooperative Research Center.
CHAPTER 3. MULTI-SCALE MODELING EXTENSION

3.1 Introduction

A less studied area for SfM is the use of targeted multi-scale modeling. For regular inspections of infrastructure, there are often areas of higher interest to the inspector due to the higher risk for failure. In some cases, more thorough inspection is mandated by national inspection standards. Martin et al. touched on multi-scale modeling with their approach to monitoring long linear features; however, camera positions and poses for the different scales were planned separately [20].

In a recent publication, Khaloo et al. reported a proof of concept study on multi-scale modeling demonstrating that multi-scale photos can be used to achieve varied desired point densities for a model [25]. Later the same group applied these findings to a test case in [26]. In that study the specified overlap and distances were used to create flight paths that were then flown manually. While they achieved desired accuracies, almost 5,000 photographs of the 85 m long bridge were collected. In this work, the author applies traditional camera placement algorithms to multi-scale monitoring, reducing the number of photos needed while maintaining desired resolutions.

3.1.1 Iterative Modeling

For multi-scale modeling, it is important to have an accurate model to plan flights from for the high accuracy areas. An accurate model ensures that the smaller features will be captured with the optimized camera arrangement. Publicly available elevation data is relatively sparse and does not model small features. Additionally, objects like trees and houses are not included in the models. These can pose a risk for the drone if a camera position is selected within these objects and could lead to a crash. One method explored to create more accurate models to plan from is iterative modeling.
In recent years there has been minimal work on iterative modeling for SfM. For iterative modeling a high-level flight is planned for based on old or less accurate data. This first flight is used to create a more accurate model. Subsequent flights are then planned based on the new and more accurate model, allowing for more reliable flight plans. For example, Schmid et. al. first performed a high-level flight based on a low resolution digital elevation model (DEM). They then developed a more accurate model based on the high-level flight to plan subsequent flights from [17].

In a similar, but different study, Martin et al. explored iterative modeling when applied to long linear features [20]. In their paper, they explored the decoupled problem of a high-level inspection and low level (higher detailed) inspection of objects of interest discovered during the high-level flight. This unique approach allows for increased accuracy when objects of interest were discovered, while still maintaining coverage over the entire area for monitoring.

Iterative modeling allows for view plaining with greater confidence without the need to apply probabilistic planning due to uncertainty. Although there has been significant research on planning the next best view in an unknown environment [21–24], for many regular inspection tasks, it is not necessary to add this level of complexity due to an already accurate model of the infrastructure. A simple high-level flight, or a prior model can be sufficiently accurate to plan flights from. These models also allow for greater confidence when planning flights closer to the ground.

3.2 Methods

For a test site, areas of higher priorities are specified along with a camera to surface distance for the region. The shorter the distance, the higher the resolution of the photographs, resulting in a higher density model. For the rest of this paper, the author will refer to high priority regions as the regions selected with the highest desired resolution.

The method for camera selection as outlined in Chapter 2 were used for this work with modifications for iterative and multi-scale modeling. The iterative work-flow is shown in Figure 5.1 with the first column being an initial flight and the second column being the iterative flight based on the updated terrain model. The optimal camera sets were selected using a greedy heuristic at each of the priority levels starting from the highest priority and sequentially working out to the lowest priority. For each subsequent priority level, the portions seen by previously chosen camera
sets were marked as seen. This prevented forcing repeat information to be gathered by subsequent lower priority regions. Starting at the highest priority region ensured that only selected cameras with more detailed information can cover areas in subsequent optimizations. A work-flow graphic is shown in Figure 5.1.

![Work-flow graphic](image)

Figure 3.1: Overall work flow for multi-scale iterative camera planning.

### 3.2.1 Platform

For this study the Phantom 4 manufactured by DJI was chosen for the flight tests. The camera on the Phantom 4 has a 94° viewing angle at a 4x3 ratio of width to height. This results in a 52.1° vertical viewing angle and a 78.2° horizontal viewing angle. A 90% reduction in viewing angle was applied as a safety margin to account for potential inaccuracies in GPS placement and or gimbal pitch. Additionally, due to battery constraints on the system each flight was limited to a maximum of 98 photos to ensure that there was sufficient battery to complete the flights.

### 3.2.2 Tibble Fork Dam Test Case

The Tibble Fork Dam in Utah (see Figure 3.2) was chosen as the test site for multi-scale modeling. An initial elevation model of the dam was obtained using the Google Maps API. There was significant model mismatch for this site due to the recent replacement of the dam in early 2017.
This model mismatch, in addition to the sparsity of the elevation model, made iterative modeling of the dam essential.

3.3 Results

The results from the optimized flights as well as a summary of the accuracy analysis and comparison are included here. For a more complete summary of the results see (Paper Citation).

3.3.1 Flight Planning Results

The initial optimized flight was planned at 80 m using the USGS publicly available terrain data Figure 3.3a. In total, there were 51 optimized photo locations. The flight took approximately 11 minutes to fly using the DJI phantom 4. Following the flight, model processing in Agisoft Photoscan took a total of 3 hours and 35 minutes. The processed model of TF Dam was subsampled from 3.3 million to 10 thousand faces for planning the multi-scale flights. The subsampled model is shown in Figure 3.3b

For the higher interest regions of the dam, priority regions were specified using priority points. The corresponding priority region is defined as all of the area within a 7 meter radius of
Figure 3.3: Models used for planning optimized flights. (a) Initially optimized camera locations and poses based on publicly available elevation data. (Elevation data in m). (b) Course subsampled model used for multi-scale modeling with priority points indicated.

The optimized multi-scale flight had a total of 266 camera locations and took a total of 350 seconds to run on an Intel Core i7-3630QM processor with a clock speed of 2.4 GHz and 8 GB of ram. The breakdown for the solve time for each of the regions are included in Table 3.1. High priority region required a relatively large number when compared to the medium priority region. This relatively large difference was due to the significant overlap between the high and medium sections on the left side of the Dam and the increased distance from the surface. The low priority region also needed a lot of cameras due to its relatively large size compared to the medium and high priority regions. The locations and poses for the cameras chosen are shown in Figure 3.4.

<table>
<thead>
<tr>
<th>Distance (m)</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>7.7</td>
<td>11.1</td>
<td>331.6</td>
<td>350.4</td>
</tr>
<tr>
<td>Cameras Selected</td>
<td>99</td>
<td>26</td>
<td>141</td>
<td>266</td>
</tr>
</tbody>
</table>

Table 3.1: The number of cameras and the computational timing results for each of the optimized sections.

For comparison, optimized flights were planned at each of the priority levels for the entire dam. The results and computational time for these flight plans are included in Table 3.2. As is
expected the higher the priority (or the closer the flight) the more photos needed to adequately model the entire dam. If the entire dam was flown at 15 m, then over 1,000 photos would be needed. By using a multi-scale tiered system, the photos needed for the inspection of the dam was reduced by 25%. This also translates to reduced processing time when constructing a model from the photos [8]. Additionally the serial method of selecting from each viewing level decreased the number of photos needed at the lower priority regions. For example the low priority region alone took 159 photos to cover the entire site; however, for the multi-scale flight the higher priority photos covered a portion of the site, reducing the total number of needed photos for the low priority region to 141.

<table>
<thead>
<tr>
<th>Distance (m)</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>1344</td>
<td>674</td>
<td>372</td>
</tr>
<tr>
<td>Cameras Selected</td>
<td>1063</td>
<td>434</td>
<td>159</td>
</tr>
</tbody>
</table>
3.3.2 Model Quality Results

The model from the multi-scale flight is shown in Figure 3.5. The analysis for the model accuracies and quality for the different priority regions were explored. The point accuracies for the different regions were not statistically different. While the accuracies ranged from 1 to 1.6 cm going from highest to lowest priority, the small sample set of 5 check points per priority region was insufficient to show a difference between the different areas. A larger sample size would allow for showing a statistically significant different assuming that the estimated standard deviations were correct.

Although the check points were insufficient to show a difference between the accuracies of the areas, the higher priority regions were able to capture more detail than the lower priority regions. This was not reflected in the accuracy testing because each of the check points were placed on large obvious features. The small features, for example the instrumentation on the dam, were not included in check points; however, for these smaller features, the high priority region was able to reconstruct a 3D version of them. For the low priority region, these small features were not modeled, with images of them projected down onto the surface.

3.4 Conclusions

In this work a multi-scale extension to existing camera planning algorithms for SfM modeling was explored. The optimization work showed that using multi-scale can reduce the number of photos needed to adequately inspect infrastructure. For the test case explored, multi-scale reduced the number of photos needed by 75%. Additionally, the serial method of selecting cameras moving from high to low priority regions limited the number of photos needed for the lower priority region. The number of cameras needed to cover the entire dam at the low priority level was reduced by 18 when accounting for information already captured in higher priority photos.

Some future studies can be done on these methods. Currently each of the priority levels are solved serially instead of simultaneously. A valuable future study would be to explore the potential gains of solving all the priority regions in a single set covering problem.
Figure 3.5: The final model processed in Agisoft from the multi-scale flight.
CHAPTER 4. SET COVERING ALGORITHMS FOR SFM CAMERA PLANNING

The heuristic used to solve the set covering problem during optimization of the camera placement for structure from motion modeling can have a significant effect on the performance of the optimization. Due to the wide variety of applications for the set covering problem, a variety of algorithms and heuristics for solving the set covering problem have been developed. In this section the author explores several of the different proposed algorithms to solve the set covering problem and their performance on the camera planning problem for structure from motion modeling.

