ASTRO-TSP: TRAVELING SALESMAN PROBLEM BASED SOLUTIONS FOR SCHEDULING ASTRONOMICAL BASED OBSERVATIONS

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Mitchell Humphries

2023

COMMITTEE MEMBERSHIP

PROJECT: ASTRO-TSP: TRAVELING SALESMAN PROBLEM BASED SOLUTIONS FOR SCHEDULING ASTRONOMICAL BASED OBSERVATIONS

AUTHOR: Mitchell Humphries

DATE SUBMITTED: Fall 2023

Department of Computer Science

Dr. John Korah
Project Committee Chair
Assistant Professor of Computer Science

Dr. Breanna Binder
Assistant Professor of Astronomy and Astrophysics

Dr. Charles Shapiro
Scientist/Technologist
Jet Propulsion Laboratory, California Institute of Technology

ACKNOWLEGEMENTS

I would like to acknowledge the contributions of George Matta, who provided invaluable assistance with implementing and troubleshooting the methods developed for this project.

Additionally, I would like to acknowledge Dr. Charles Shapiro for defining the project's objectives as part of the JPL University Crowdsourcing Initiative.

ABSTRACT

In the fields of Astronomy and Astrophysics, efficient utilization of observatory observation time is essential. This paper introduces the Astronomer's Traveling Salesman Problem (Astro-TSP), a scheduling problem for the process of creating an observation schedule. The Astro-TSP shares many similarities to the traditional Traveling Salesman Problem (TSP). The Astro-TSP considers unique observational constraints such as time-dependent observation lengths, limited observation windows, and observation priorities. While the classic TSP establishes a foundation for this problem, the complexities introduced by the specific requirements of astronomical observations require a specialized solution. Considering these complexities, we propose and demonstrate a Genetic Algorithm tailored to solve the Astro-TSP.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1:

INTRODUCTION

Observatory observation time is a highly valued resource for the fields of astronomy and astrophysics. Astronomers, if granted time to use an observatory, make all efforts to use their allocated time efficiently. To do this, astronomers must create a schedule of the observations they wish to perform. While this task can be done manually, when only a few observations need to be scheduled, this problem grows in complexity as more observations are considered. As more observations need to be considered, the schedule produced manually is likely to be inefficient leaving out observations which may have been possible with a better optimized schedule. An automated solution has the potential to create this more efficient schedule and improve the workflow for astronomers, potentially leading to an increase in the scientific impact of observatories.

For every observation, an astronomer must consider a variety of variables. These may come from the specific observatory being used, the weather of that night, the position on earth of the observatory, and the characteristics of the target the astronomer wishes to observe. These variables all create unique constraints that limit when the observation can be performed and influence the time required to complete the observation. These constraints include time-dependent observation lengths and time windows. The observation length, the length of time required to perform an observation, is time-dependent when targeting a fixed signal-to-noise ratio. This time dependency is required to consider varying amounts of atmospheric interference and background noise. Additionally, each observation has a time window; this window may be restricted to ensure that the target can be viewed, or the time window may be restricted to limit background noise in the data collection processes. Astronomers may also set a priority for the

observation as a measure of the importance. When observations have varying importance, we aim to maximize the property of the observations scheduled. We name this scheduling problem the Astronomers Traveling Salesman Problem (Astro-TSP). If we were to remove the time dependency, time windows, and priority constraints from the problem, then creating this schedule resembles the classical Traveling Salesman Problem (TSP). While several variants of the TSP are shown to share similarities with Astro-TSP, no complete solution exists for the Astro-TSP. This led to the development of a Genetic Algorithm (GA) for Astro-TSP, which could leverage the strong west-to-east heuristic solution currently used by astronomers.

While we have created the Astro-TSP problem specifically to find a solution to the scheduling problem astronomers face, the GA solution discussed in this paper can be used for related scheduling problems. Notably, our solution leverages the ability to create a strong solution heuristically and then improve upon this solution in an evolutionary manner. For Astro-TSP, our solution also uses parallelization techniques to reduce the runtime allowing it to be applied to problems with larger search spaces. With the parallelization methods of our solution, the relationship between the number of observations to be considered and the runtime is approximately linear.

In this paper, we first further motivate the need for a solution for the Astro-TSP in Chapter 2. This includes why we want to solve Astro-TSP and the benefits a solution would have beyond observation scheduling. Then, in Chapter 3, we look at the current solution astronomers may use, and we explore several TSP Variants and proven techniques for solving them. Next, the specific requirements of Astro-TSP are provided in Chapter 4. A brief introduction to key astronomy concepts, along with an overview of Genetic Algorithms (GA) is presented in Chapter 5. Chapter 6 presents an overview and explanation of our developed GA.