4.1 Introduction

One of the earliest examples of a view covering problem was introduced by Victor Klee in 1973 and is known as the art gallery problem [27]. For this problem, an art gallery is trying to place guards in such a way as to minimize the number of guards needed to view all the walls. The problem was solved by reformulating it as the coloring problem. For the coloring problem solution, the vertices of the room are meshed into triangles. The vertices are then colored, in such a way that no connected vertices are the same color. Finally, one of the color that appears the least frequently is chosen. The vertices with the selected color make up the set of locations needed to view the entire area with a minimum number of guards (see Figure 4.1 for a visual representation of this solution process.)

This method has been explored for the best way to triangulate a 2D space and assign colors to ensure that the global minimum is found. The vertex coloring formulation of the problem used the geometry of the gallery to obtain a fast solution to the problem. This problem is relatively simple in 2D. For the more complex SFM camera placement problem, this simple solution is infeasible. This means that only general set covering algorithms have been applied to the camera placement problem for SFM. According to the author’s knowledge, there has been no work to connect the quality and speed of these algorithms to the geometry of the terrain.
4.1.1 Combinatorial Optimization

Combinatorial optimization (CO) is a branch of optimization that focuses on choosing the best set from a finite set (or countably infinite set). There are many types of problems that fit into combinatorics. Some of the common problems studied include the traveling salesman problem, the knapsack problem and the set covering problem. Formulation of the camera planning problem as a set covering problem allows for leveraging the significant research completed on efficient ways to solve these common problems.

CO problems are typically categorized as being NP (non-deterministic polynomial-time), NP-hard or NP-complete. NP means that a given solution to the problem can be verified in polynomial time. NP-hard problems are problems that can be shown to be at least as hard as the hardest NP problems. NP-complete problems are problems that are both NP and NP-hard. The set covering problem in NP-complete [28]. For many NP-complete problems, heuristic algorithms are used to solve them in polynomial time.

4.2 Set Covering Problem Solutions

There are many common algorithms used to solve the set covering problem, within a threshold of accuracy. Some of the popular algorithms used to solve this problem include greedy heuris-
tics, particle swarm optimization, and genetic algorithms. Greedy heuristics progress to a solution by selecting the current best additional view and adding it to the list of selected views until the set is covered. Particle swarm (PS) optimization (PSO) takes advantage of swarm intelligence to solve the problem. Swarm intelligence leverages each of the solutions by sharing knowledge between independent particles (optimizations) to be used when planning for their next location to test. Finally, genetic algorithms (GA) treat solutions like genes and use natural selection principles to “evolve” a pool of solutions to converge on an optimal solution. The following sections give background on the algorithms explored in this study.

The partial SCP is an extension to the traditional SCP. For the partial SCP only a specified number or fraction of the elements need to be covered. This problem has been explored in some detail by [40–42]. For many cases, partial SCP is solved by adapting the methods for the complete SCP. This method of adaptation is the approach taken in this work.

4.2.1 Greedy Heuristics

Greedy algorithms are popular due to their ease of implementation and the relatively fast solve times. Although extremely fast, greedy heuristics are prone to getting stuck in local minima. This tendency is due to the heuristic’s inability to re-evaluate past decisions or to make decisions accounting for the effect this will have on future decisions. This especially problematic for problems like the TSP, where a decision to visit the next closest city can quickly put the salesman in a location with no unvisited cities close by. For the set covering problem, the greedy algorithm can be effective if the problem has optimal structure [43].

There has been significant research on methods to improve the greedy algorithm to avoid local minima while maintaining the relatively fast solutions speeds. One of the earliest examples was the development of greedy randomized adaptive search procedure (GRASP) for the set covering problem [44]. When the GRASP algorithm is used each step is randomly chosen from several of the best options, allowing for locally suboptimal choices to be chosen with the potential to find some better global minima. This is done for a specified number of iterations and the best solution found is returned. Current researchers continue to expand and improve this method [45].

A second method to improve the greedy algorithm is allowing a re-evaluation of past decisions. This allows for the algorithm to change decisions made early on before later decisions are
considered. One example of this type of method is the Carousel Greedy (CG) algorithm proposed in [46]. For the CG algorithm the set covering problem is solved using the traditional Greedy algorithm; however, once a solution is obtained the earliest chosen cameras (vertices) are removed from the chosen set and new cameras are chosen. It is possible that the same cameras are chosen; however, with the additional information of the other cameras chosen, a different camera might be a better choice.

Another method stemming from greedy algorithm is the reverse greedy algorithm. In this case, rather than selecting the best choices, the worst choices are greedily removed from the solution set until no more can be removed while still covering the set. This method is used when only seeking to remove a few options from a set.

4.2.2 Linear Programming

The set covering problem can be described as a Binary Integer Programming (BIP) problem. This problem is a subset of the Integer Linear Programming (ILP) problem. The formulation of this problem is shown in Equation (4.1). One of the most common methods for solving ILP problems is the branch and bound algorithm, which was introduced in the 1960’s [47]. For this problem the relaxed Linear Problem (LP) is solved. This relaxed problem can be solved quickly. For one of the optimized values that are not already integers, the problem is branched with two problems introduced, one with a minimum constraint at the rounded-up value of the branched value and the other with a maximum constraint of the rounded down value. For the BIP problem, rather than adding a constraint the variable is set to be 0 and 1. This formulation has been used for the set covering problem for view planning with some success [31].

\[
\begin{align*}
\text{minimize} & \quad \sum x \\
\text{s.t.} & \quad Ax \geq 1 \\
& \quad x \in 0, 1
\end{align*}
\] (4.1)
4.2.3 Particle Swarm Optimization

PSO is a swarm intelligence method that takes advantage of swarm knowledge as well as individual knowledge. It was first proposed in 1995 by Kennedy and Eberhart [48]. For PSO an initial population is initialized with values and velocities. The optimization progresses with each particle taking a step in the direction of its velocity. After each step the velocity is updated considering personal best-found locations and the current best global value. Kennedy and Eberhart then extended this algorithm to work with for binary systems [49]. This technique has been adapted to solve the camera placement problem [50].

4.2.4 Simulated Annealing

Simulated Annealing (SA) is an optimization method that is based on the cooling of metals. For SA the variables are initialized. Then random changes or variations on the variables are introduced. The new solution is evaluated to determine if the solution is better than the previous best. If the solution is better, the variable values are accepted; however, if it is worse the new variable values might still be accepted. Whether or not the step is taken, is based on the current temperature of the system and how much worse the new set is. The higher the temperature the more likely a step to a worse solution will be accepted. As the temperature cools, the algorithm rejects the uphill steps, zeroing in on an optimal solution. This method is popular for its ability to avoid getting stuck in local minima. The ability to make steps to a worse solution allows for a wider exploration of the search space. Simulated annealing has been used to solve the SCP in several studies [31, 51–54]

4.2.5 Genetic Algorithms

The genetic algorithm utilizes the principles of natural selection seen in nature. This technique was first developed by Holland [55]. This system uses a population composed of several actors. Based on the fitness of these actors, they have the chance to pass on their composition to the next generation. Over many generations the “fittest” populations are more likely to reproduce so the population should converge to an optimal solution. This solution approach was applied to the SCP in 1996 by [56, 57] with some variations to account for specifics in the SCP. Research
utilizing the principles of a genetic algorithm for the SCP have continued for a variety of different set covering problems [53, 58].

4.2.6 Ant Colony Optimization

Ant colony optimization (ACO) was originally introduced in 1999 by [59]. ACO was later suggested for the set-covering problem by [60, 61]. ACO revolves around ants searching the solution space and using pheromones to tag good solutions and to communicate this to subsequent ants [59]. Ideally the optimal solution should have its set of pheromones reinforced until most ants follow the nominally optimal path. ACO in the set covering problem utilizes the constraint system unique to the SCP and the statistical knowledge gained by the ants to determine an optimal solution [61]. Many of the recent advances in using ACO for the SCP improve the overall solver by combining a more traditional ACO with heuristics and/or local search [14, 62, 63].

4.2.7 Novel Contributions

This chapter includes several novel contributions. To the authors knowledge, no work has been done to evaluate the different set covering methods for solving the camera planning problem for SfM. While there has been some research into each of the methods for other camera planning problems, the unique formulation of the SfM problem poses it’s own set of challenges for the optimizers. Specifically, with tracking the angles at which the points are seen, the problem grows in size faster than the traditional coverage problem. The author adapted several methods in the literature for the partial SCP to work with the SfM problem and tuned parameters for the best performance. Finally, the author evaluated the different methods to determine the best method for camera planning. This is done exploring the trade-offs between solution quality and solve time.

4.3 Methods

The problem is formulated as described in Chapter 2 of this thesis; however, for this study only the set covering portion of the problem is considered. For each of the selected test sites, a graph of the coverage by each camera is generated and saved to be used later for testing the set covering algorithms. The algorithms described later in the section are all included in Appendix A.
4.3.1 Site Selection

For this study 2 groups of sites are selected for testing. The first group is a randomly selected set of test sites. One hundred random locations within the continental US are selected Figure 4.2. These are selected using NumPy’s random number generator. For each of the random locations an area of interest is selected. The area of interest is created by generating a circle of random radius (ranging from 50 to 150 m) at each of the selected locations. The circles are then perturbed by scaling some of the points towards or away from the center point. In summary, sites of random size, shape and location are generated to test the different set covering algorithm’s performances. These sites will be used to make statements on the general trends and performances of the algorithms.

For the second method, 20 sites of interest (eg. landslides, dams, earthquake locations) are chosen by the author. These sites include manmade structures (eg. Tibble fork dam), places of historical significance (eg. Gettysburg memorial site), industrial sites (eg. Kennecott copper mine) and portions of national parks (eg. angles landing in Zions National Park). The variety of sites are chosen to capture a wide variety of potential terrain, some examples include, four corners (a completely flat test site), a golf course (a relatively smooth site with rolling hills) and the Grand Canyon (with steep walls and a jagged face). The performance of the SCP for these handpicked sites will be compared to the results from the randomly selected sites. The authors selected sites are shown in Figure 4.2.

4.3.2 Site Characterization

Each of the sets of sites, both randomly chosen and handpicked, are characterized by the terrain type. The simple characterizations used are surface area, change in elevation, and number of data points. In addition to these characterizations, the steepness and roughness of the selected areas are quantified. The steepness is quantified by averaging the absolute value of the first derivative with respect to displacement. The roughness is quantified as the average of the absolute value of the Hessian (the second derivative with respect to displacement). The differences in these values are shown for three illustrative examples of the handpicked sites in Table 4.1.
plots of the selected sites are shown in Figure 4.3. The metrics of size, change elevation, steepness and roughness can capture much of the variation in the types of sites that can be modeled.

Table 4.1: Site characterization values for three of the 20 hand selected sites. Plots for these three sites are shown in Figure 4.3

<table>
<thead>
<tr>
<th>Location</th>
<th>Cameras</th>
<th>Points</th>
<th>Area (m$^2$)</th>
<th>( \Delta ) Elevation (m)</th>
<th>Steepness</th>
<th>Roughness (m$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Golf Course</td>
<td>653</td>
<td>1956</td>
<td>6628</td>
<td>9.6</td>
<td>0.045</td>
<td>0.003</td>
</tr>
<tr>
<td>Grand Canyon</td>
<td>11371</td>
<td>33936</td>
<td>503296</td>
<td>519.4</td>
<td>0.674</td>
<td>0.018</td>
</tr>
<tr>
<td>Copper Mine</td>
<td>12636</td>
<td>37867</td>
<td>34798</td>
<td>303.5</td>
<td>0.572</td>
<td>0.079</td>
</tr>
</tbody>
</table>

4.3.3 Greedy Heuristics

As a baseline, a simple greedy heuristic is used for this study. The greedy simple greedy method is optimized for speed. The optimization is done using SciPy's sparse matrices in combination with NumPy. The code for this implementation can be found in Appendix A.

The carousel greedy method is based on the simple greedy algorithm. First an initial solution set is found using the greedy method, then the first 10% of cameras chosen are removed.
After removing 10% of the selected cameras, the camera that captures the most, new information is greedily added to the camera set. Once a new camera is added, the earliest chosen camera remaining is removed from the solution set. This cyclic adding and removing continues until each of the initially selected cameras have been removed. The remaining camera set is then greedily added to until it reaches the specified coverage.

The reverse greedy is coded up to remove the camera that had the least rare piece of information. This is accomplished by determining the number of times each of the points are seen for each viewing angle. This is then used to sort the points according to rarity. The rarest point for each of the cameras is then determined and the camera with the least rare point is selected. This method had problems scaling with size due to the sort necessary for each step.

### 4.3.4 Linear Programming

Some minor modifications to the traditional BIP formulation for the set covering problem are needed for our application due to only requiring a partial covering of the points. The problem formulation is changed from the formulation in Equation (4.1) to Equation (4.2). $x$ is constrained to be a binary integer; however, $y$ does not need to be constrained to be an integer, only to be less than or equal to one. This allows the $y$ variables to be continuous avoiding the need for branching on the variable.
\[ \text{minimize} \quad \sum x \]
\[ \text{s.t.} \quad Ax \geq y \]
\[ \sum y \geq n_{\text{required}} \]
\[ y \leq 1 \]
\[ x \in 0,1 \]

This problem is solved using Pyomo’s framework [64, 65] to interface with the COIN-OR solver [66]. The solver is set with a band gap of zero with a maximum solve time. For the 100 random locations, the solver is terminated after 15 minutes of solving. For the 20 hand selected sites, the solvers are terminated after 1 hour of solve time. When the problem is terminated due to time constraints, the current best feasible solution is returned by completing a single branch.

### 4.3.5 Ant Colony Optimization

The implementation of the ant-colony solver in this work is largely based on [63]. The solver relied on the initial iterative search of a single ant. In the initial iteration a row of the point histogram is chosen at random, the cameras that saw previously unseen points on this row are then listed and one is selected with equal probability. This process continued until 95\% coverage is achieved. Cameras are initialized with an equal probability of being selected, but upon the completion of a solution, the pheromone matrix (\( \tau \)) is updated to give additional weight to the cameras selected. Additionally, stronger weights are given when the solution is found with a fewer number of cameras and an evaporative rate of 5\% is given to the pheromones so unreinforced pheromones would decrease in strength. The formula for updating the pheromone matrix is given in Equation (4.3), where \( x \) is the boolean matrix indicating the selected cameras. This iteration is repeated until there had been no improvement for a set number of generations or a maximum number of iterations is reached.

\[ \tau = 0.95(\tau + \frac{100x}{\sum x}) \] (4.3)
4.3.6 Particle Swarm Optimization

The PS optimizer used in this work is largely based on the work by [49] as modified by [50]. The algorithm is modified to avoid the computationally expensive nearest neighbor method for each particle at each step. This significantly decreases solve time and allows for larger particle sets. The objective function used for the PS optimizer is show in Equations (4.4) and (4.5). This objective gives a penalty if insufficient points are seen and does not give any added benefit for capturing more points than are needed. This formulation of the objective function transforms the problem from the traditional SCP to the partial SCP. The author explored several different weightings for the insufficient coverage penalty. For PSO, it is found that a penalty of 5 ($P = 5$) had the best convergence properties, ensuring that specified coverage is maintained.

\[
\text{seen} = [A \cdot x \geq 1] \tag{4.4}
\]

\[
\text{objective} = P \cdot \max(0, n_{\text{required}} - \sum \text{seen}) + \sum x \tag{4.5}
\]

4.3.7 Genetic Algorithms

A genetic algorithm is implemented using the traditional steps of mutation and crossover. The fitness function returned a function of the points not seen (taking into account user specified desired coverage) and the number of cameras used (see Equations (4.4) and (4.5)). An initial population is randomly created and run through the fitness test. The reciprocal of the fitness score for each function is taken and raised to a power of 5. This was selected because of improved convergence properties with this transform of the fitness function. These values are normalized across the whole population and became the probability distribution for the selection of pairs to create the next generation. For each new member of the next generation two parent’s are selected from the previous generation based on the probability distribution, including the probability of the same parent being selected twice. The parents chromosomes where mutated and crossed, and one of the chromosomes is selected at random. This is repeated for all members of the new population. Subsequent populations are created until there is not improvement in the lowest fitness score for a set number of generations or maximum iterations are reached.
4.3.8 Simulated Annealing

A traditional simulated annealing optimizer is used in this work with the following properties. The selected camera set is initialized by randomly selecting 100 cameras. The method uses a linear cooling profile to reduce the temperature. At each iteration the number of cameras swapped is determined by the temperature with decreased swapping with decreasing temperature. Finally, to ensure complete coverage the simulated annealing optimizer is tied to a heuristic search to improve performance. The set created after swapping cameras is then either greedily added to or removed from to achieve specified coverage. Because the problem is guaranteed to have specified coverage, the different camera sets are only evaluated by the number of cameras needed to entirely cover the region. If the new set is smaller than the previous one, the new set is always kept; however, if the new method is worse, a Boltzmann distribution is used to decide if the worse set should be kept.

4.4 Results

Each of the methods described in Section 4.3 are used to solve for an optimal camera set for the randomly selected and handpicked sites. Each of the optimizations are solved on a Intel Core i7-3630QM at 2.4 GHz with 8 GB of ram. In total almost 80 hours of solve time were needed to solve the problem for each of the sites using the different methods.

4.4.1 Randomly Selected Sites

The average solve time and the number of cameras needed for the randomly selected sites are shown in the Pareto front in Figure 4.4. The cameras needed are calculated relative to the base method (the greedy algorithm). This is chosen instead of using the actual number of cameras needed because of the variation in the sites. A similar Pareto front is made normalizing the cameras against the area; however, the confidence intervals on the mean value is significantly larger.

Several of the methods did not work well for this problem, including RG, PS, SA and GA. These algorithms did not have a heuristic for selecting cameras. In the cases of the PS, SA and GA, the methods rely on random number generators to select the next camera. For the RG algorithm, rather than selecting the best cameras, the worst are eliminated. In other work the PS, SA and GA methods have been shown to have superior performance when compared to some of the heuristic
methods like the Greedy algorithm [31, 50, 52, 53, 54, 58]; However, for these studies, few cameras were needed to cover the area. Unfortunately, this combinatorial problem scales poorly with size. For example, when choosing between 50 different cameras, there are $2^{50}$ or $1.1 \times 10^{15}$ different combinations of cameras. As this increase to 100 different cameras for the larger sites this becomes $1.2 \times 10^{30}$. For the sites in this study the number of cameras ranged from 265 to 9988 cameras with a median of 2636.5 cameras and a mean of 3154.5 cameras. This large number of potential cameras makes exploring a reasonable amount of the search space difficult when the exploration is done randomly without any overall search heuristic. While there is the potential to randomly select one of the better solutions, with such a small portion of the search space explored, the chance of randomly picking one a superior combination is minimal.

There are several possible ways to improve the performance of PS, SA, and GA methods. These algorithms performance could be improved by introducing a heuristic to for picking cameras instead of using a random number generator. This could be done similar to the method used to adapt the ACO algorithm to the SCP. Another method to improve the performance is to increase the run time or number of iterations before termination while increasing the amount of random variation.
introduced to ensure exploration of more of the search space. Exploring additional methods for adjusting these algorithms is outside of the scope of this study; however, when intruding additional complexity in the camera selection method or increasing the number of iterations for the methods, the solve time will increase, potentially making the additional gains not worth the added time.