Each step of the GA is provided in Sections 6.1.1 to 6.1.5, and Section 6.1.6 details the parallelization technique used. Chapter 7 presents our experimental validation. We introduce two algorithms developed for comparison; additionally, we explain how a schedule is evaluated. In Chapter 8 we provide our concluding thoughts and ideas for future work to improve upon our solution further.

CHAPTER 2:

MOTIVATION

One of the most valued tools available to astronomers is observatories and it is essential to use them efficiently. One such observatory is the Palomar Observatory[1] in Southern California, where the Astro-TSP is solved by hand using experience and intuition.

As part of the Next Generation Palomar Spectrograph (NGPS) for the 200" Hale telescope, which is expected to begin operations in 2024, Jet Propulsion Laboratory (JPL), brought forward the requirements and criteria that make up the Astro-TSP. The NGPS is designed to be highly efficient relative to other spectrographs. This was achieved by allowing more light into the instrument leading to a reduction in observation time and by making the instrument highly automated. With these design advantages an astronomer's choice in scheduling can become a limiting factor in the amount of data collected.

While the NGPS serves as the initial motivation for the Astro-TSP, it is not its only use. A developed solution to the Astro-TSP could be used by astronomers who are trying to plan their observation schedule at any number of observatories.

In addition to helping advance the field of Astrophysics, a solution to Astro-TSP could be used to solve similar problems. Problems that have similar constraints to Astro-TSP will be of greater interest; these constraints include time windows and time dependency. Problems that have a strong heuristic solution are also of higher interest. For Astro-TPS, we consider the natural west-to-east solution, which is currently used by astronomers when creating a schedule, to be the strong heuristic solution. For Astro-TSP, this heuristic exists due to targets appearing to

[1] Palomar Observatory Information: https://sites.astro.caltech.edu/palomar/homepage.html

rise in the east and set in the west. As seen in Figure 1, position A occurs earlier in the night and has a greater amount of atmosphere, between the observatory and the target than position B. Waiting till the observation is directly above the observatory, such as in position B, reduces the amount of atmosphere interference. It is important to minimize the amount of astrosphere because "the earth's atmosphere is absolutely opaque to many wavelengths and highly absorbent to others" [1]. This results in the general trend that the less atmosphere present at the time of the observation, the stronger the signal will be and thus require less time.

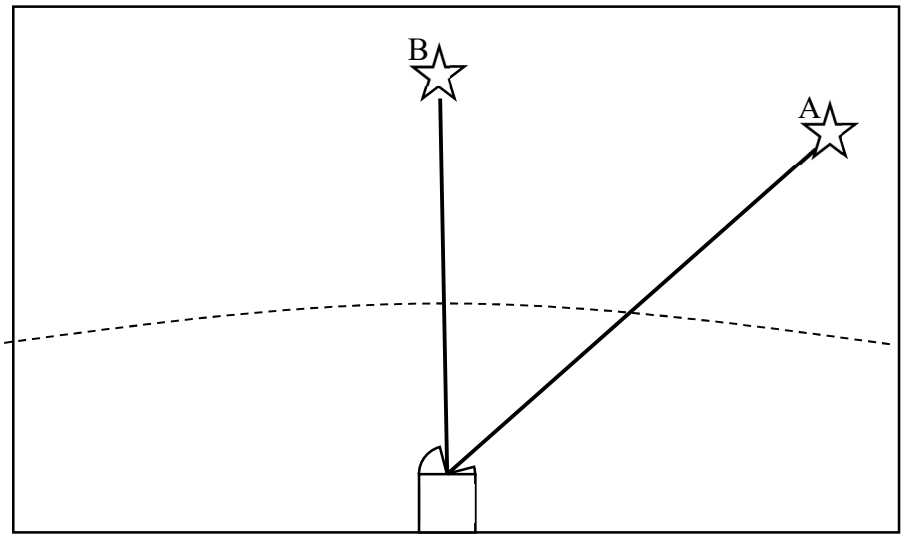


Figure 1: Possible Observation Positions

A solution to Astro-TSP will also utilize parallel processing due to the large search space. As presented in Section 6.1.6, this parallelization was done with a Genetic Algorithm (GA). Due to the versatility of GA and being population-based, the parallelization methods applied to Astro-TSP may be applicable to other problems as well.

CHAPTER 3:

LITERATURE REVIEW

The Astro-TSP is a unique new problem whose requirements were set for the specific problem of scheduling observations at observatories when targeting a fixed signal-to-noise ratio. As such, no existing research has been done that fully considers all requirements of the problem. A typical workflow for most astronomers is to start with a rough west-to-east sorting and then manually adjust the order based on the scientific value of the targets and other considerations..