**Number of Cameras**

Of all the variables explored as discussed in the methods section, only the size directly related to the number of cameras needed. A plot of the number of cameras selected against the size of the site is shown in Figure 4.5. For the most part the methods resulted in a linear relationship between the number of cameras selected and the site size, following the expected trends. The parameters for a linear fit for the number of cameras is shown in Table 4.2. The smaller the slope, the better the method performed. In agreement with the findings when examining the Pareto front, the RG, GA, and PS methods performed significantly worse; however, for the closer methods (Greedy, CG, CBC, and ACO), it is not clear if there is a statistical difference between the performance of the methods, due to the fact that 95% confidence interval for the slopes overlap.

<table>
<thead>
<tr>
<th>Method</th>
<th>Slope (cameras/km²)</th>
<th>Intercept (cameras)</th>
<th>R-squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>1136</td>
<td>9.9</td>
<td>0.936</td>
</tr>
<tr>
<td>CG</td>
<td>1046</td>
<td>9.5</td>
<td>0.945</td>
</tr>
<tr>
<td>RG</td>
<td>2320</td>
<td>23.9</td>
<td>0.685</td>
</tr>
<tr>
<td>CBC</td>
<td>1380</td>
<td>3.9</td>
<td>0.800</td>
</tr>
<tr>
<td>ACO</td>
<td>1319</td>
<td>9.2</td>
<td>0.959</td>
</tr>
<tr>
<td>GA</td>
<td>2657</td>
<td>24.1</td>
<td>0.763</td>
</tr>
<tr>
<td>PS</td>
<td>2779</td>
<td>46.7</td>
<td>0.331</td>
</tr>
<tr>
<td>SA</td>
<td>1677</td>
<td>18.1</td>
<td>0.911</td>
</tr>
</tbody>
</table>

A paired t-test is used to determine if there is a difference between the four best methods. The results from the paired t-test are included in Table 4.3. In all cases there was a statistically significant difference in algorithm performance except for the comparison between the Greedy and CBC algorithms’ performance. With a two-sided p-value of 0.735 there is no evidence of a
Figure 4.5: The number of cameras needed versus the area to be modeled for the randomly selected sites.

difference between the two methods. Based on these results, for sites throughout the United States, the CG algorithm is the best algorithm to use to obtain the fewest number of cameras. These findings can be extended to the entire continental United States because the sites are randomly selected; however, the sites are specified to be approximately circular with radii ranging from 50 to 150 m limiting the application to sites that fit this description.

Table 4.3: Two-sided p-value for the paired t-test on the average number of cameras needed.

<table>
<thead>
<tr>
<th></th>
<th>ACO</th>
<th>CBC</th>
<th>CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>&lt;0.0001</td>
<td>0.735</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>CG</td>
<td>&lt;0.0001</td>
<td>0.0049</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>0.0004</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In addition to the direct relationships, other forms of the terrain parameterization are used. It is found that the number of cameras selected is closely related to the inverse of the graph den-
sities. These fits are shown in Figure 4.6. The slopes are also indicated in the figure with the confidence intervals and R-squared values indicated in Table 4.4. In this case the intercepts are fixed to zero, so there is only one degree of freedom. Despite removing a degree of freedom, these fits had larger R-squared values when compared to the area fit. Additionally, this metric accounts for the distance from the ground that the drone will be flown.

![Figure 4.6: The number of cameras needed versus the inverse of the graph density. Linear fits were made for each method with a fixed intercept of zero. The slopes are given in the table.](image)

**CBC Performance**

It was not expected that CBC would perform as poorly as it did. The method used a band gap of zero and no trimming for the branch and bound algorithm will result in the returning the
optimal value; however, the CBC algorithm was terminated after solving for 15 minutes. With this time limit, only 9 of the 100 test cases are able to solve in the given time allotment. For these cases, the solutions are better than any of the other methods, meaning that none of the other methods found the true optimal value. All the different characterizations are used to try to model if CBC would complete in the time limit.

The main indicator for the success of the CBC method is the number of elements in the set covering problem (see Figure 4.7). Using a cut-off of two thousand elements, 8 of the 10 successful maps are identified, while 5 of the 90 unsuccessful maps are included with the successful solutions. Although there are 5 maps that did not completely solve to an optimal solution, in each of these cases CBC selected a smaller final set of cameras than any of the other methods.

Solve Time

In addition to considering the number of cameras needed to completely cover the area to SfM modeling, it is important to take into account the time it takes to obtain the solution. The solve time, according to site size is shown in Figure 4.8 For each of the methods, there are 3 clearly separate sets of points Figure 4.9a. These can be separated by looking at the point density of the modeled area. The three point densities are indicated in Figure 4.9b by the point shapes. The different sizes are due to different densities from Google Maps. The frequency of the elevation data

<table>
<thead>
<tr>
<th>Method</th>
<th>Slope (cameras · density)</th>
<th>95% CI</th>
<th>R-squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>1.75</td>
<td>1.68 - 1.81</td>
<td>0.968</td>
</tr>
<tr>
<td>CG</td>
<td>1.62</td>
<td>1.56 - 1.68</td>
<td>0.970</td>
</tr>
<tr>
<td>RG</td>
<td>3.73</td>
<td>3.54 - 3.93</td>
<td>0.935</td>
</tr>
<tr>
<td>CBC</td>
<td>1.74</td>
<td>1.62 - 1.87</td>
<td>0.908</td>
</tr>
<tr>
<td>ACO</td>
<td>1.95</td>
<td>1.88 - 2.02</td>
<td>0.967</td>
</tr>
<tr>
<td>GA</td>
<td>4.11</td>
<td>3.89 - 4.34</td>
<td>0.928</td>
</tr>
<tr>
<td>PS</td>
<td>5.05</td>
<td>4.55 - 5.55</td>
<td>0.796</td>
</tr>
<tr>
<td>SA</td>
<td>2.68</td>
<td>2.58 - 2.79</td>
<td>0.961</td>
</tr>
</tbody>
</table>
is reported by the API. The point cloud is then created with double the frequency of the Google data. This created terrain data with significantly different point densities.

### 4.4.2 Handpicked Sites

The results for the handpicked sites are similar to the results for the randomly selected sites. A Pareto front for the handpicked sites is shown in Figure 4.10. This is fairly similar to the Pareto front for the randomly selected sites with a few notable exceptions. First, the 95% confidence interval on the percent more cameras for the RG, GA, and PS is significantly larger (more than doubling the size of the confidence interval). The other notable difference is the increased relative time for the SA and CBC methods to solve. For the handpicked sites, the CBC method is set to time out after 1 hour of solve time. This increased termination time is possible because there are less sites being investigated.
Figure 4.8: The solve time based on the model area for the randomly selected sites.

Figure 4.9: The solve times according to site size and sorted by point density.
Figure 4.10: Pareto front based on solve time and performance relative to the greedy algorithm for the handpicked sites.

Number of Cameras

One other major difference is the number of cameras needed and the area to be modeled. For the hand selected sites, instead of using a fixed distance, the UAV distance from the ground is varied based on the size of the site and the level of detail desired. This variability is apparent in Figure 4.11 with a significantly larger scatter in the lines for each of the methods. This is expected because the distance from the ground has a significant impact on the number of cameras needed to model a site. This difference is not accounted for with only fitting the against the area to be modeled.

On the other hand, the lines relating the inverse of the graph density and the number of cameras needed still followed a fairly linear line Figure 4.12. The slopes of these fits are included in Table 4.5. Although the flights are flown at varying elevations with a wide variety of terrain types and site sizes, these fits have a very high R-squared in general with the lowest being the CBC method with an R-squared of 0.906. The larger scatter in the CBC method is likely due to the fact that only some of the methods successfully solved before the timeout condition is reached.
Figure 4.11: The number of cameras needed versus the area to be modeled for the handpicked sites.

Table 4.5: Fitting parameters with the 95% confidence interval for the density fits.

<table>
<thead>
<tr>
<th>Method</th>
<th>Slope (cameras · density)</th>
<th>95% CI</th>
<th>R-squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>1.81</td>
<td>1.67 - 1.94</td>
<td>0.975</td>
</tr>
<tr>
<td>CG</td>
<td>1.67</td>
<td>1.54 - 1.8</td>
<td>0.974</td>
</tr>
<tr>
<td>RG</td>
<td>5.52</td>
<td>4.91 - 6.12</td>
<td>0.951</td>
</tr>
<tr>
<td>CBC</td>
<td>2.39</td>
<td>2.00 - 2.77</td>
<td>0.906</td>
</tr>
<tr>
<td>ACO</td>
<td>2.04</td>
<td>1.88 - 2.21</td>
<td>0.972</td>
</tr>
<tr>
<td>GA</td>
<td>4.59</td>
<td>4.14 - 5.05</td>
<td>0.959</td>
</tr>
<tr>
<td>PS</td>
<td>6.09</td>
<td>5.21 - 6.97</td>
<td>0.917</td>
</tr>
<tr>
<td>SA</td>
<td>2.01</td>
<td>1.84 - 2.18</td>
<td>0.970</td>
</tr>
</tbody>
</table>
In comparison with the findings from the randomly selected sites, many of the slopes fall within the 95% confidence interval from the randomly selected sites with only a few exceptions. For the handpicked sites, the CBC method did worse on average, especially compared to the other methods like the greedy algorithm. This could be due to a higher percentage of the sites selecting having more complex terrain.

4.5 Conclusions

In this work a variety of algorithms for solving the set covering problem for the SfM camera planning problem are explored. The explored algorithms included greedy heuristics (a traditional greedy, CG, and RG methods), BIP (in the form of the open source COIN-OR solver), swarm
intelligence (PS optimizer and ACO) and other heuristic methods (GA and SA). Based on the performance of the different methods and the tradeoff between time and solution quality, the author suggests using the carousel greedy method to solve the set covering problem for SfM camera planning. Although this is not the fastest method, the increase of average solve time when compared to the fastest method (a traditional greedy algorithm) makes the improved solutions worth the added time.

Based on this work there are several areas of future work to be explored. First this study relied on the assumption that the coverage evaluation is accurate. This allowed for comparing methods solely based on the number of cameras needed for calculated coverage. This assumption will be explored and validated in the next chapter.

Other future areas of work include a similar study using more accurate models. For this study, all the flights are planned based on publicly available elevation data. This data is sparse with little detail. The conclusions made only apply to this data and do not extend to more detailed models. Determining how the methods perform on better models would be a valuable follow up study.
CHAPTER 5. PHYSICAL FLIGHT TESTS

5.1 Introduction

Physical validation of simulated performance is an essential step to ensure that optimization results are not finding flaws in the modeling methods [67]. In chapter 4 the author explored different methods for picking camera sets for SfM modeling. It is assumed in Chapter 4 that two camera sets with similar reported coverage will result in similar models. This assumption allowed for conclusions on the best algorithm to select optimal camera position and poses for SfM. These conclusions only are valid if the assumption is also valid. In this chapter the modeling is validated for a selected test site. The test site is modeled using each of the methods selected camera sets. If this can be shown then the methods in Chapter 4 that select the fewest cameras to obtain complete coverage according to the model, are the best set of cameras to use.

5.2 Literature Review

There have been limited studies where camera planning has been physically verified outside of simulation. Some examples include [29, 53, 68–70]. There are several studies that have been performed for structure from motion that have been done in simulation. One example of physical verification is the study done by Schmid et. al [17]. Schmid tested camera planning on a hillside; however, there were several challenges with the reconstruction of these models. The site had significant amounts of snow when it was flown. This caused several holes in the model. The authors attributed the holes to the cameras that the SfM software was unable to use due to difficulties triangulating camera positions with the tie points. The authors did not analyze the relative size or percentage of the target area that was covered.

In another study Hoppe et. al. explored the camera planning application on a house [16]. The flight was planned based on an initial model created using SfM based on photos taken during
a manual piloted flight. Hoppe et. al. demonstrated that a good model could be created, however this model also, had holes in it. In this case the holes were attributed to the lack of texture in some regions, making complete model creation difficult. Like Schmid et. al. Hoppe et. al., this study did not report the size or relative amount of model missed.

One study by Martin et. al. reported the relative number/size of holes in the models [4]; however, in this case all the work was done in simulation using Terragen. This is the work that is the closes to the work done in this study. There are several novel contributions in this study. First, the work done by Martin et. al. was extended to use in physical flight tests. Second, the method is rigorously tested using several different camera sets, obtained using different camera selection methods, but evaluated using the same model. This more rigorously verifies the simulation method because it indicates that despite significantly varying camera sets, the similar models and similar evaluation in the simulation show the good performance of the method.

5.3 Methods

The selected site for this study is the spillway from the runoff collection dike at rock canyon park in Provo, UT. This site is chosen because although it is relatively small, the site is like dams and levees, which are common site of interest for structure from motion modeling. Additionally, this site has been used in previous studies [4, 20], allowing for comparisons to these studies. For the selected site, each of the set covering methods are used to optimize the camera set used for SfM reconstruction. The methods used are outlined in Chapter 4. The timing results and the number of camera poses chosen are shown in Table 5.1. A path to collect each of the optimized photo sets is planned using the Christofides algorithm to solve the TSP.

The optimized flights are flown using the DJI Phantom 4. This platform was chosen for its relatively high accessibility and market penetration, 72% of the worldwide drone market share [71]. The platform is ideal to test the optimized flight with due to its ease of use and reasonable price point. Key specifications for the camera on the platform are included in Table 5.2. The drone was autonomously flown with an in-house developed Android App using DJI’s APK. The app used the default automatic camera settings for focal length and exposure. All the flights were flown in a little under a continuous 3-hour period. The approximate flight time for each of the methods is also included in Table 5.2. The approximate times are calculated using the photo meta data.
Table 5.1: Solve time and number of camera positions and poses chosen for the SCP for RCP test site. The approximate flight time to collect the photos is also included.

<table>
<thead>
<tr>
<th>Method</th>
<th>Cameras</th>
<th>Solve Time (s)</th>
<th>Flight Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>49</td>
<td>0.017</td>
<td>9</td>
</tr>
<tr>
<td>CG</td>
<td>45</td>
<td>0.094</td>
<td>8</td>
</tr>
<tr>
<td>RG</td>
<td>92</td>
<td>1.994</td>
<td>16</td>
</tr>
<tr>
<td>CBC</td>
<td>42</td>
<td>3603</td>
<td>8</td>
</tr>
<tr>
<td>ACO</td>
<td>50</td>
<td>36.95</td>
<td>9</td>
</tr>
<tr>
<td>GA</td>
<td>61</td>
<td>40.49</td>
<td>10</td>
</tr>
<tr>
<td>PS</td>
<td>84</td>
<td>4.471</td>
<td>17</td>
</tr>
<tr>
<td>SA</td>
<td>63</td>
<td>10.64</td>
<td>12</td>
</tr>
</tbody>
</table>

by subtracting the earliest time from the latest. Each of the methods would also have a slight increase in time for the take-off and landing. The methods were flown back to back to minimize the difference in weather and lighting between the different methods; however, on the day of the flights, there were occasional clouds that blocked the sun, causing some variation in lighting.

Table 5.2: Several of the camera specifications for the DJI Phantom 4’s camera.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>1/2.3” CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field of View</td>
<td>94</td>
</tr>
<tr>
<td>Image Size (Pixels)</td>
<td>4000x3000</td>
</tr>
<tr>
<td>Horizontal Viewing Angle</td>
<td>75.2</td>
</tr>
<tr>
<td>Vertical Viewing Angle</td>
<td>56.4</td>
</tr>
</tbody>
</table>

The collected photos are processed using Agisoft’s SfM software, PhotoScan [72]. No ground control points are used to aid in the processing of the models or to geographically reference the models; however, the metadata added to the Photos by the Phantom 4 included the GPS locations of the photos, allowing for georeferencing of the models. The models are processed using Agisoft’s suggested workflow: aligning the photos, optimizing the alignment, building a dense cloud and then building a mesh. The output report from Agisoft is used to analyze the number of cameras that can see each point.

Coverage for the models is determined using CloudCompare. The models from Agisoft are imported to CloudCompare. The models are trimmed to the region of interest for comparison.
mesh for each of the trimmed region are sampled to have the same number of points as the model. Finally, the distance from the sample points to the original point cloud is calculated. Distances greater than 4 cm are deemed to be holes in the model. An example work flow for this method is shown in Figure 5.1. Finally, the distances for each of the sampled points are exported for numerical analysis.

![Sample point cloud with hole](image1)
![Mesh of the sampled cloud](image2)
![Sub-sampled points from the mesh](image3)
![Points colored by distance](image4)

Figure 5.1: Visual of the work flow for finding holes in a model.

### 5.4 Results and Discussion

The camera set used included a total of 512 potential cameras. The percent cameras in common between the different methods are shown in Table 5.3 calculated using Equation (5.1) where the row method is the reference method. The similar heuristically based Greedy and CG algorithms had high common coverages. This is not surprising because the CG algorithm starts with the Greedy heuristic and then systematically removes and adds cameras. This would indicate that many of the cameras that are removed are then replaced back into the camera set because it is still the current best option to cover the newly uncovered points. Interestingly the CG and CBC
had relatively high overlap (52%); however, the Greedy algorithm and CBC only had 17% overlap. This happened even though that the Greedy and CG had a 47% overlap with respect to the CG set.

\[ \text{pct}_{\text{common}} = \frac{n_{\text{common}}}{n_{\text{reference}}} \]  

(5.1)

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>CG</th>
<th>RG</th>
<th>CBC</th>
<th>ACO</th>
<th>GA</th>
<th>PS</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>100%</td>
<td>43%</td>
<td>22%</td>
<td>14%</td>
<td>16%</td>
<td>12%</td>
<td>16%</td>
<td>22%</td>
</tr>
<tr>
<td>CG</td>
<td>47%</td>
<td>100%</td>
<td>24%</td>
<td>49%</td>
<td>29%</td>
<td>16%</td>
<td>20%</td>
<td>27%</td>
</tr>
<tr>
<td>RG</td>
<td>12%</td>
<td>12%</td>
<td>100%</td>
<td>12%</td>
<td>14%</td>
<td>17%</td>
<td>18%</td>
<td>17%</td>
</tr>
<tr>
<td>CBC</td>
<td>17%</td>
<td>52%</td>
<td>26%</td>
<td>100%</td>
<td>29%</td>
<td>26%</td>
<td>26%</td>
<td>21%</td>
</tr>
<tr>
<td>ACO</td>
<td>16%</td>
<td>26%</td>
<td>26%</td>
<td>24%</td>
<td>100%</td>
<td>14%</td>
<td>22%</td>
<td>16%</td>
</tr>
<tr>
<td>GA</td>
<td>10%</td>
<td>11%</td>
<td>26%</td>
<td>18%</td>
<td>11%</td>
<td>100%</td>
<td>20%</td>
<td>13%</td>
</tr>
<tr>
<td>PS</td>
<td>10%</td>
<td>11%</td>
<td>20%</td>
<td>13%</td>
<td>13%</td>
<td>14%</td>
<td>100%</td>
<td>14%</td>
</tr>
<tr>
<td>SA</td>
<td>17%</td>
<td>19%</td>
<td>25%</td>
<td>14%</td>
<td>13%</td>
<td>13%</td>
<td>19%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The probability of randomly selecting the same cameras if both algorithms randomly selected their solution set is also calculated. This is done for each of the combinations. Equation (5.2) is used to determine the percentage in common between the cameras. It is the probability that camera set \( A \) will have at least \( n_{\text{com}} \) cameras in common with set \( B \), where \( n_A \) and \( n_B \) are the number of cameras in the camera sets and \( n_{\text{tot}} \) is the total number of cameras that the sets are chosen from. The resulting probabilities for the camera to have at least as much overlap as they did are given in Table 5.4. When comparing the better performing algorithms, it is found that the probability of randomly selecting that many in common is very low; however, for many of the other methods, they have a high probability of randomly selecting as many as they did in common. This high probability means null hypothesis, that the camera set is randomly selected, cannot be rejected.

\[ \text{probability}(A|B) = \left[ \sum_{i=n_{\text{com}}}^{n_A} \binom{n_B}{i} \cdot \binom{n_{\text{tot}}-n_B}{n_A-i} \right] / \binom{n_{\text{tot}}}{n_A} \]  

(5.2)

Models for each of the flights are generated using AgiSoft’s PhotoScan software package and are included in Appendix B. Several of selected models are included in Figure 5.3. Qualitatively each of the models appeared to be similar. In all the cases there is some difficulty in completely modeling the vertical walls in the center of the spillway, see Figure 5.2. Some of this
Table 5.4: The probability of randomly drawing at least as many in common between the two camera sets.

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>CG</th>
<th>RG</th>
<th>CBC</th>
<th>ACO</th>
<th>GA</th>
<th>PS</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>0%</td>
<td>0%</td>
<td>29%</td>
<td>11%</td>
<td>11%</td>
<td>58%</td>
<td>62%</td>
<td>3%</td>
</tr>
<tr>
<td>CG</td>
<td>0%</td>
<td>0%</td>
<td>20%</td>
<td>0%</td>
<td>0%</td>
<td>32%</td>
<td>35%</td>
<td>1%</td>
</tr>
<tr>
<td>RG</td>
<td>29%</td>
<td>20%</td>
<td>0%</td>
<td>14%</td>
<td>11%</td>
<td>8%</td>
<td>39%</td>
<td>10%</td>
</tr>
<tr>
<td>CBC</td>
<td>11%</td>
<td>0%</td>
<td>14%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>8%</td>
<td>7%</td>
</tr>
<tr>
<td>ACO</td>
<td>11%</td>
<td>0%</td>
<td>11%</td>
<td>0%</td>
<td>0%</td>
<td>43%</td>
<td>21%</td>
<td>30%</td>
</tr>
<tr>
<td>GA</td>
<td>58%</td>
<td>32%</td>
<td>8%</td>
<td>1%</td>
<td>43%</td>
<td>0%</td>
<td>33%</td>
<td>53%</td>
</tr>
<tr>
<td>PS</td>
<td>62%</td>
<td>35%</td>
<td>39%</td>
<td>8%</td>
<td>21%</td>
<td>33%</td>
<td>0%</td>
<td>38%</td>
</tr>
<tr>
<td>SA</td>
<td>3%</td>
<td>1%</td>
<td>10%</td>
<td>7%</td>
<td>30%</td>
<td>53%</td>
<td>38%</td>
<td>0%</td>
</tr>
</tbody>
</table>

error can be attributed to model mismatch. The terrain elevation data used to plan the flights lacked the detail needed to capture the short vertical walls on the spillway.

Figure 5.2: Point cloud for the model using the RG algorithm’s selected cameras.

Holes in the models are then determined using the method outlined in Section 5.3. The resulting colored clouds for several of the selected sites are shown in Figure 5.3. These plots verified the original observation that many of the models struggled to adequately capture the vertical walls. In Figures 5.3b, 5.3d and 5.3f the red points represent a hole with a minimum radius of 4 cm.

In addition to a visual comparison of the models, the cloud to cloud distances are compared numerically. The cumulative distribution function (CDF) for each of the methods are shown in Figure 5.4. The model distances are very similar with slight variations. At the 4 cm cutoff the variation in the CDF is less than 0.02. This indicates that the models perform within 2% of each
other for coverage of the selected site. The relatively similar coverage performance between the models would indicate that the structure used to evaluate if a set of cameras will obtain sufficient coverage is accurate. This validates the conclusions made in Chapter 4 that are solely based on the number of cameras needed according to the algorithms.

The final aspect explored for this study is the number of cameras that viewed each point. Plots of the number of cameras viewing each point for several of the camera sets are shown in Figure 5.5. The CG, CBC and ACO models are chosen because they are some of the models

Figure 5.3: Models for three of the selection methods including the visualization of the holes (shown as red points) in the models.
made from the fewest photos (45, 42 and 50 photos respectively). The PS model is selected to represent the models made from significantly more photos (84 photos). In all the cases the area selected is covered by significantly more cameras than the three required by the algorithm. There are two main contributors to this result. First, the set covering problem in not set up as the exact set cover, meaning that repeat cameras are not excluded from the optimization problem. Secondly, the buckets only include angles up to 45°. This means that any photos that capture any information outside of this angle are not accounted for in simulation. This also contributes to the excess of cameras.

The models created each had significant extra overlap; however, the CBC camera set has noticeably less extra overlap when compared to the other models. Even the models with only a couple more cameras had significantly more extra overlap. This additional information could help with the accuracy of the model; however, accuracy is outside the scope of this study.

The extra coverage for the models illustrates on the major limitations of the current problem formulation method for camera planning. By artificially created viewing “buckets”, the problem becomes easier to optimize; however, this ignores the additional information that a camera can provide even if it is in the same bucket. As an illustrative example see Figure 5.6. In this case the three cameras are sufficient for the single point; however, the buckets this time do not demonstrate this.
5.5 Conclusions

In this study the validity of the simulated performance of a camera set is evaluated. All 8 camera selection methods outlined in Chapter 4 are used to optimize camera sets for SfM modeling of a test site. Although the camera sets varied greatly in number of cameras and had relatively few cameras common between them, especially when compared to the number of common cameras if they are randomly selected, the model coverage is similar between the models with a maximum
Figure 5.6: Illustrative example of the problem that buckets create for accurately determining if a set of cameras are sufficient to model a point.

variation of 2%. This demonstrates for the selected site that the algorithms models are similar if the algorithm evaluates them to have similar coverage. There are several limitations with the overall method that are illustrated by the study. The major one is the excessive overlap for all the methods due to the way that the cameras are grouped based on the angle from the normal. In future work, exploring subdividing the buckets or accounting for the distance between the cameras could improve the performance of the algorithm.
CHAPTER 6. CONCLUSIONS

Camera planning for 3D modeling has the potential to improve current industrially accepted infrastructure monitoring techniques. In this work, several aspects of camera planning for 3D modeling were explored adding to the scientific pool of knowledge in the area.

The inherited problem set-up and code was improved and optimized. Although the ideas applied to the problem here were not, in and of themselves novel, the adjustments made for the methods to work for the specific application were. The adjustment to using sparse matrices and matrix math instead of serial calculations were key for the findings in the rest of the work of this thesis.

An extension to multi-scale modeling from the more traditional approach was tested and verified. It was shown that the number of cameras could be significantly reduced by planning cameras based on a tiered accuracy instead of a single high accuracy. Additionally, the author demonstrated that using a serial approach from higher to lower priority and accounting for cameras already selected, reduced the overall number of cameras needed when compared to planning the tiers separately. This method was then verified at the Tibble Fork Dam, demonstrating it’s effectiveness.

In addition to exploring the use of iterate multi-scale modeling, more rigorous methods for solving the set covering problem were explored. 8 common methods used for camera planning were implemented and explored for the Structure from Motion camera planning problem. The author evaluated the different methods based on camera set selected and the time to find the solution. The author suggests that for the given problem, the carousel greedy algorithm is the best method to solve the problem. This algorithm is a good balance between the needed solve time and the quality of the solution found. It was also found that this method was within 5 to 10% of the optimal value when the true value was known.
Finally, the findings in Chapter 4 were validated on a test site. A test site was used to plan an optimized flight for each of the different set covering problem algorithms. The flights for each of the methods were flown and models were created from them. No significant difference between the resulting models was found.

6.1 Future Work

There are several potential future studies that could be explored based on the findings from this work. The suggested future work is broken down by chapter study.

Chapter 2

Although there was significant improvement made in the code, further optimization will improve the performance. Specifically, there are several instances in the problem formulation where the same information is repeatedly calculated. A careful review of the entire code can improve the speed by storing the information that will be needed later.

Chapter 3

The multi-scale extension to the traditional planning problem demonstrated the effectiveness of the method; however, each of the priority levels were solved sequentially. A simultaneous formulation could result in an improved solution; however, this will require a longer solve time. Although the combined problem will probably be too computationally expensive to use regularly, understanding the impact that separating the levels has on the solution will lead to a better understanding of the cost of breaking the problem into portions.

Chapter 4

The different set covering methods were all tested against publicly available terrain data. This data was typically at least a meter apart, meaning that small details were lost. Future work exploring the performance of the different methods on more detailed models (for example ply files
from previously flown sites) will allow for evaluation of the different algorithms performance in these cases.

Additional work can also be done on some each of the methods used. While the ACO algorithm performed the best out of the algorithms not based on the greedy algorithm, improvements in tuning and the camera selection method could lead to a better performing method. Several of the other methods (GA, SA and PSO), also showed some potential and with improved selection techniques could result in a reliable method for selecting a good camera set.

Chapter 5

In this chapter it was clearly shown that more photographs are taken than are needed to model the area. This is in large part due to the formulation method. This formulation method allows for the application of the set covering problem; however, it means that new information captured by a camera is sometimes classified as information captured by previous cameras. This leads to an excess of cameras selected. Future work in reformulating the problem or making more angle groupings could improve the overall performance, reducing the number of cameras needed.

Additionally, the study in Chapter 5 was done using publicly available terrain data. Repeating the study on a more accurate model, perhaps from a previous flight, would add additional incite as much of the detail is not captured in the publicly available terrain data.

Additional Future Areas

In addition to the specific studies stemming from the chapters of this work, several other topics would be of value to explore. One valuable study would be to explore the combined set covering problem and traveling salesman problem. This problem is described as the generalized traveling salesman problem. This problem is significantly more difficult to solve than either of the separate problems; however, by separating the problems into the two parts, it is likely that a suboptimal solution is found. Although separating the problem into parts likely results in a suboptimal solution, the methods to solve the generalized traveling salesman problem are slow and are not guaranteed to find the optimal solution. It will be valuable to examine the relative solve times and solutions found between the combined and separated problem. The author predicts that
the separated problem formulation will outperform the combined formulation; however, to the authors knowledge this had not been shown in the literature.

Extensions of the existing methods to account for new sensor types is also a valuable area of study. A wide variety of sensors are used to conduct infrastructure monitoring. While much of the monitoring is visual, other methods, for example, infrared or chemical sniffers, are also used. Extending the existing methods and applying the techniques used for structure from motion can add additional information for inspectors.
REFERENCES


[38] Mathworks, “Matlab mapping toolbox.” 16


66


APPENDIX A. SET COVERING SOLVERS

This appendix includes the solvers used for each of the Methods in Chapters 4 and 5. All of the code is in python. The following import statements are used throughout and will not be repeated for each of the functions.

Listing A.1: Imports used by all of the optimizers

```python
import numpy as np
import scipy as sp
import math
from scipy import sparse
import matplotlib.pyplot as plt
```

Listing A.2: Greedy Method

```python
def greedy_sort(hist, coverage, max_cameras, use_camera_limit):
    # This is just the regular greedy algorithm. It should take in a 2d
    # histogram [triangles x cameras]
    points, cameras = np.shape(hist)

    # Remove points that are never seen.
    enough_seen = points * coverage
    not_needed = int(points - enough_seen)

    # Precalculate the transpose to make the rest faster
    transhist = hist.transpose()

    # Initialize a matrix to track cameras already chosen. Also load in the
    # previous camera information.
    chosen = np.zeros(cameras)
    notseen = np.ones(points)
```
camerascore = transhist.dot(notseen)

# Keep picking cameras until all the points have been seen
while (notseen.sum() > not_needed or (chosen.sum() < max_cameras and use_camera_limit)):
    picked = np.argmax(camerascore)  # Find the camera that sees the most unseen points
    chosen[picked] = True  # Select that camera
    notseen[hist.getcol(picked).A.squeeze()] = 0  # Update the new points that were seen
    camerascore = transhist.dot(notseen)

return chosen.astype(bool), True

---

def carouselgreedy(hist, coverage, max_cameras, use_camera_limit, alpha=10, beta=.90):
    ### Start with a greedy sort
    points, cameras = np.shape(hist)

    # Remove points that are never seen.
    enough_seen = points * coverage
    not_needed = int(points - enough_seen)

    # Precalculate the transpose to make the rest faster
    transhist = hist.transpose()

    # Initialize a matrix to track cameras already chosen. Also load in the previous camera information.
    chosen = np.zeros(cameras)
    notseen = np.ones(points)
    picked = []

    camerascore = transhist.dot(notseen)
# Keep picking cameras until all the points have been seen
while (notseen.sum() > not_needed or (chosen.sum() < max_cameras and use_camera_limit)) \ 
    and any(camerascore > 0):
    picked.append(np.argmax(camerascore))  # Find the camera that sees the
    chosen[picked[-1]] = True  # Select that camera
    notseen[hist.getcol(picked[-1]).squeeze()] = 0  # Update the new
    camerascore = transhist.dot(notseen)

# Now set up for the carousel greedy
size_keep = int(math.floor(np.shape(picked)[0]*beta))
notseen_init = (hist.dot(chosen) < 1).sum()

i = 0
while i < alpha*size_keep:
    picked = picked[-size_keep:]
    chosen = np.zeros(cameras, bool)
    chosen[picked] = 1
    notseen = (hist.dot(chosen) < 1).astype(int)
    picked.append(np.argmax(transhist.dot(notseen)))
    i = i+1

chosen = np.zeros(cameras, bool)
chosen[picked] = 1
notseen = (hist.dot(chosen) < 1).astype(int)
while notseen.sum() > notseen_init:
    picked.append(np.argmax(transhist.dot(notseen)))
    chosen[picked[-1]] = 1

notseen[hist.getcol(picked[-1]).squeeze()] = 0
return chosen, True

Listing A.4: Reverse Greedy Method
```python
from pyomo.environ import *
from pyomo.opt import SolverFactory

def pyomo_cbc(hist, coverage, max_cameras, use_camera_limit):
    # Create the model
    model = make_model(hist, coverage, max_cameras, use_camera_limit)

    opt = SolverFactory('cbc')  # bonmin, cbc
    opt.options['seconds'] = 60*60  # Time to run before termination

    print('starting solve now')
    results = opt.solve(model)  # To see solver output add: tee=True
    results.write()
```

Listing A.5: CBC Method
model.solutions.load_from(results)

# model.pprint()
values = {}
for v in model.component_objects(Var, active=True):
    print("Variable", v)
    varobject = getattr(model, str(v))
    values[str(v)] = [varobject[index].value for index in varobject]
picked = [0 if v is None else v for v in values['x']]
# ys = [0 if v is None else v for v in values['y']]
# current_coverage = sum(hist.dot(picked) > 0) / hist.shape[0]

if results.solver.termination_condition._EnumValue._get_key() == 'optimal':
optimal = True
else:
optimal = False

return np.asarray(picked), optimal

---
def aco(hist, coverage=0.95, max_cameras=0, use_camera_limit=False):
    hist_csc = hist
    hist = hist.tocsr()
    points, cameras = np.shape(hist)

    enough_seen = points * coverage
    not_needed = int(points - enough_seen)

    transhist = hist.transpose()

    point_array = np.zeros(points)
    for i in range(points):
        point_array[i] = i

    camera_array = np.zeros(cameras)
    for i in range(cameras):
        camera_array[i] = i
\[
\text{tau} = \text{np.ones(\text{cameras})} \quad \# \text{will be the matrix that keeps track of the pheromone data} \\
\hspace{0.5cm} \# \text{will control the secondary features}
\]

\[
\text{min_value} = \text{cameras} \\
\text{count} = 0 \\
\text{best_sum} = \text{min_value}
\]

\[
\text{max_iterations} = 1000 \\
\text{no_change} = 200 \\
\text{result} = \text{False}
\]

\[
\text{for k in range(\text{max_iterations}):} \\
\hspace{1cm} \text{chosen} = \text{np.zeros(\text{cameras})} \\
\hspace{1cm} \text{notseen} = \text{np.ones(\text{points})} \\
\hspace{1cm} \text{chosen2} = \text{np.ones(\text{cameras})} \quad \# \text{will keep track of cameras that have not been selected} \\
\hspace{1cm} \text{secondary} = \text{transhist.dot(\text{notseen})}
\]

\[
\text{while notseen.sum()} > \text{not_needed:} \\
\hspace{1cm} \text{point_probability} = \text{notseen} / \text{notseen.sum()} \\
\hspace{1cm} \text{row_selection} = \text{int(np.random.choice(\text{point_array}, 1, p=point_probability)[0])}
\]

\[
\text{row} = \text{hist.getrow(\text{row_selection})} \quad \# \text{Get row selected by row_selection} \\
\hspace{1cm} \text{options} = \text{sp.sparse.csr_matrix(row).multiply(\text{chosen2}).todense()} \quad \# \text{Get list of cameras that haven't been selected}
\]

\[
\text{camera_probability} = \text{np.multiply(\text{tau, options})} \\
\hspace{0.5cm} \text{camera_probability} = \text{np.multiply(\text{camera_probability, secondary})} \\
\hspace{1.5cm} \text{camera_probability} = \text{camera_probability / np.sum(camera_probability)} \\
\hspace{1.5cm} \text{camera_probability} = \text{np.asarray(camera_probability).ravel()}
\]
camera_selection = int(np.random.choice(camera_array, 1, p=
camera_probability.squeeze())[0])

chosen[camera_selection] = 1
chosen2[camera_selection] = 0

notseen[hist_csc.getcol(camera_selection).A.squeeze()] = 0
secondary = transhist.dot(notseen)

factor = 100
factor2 = 0.95
factor3 = 100
num_cameras = chosen.sum()
for i in range(cameras):
    if chosen[i] == 1:
        tau[i] = tau[i] + factor * 1/num_cameras
    tau = tau * factor2

if chosen.sum() < min_value:
    min_value = num_cameras
    best = chosen
    best_sum = min_value
    count = 0
    count = count + 1
    if count == no_change:
        result = True
        break

return best, result

-----

def RandomChange(initialArray, probability, probability2 = 0.4):
    array = initialArray
    for i in range(len(array)):
        if np.random.rand() < probability:
            if array[i] == 1:
array[i] = 0

else:
    if np.random.rand() < probability2:
        array[i] = 1

return array

def Crossover(initialArray1, initialArray2, crossoverRate):
    if np.random.rand() > crossoverRate:
        crossoverPoint = np.random.randint(0, len(initialArray1))
        fraction1 = initialArray1[:crossoverPoint]
        fraction2 = initialArray1[crossoverPoint:]
        fraction3 = initialArray2[:crossoverPoint]
        fraction4 = initialArray2[crossoverPoint:]

        newArray1 = fraction1
        newArray1 = np.append(newArray1, fraction4)
        newArray2 = fraction3
        newArray2 = np.append(newArray2, fraction2)
    else:
        newArray1 = initialArray1
        newArray2 = initialArray2

    return newArray1, newArray2

def FitnessTest(x, hist):
    seen = hist.dot(x.T) >= 1
    coverage_score = np.maximum(0, 0.95 * hist.shape[0] - seen.sum(axis=0))

    return coverage_score + np.sum(x), np.sum(x), coverage_score

def gen_opt(hist, coverage=0.95, max_cameras=0, use_camera_limit=False):
    hist_csc = hist
    hist = hist.tocsr()

    points, cameras = np.shape(hist)
enough_seen = points * coverage
not_needed = int(points - enough_seen)
transhist = hist.transpose()

point_array = np.zeros(points)
for i in range(points):
    point_array[i] = i

camera_array = np.zeros(cameras)
for i in range(cameras):
    camera_array[i] = i

population = []
populationSize = 100
fitnessScore = np.ones(populationSize)
populationPointer = np.zeros(populationSize)
cameraScore = np.zeros(populationSize)
coverageScore = np.zeros(populationSize)

for i in range(populationSize):
    populationPointer[i] = i

for i in range(populationSize):
    #population.append(np.random.randint(2, size=cameras))
    population.append(np.zeros(cameras))

count = 0
best_cameras = np.zeros(cameras)
best_fitness = 100000

max_iterations = 5000
no_change = 100
result = False
selectionProbability = np.zeros(populationSize)
for j in range(max_iterations):
    for i in range(populationSize):
        fitnessScore[i], cameraScore[i], coverageScore[i] = FitnessTest(population[i], hist)
        for i in range(populationSize):
            if fitnessScore[i] < best_fitness:
                count = 0
                best_fitness = fitnessScore[i]
                best_coverage = coverageScore[i]
                best_cameras = population[i]
                count = count + 1

    if count > no_change:
        result = True
        break

mutationRate = 0.001
 crossoverRate = 0.05

selectionProbability = 1/(fitnessScore)**5
selectionProbability = selectionProbability / np.sum(selectionProbability)  # Normalize to use in the np.random.choice() function

for i in range(populationSize):
    chromosomeOnePointer = int(np.random.choice(populationPointer, 1, p=selectionProbability.squeeze())[0])
    chromosomeTwoPointer = int(np.random.choice(populationPointer, 1, p=selectionProbability.squeeze())[0])
    chromosomeOne = RandomChange(population[chromosomeOnePointer], mutationRate)
    chromosomeTwo = RandomChange(population[chromosomeTwoPointer], mutationRate)
    option1, option2 = Crossover(chromosomeOne, chromosomeTwo, crossoverRate)

    if np.random.rand() < 0.5:
population[i] = option1
else:
    population[i] = option2

#return best_cameras, best_fitness, best_coverage, result
return best_cameras, result

Listing A.8: Particle Swarm Method

def swarmopt(hist, coverage, max_cameras, use_camera_limit):
    '''
    * c1 : float
cognitive parameter
    * c2 : float
social parameter
    * w : float
inertia parameter
    * k : int
    number of neighbors to be considered. Must be a positive integer less than :code:`n_particles`
    * p : int {1,2}
    the Minkowski p-norm to use. 1 is the sum-of-absolute values (or L1 distance) while 2 is the Euclidean (or L2) distance.
    '''

def objFun(x):
    seen = hist.dot(x.T) >= 1
    coverage_score = np.maximum(0, coverage * hist.shape[0] - seen.sum(axis=0))
    return coverage_score * 5 + x.sum(axis=1)

n_particles = 1000  # 50000
options = dict(c1=0.1, c2=0.1, w=4, k=[], p=1)  # int(n_particles)
opt = binary.BinaryPSO(n_particles, hist.shape[1], options)
cost_best, pos_best = opt.optimize(objFun, print_step=1, iters=3, verbose =0)
Listing A.9: Simulated Annealing Method

def sim_anneal(hist, coverage, max_cameras, use_camera_limit):

    # Clean up the histogram will need to reverse the indexing to return final results.
    hist_orig = hist
    good_cameras = np.any(hist.A, axis=0)
    hist = hist_orig[:, good_cameras]

    t = 500  # Temperature parameter
    n = 0    # References for switched indicies
    m = 0    # Second Reference for switched indicies
    # boltz = 1.38064852e−23  # Constant for cooling
    boltz = 0.1
    #initialize a set of cameras with 95% coverage
    # chosen = greedy_sort(hist, coverage, max_cameras, use_camera_limit)
    points, cameras = np.shape(hist)
    # Initialize a matrix to track cameras already chosen. Also load in the previous camera information.
    idx = np.arange(cameras)
    chosen = np.zeros(cameras, dtype=bool)
    chosen[np.random.choice(idx, 100)] = 1
    transhist = hist.transpose()
# Determine total points needed to be seen.

enough_seen = points * coverage
not_needed = int(points - enough_seen)

# To make the heuristic better we will be swapping cameras out for "like" cameras
# this sets up the comparison
# it is formatted as old_camera x probability to pick given camera based on common coverage
common = np.dot(hist.T.astype(float), hist.astype(float)).A**2
np.fill_diagonal(common, 0)
prob = common/common.sum(axis=1, keepdims=True)

prev_best_score = np.sum(chosen) + \max(0, \sum(hist.dot(chosen) < 1) - not_needed)
best_chosen = np.copy(chosen)

# try to replace different cameras
while (t >= 270):
    # print(t, '
    # need to do fewer swaps as time goes on
    chosen_new = np.copy(chosen)
    numchanges = np.rint(.40*sum(chosen)*t/273).astype(int)
    notseen = hist.dot(chosen_new) < 1  # gives booleans for all faces of whether they are seen or not

    # Camera to remove
    n = np.random.choice(idx[chosen_new], size=numchanges)

    # Camera to add back in
    prob_current = prob[n, :][:, np.logical_not(chosen_new)] / prob[n, :][:, np.logical_not(chosen_new)].sum(axis=1, keepdims=True)
    prob_current = prob_current.sum(axis=0)
    prob_current = prob_current / prob_current.sum()
m = np.random.choice(idx[np.logical_not(chosen_new)], size=numchanges)
# This is based on the commonality between the two
# m = np.where(not_chosen)[np.random.randint(0, np.sum(not_chosen))] #
take all places we haven't already chosen cameras and choose one to put in
chosen_new[n] = 0 #switch out
chosen_new[m] = 1 #switch in
# print(chosen_new.sum())

notseen = hist.dot(chosen_new) < 1 # gives booleans for all faces of
whether they are seen or not

# Check to remove extra cameras if we have sufficient coverage
while sum(notseen) <= not_needed:
    # print('not seen:', notseen.sum(), '\t not needed:', not_needed)
    # print('ncams:', chosen_new.sum())
times_seen = hist.dot(chosen_new.astype(int))
sort_idx = np.argsort(times_seen)
minimum_idx = np.argmax(times_seen[sort_idx] > 1) - not_needed
least_common_point = np.argmax(hist[sort_idx, :][sort_idx, chosen_new], axis=0).squeeze()
    if any(least_common_point >= minimum_idx):
        which_chosen = np.argmax(least_common_point)
        chosen_new[np.where(chosen_new)[0][which_chosen]] = 0
        notseen = hist.dot(chosen_new) < 1 # gives booleans for all
faces of whether they are seen or not
    else:
        break

# Add in cameras if not enough seen
camerascore = transhist.dot(notseen.astype(float))
while sum(notseen) > not_needed:
    # Greedily add cameras back in till sufficient coverage
    picked = np.argmax(camerascore) # Find the camera that sees the
most unseen points
    chosen_new[picked] = True # Select that camera
    notseen[hist.getcol(picked).squeeze()] = 0 # Update the new
points that were seen
camera_score = transhist.dot(notseen.astype(float))

# print('not seen:', notseen.sum(), '\t not needed:', not_needed)

# distilled = reversegreedy(hist, coverage, max_cameras, use_camera_limit)
score_old = np.sum(chosen)
score_new = np.sum(chosen_new)

# Update global best if score improves on previous best
if score_new < prev_best_score:
    prev_best_score = score_new
    best_chosen = np.copy(chosen_new)
    # print('new best found')

# accept if better model or if boltzman probability is greater than the random percent
if (score_old > score_new or np.exp((score_old**2 - score_new**2)/(boltz*t))) > float(np.random.randint(0,100))/100.0:
    chosen = chosen_new
    # print('new set selected')
t = t-1

chosen_final = np.zeros(hist_orig.shape[1])
chosen_final[np.where(good_cameras)[0][best_chosen]] = 1
return chosen_final.astype(bool), True
APPENDIX B. PHYSICAL FLIGHT TEST MODELS

Models for each of the camera selection methods.

Figure B.1: Greedy
Figure B.2: CG

Figure B.3: RG
Figure B.6: GA

Figure B.7: PS
Figure B.8: SA