Due to no preexisting research being done on the complete Astro-TSP, in this literature review, we will discuss the problem from two perspectives. First, we will examine Astroplan[2], a current tool that astronomers can use for scheduling observations. We will then examine the Astro-TSP as a combination of TSP variants. To do this, we will first briefly introduce common variants, then current state-of-the-art solutions. The TSP is well-studied in the field of operations research. Additionally, metaheuristics have been shown to be effective as well. These two areas of research may offer viable solutions to consider.

3.1: Astroplan

Astroplan, a Python[3] package part of Astropy, is currently the closest off-the-shelf tool we could find for scheduling observations. While it does offer two scheduling algorithms, Astroplan does not fully encompass the requirements for the Astro-TSP. Astroplan does not have a method for handling time dependency for the Astro-TSP. Instead, an observation must require the same amount of time regardless of when it is performed. Astroplan does allow for a constraint to be set limiting the airmass present when an observation is done. This allows astronomers to set a threshold limiting when their observation can be done to minimize the

[2]Astroplan: https://astroplan.readthedocs.io/en/latest/installation.html
[3]Python: https://www.python.org/

interference. Despite the limitation of time dependency, it is still useful to consider the two scheduling algorithms it offers to astronomers.

The first scheduling algorithm offered is a sequential sort. This method allows astronomers to set an ideal time for each of their observations. The scheduling algorithm will then place the observations in an order that places them as closely as possible to this ideal time. Simple in nature, this scheduling method is like the west-to-east solutions astronomers currently use at the Palomar Observatory. The second scheduling algorithm uses the priority of the observation to determine when it should be scheduled. This scheduling algorithm will schedule the most important observation at its ideal time; this is then repeated with the next most important observation.

Astroplan not only offers two scheduling methods but also details of its code structure. Astroplan employed an object-oriented programming approach, featuring several objects with which astronomers can interact. Among these are a Schedule, Observing Blocks, Target, and the Scheduler. The structure of Astroplan served as a reference in the development of our model and solutions for the Astro-TSP.

3.2: TSP Variants

Before discussing variations of the TSP, one must first understand the classic TSP. "The Traveling Salesman Problem is to find the route for a salesman who starts from a home location, visits a prescribed set of cities, and returns to the original city in such a way that the total distance traveled is minimal, each city is visited exactly once" [2]. The TSP problem is NP-hard, indicating that no known algorithm can solve it in polynomial time. In addition to the TSP, there is a related NP-complete decision problem named TSP-decision. In this version, given a set of cities and a specified distance of k, the question is: Does a tour exist with a total distance less

than k? The TSP decision does not aid in finding a solution for the Astro-TSP and will not be discussed further.

When viewing the Astro-TSP as a variant of the TSP, several studied variants exist that share similarities. Mathematical formulas for the following TSP variants can be found in "The Travelling Salesman and its Variants" by Punnen and Gutin [3]. The Traveling Salesman Problem with Time window (TSPTW) is a variant of the TSP where each city is assigned a window in which the salesman must visit the city. This is compatible with the requirement in Astro-TSP, which requires that each observation is performed within its time window.

The Time-Dependent Traveling Salesman Problem (TDTSP) is a variant of the TSP where the cost to travel between cities changes throughout the salesman route. Typically, the problem is stated where the cost to travel between cities changes after each city is visited; however, alternative formulations exist where the cost is dependent on the total length the salesman has traveled in the tour. The required length of an observation in the Astro-TSP is dependent on the current time of night, similar to the latter variant of TDTSP.

The Prize-Collecting Traveling Salesman Problem (PCTSP) includes a reward for visiting each city. However, a maximum total distance the salesman can travel is set. The salesman must then find the route that gathers the most reward and return to the starting city while not exceeding the set maximum distance. For the Astro-TSP, there is no limit to how much the observatory can travel. Instead, the time windows serve as the limiting variable, preventing all observations from being performed.

3.2.1: Operations Research

The field of operations research has produced many solutions to both the TSP and TSP variants. One such method is linear programming. Linear programming is a mathematical modeling method for representing problems as a set of linear equations, known as constraints, and an objective function.

The first mathematical formulation for the symmetrical TSP was proposed by Baker [4]. Baker's model could solve instances of up to 50 cities to optimality in reasonable amounts of time for the available hardware at the time in 1983. Kara expanded upon Bakers' formulation by adapting it to solve both the symmetric and asymmetric TSPTW. With Karas formulation, when tested against 30 established TSP problems, the solutions found matched for 21 instances and archived seven new best solutions [5].

Clímaco *et al*. investigated two methods for the PCTSP. These solutions were mixed-Integer Programming (MIP) solutions and Branch-and-cut (B&C). The B&C algorithm was shown to have solved all instances in a much shorter time than the other approaches from the literature yet proved unprecedented optimality for some instances [6]. Clímaco *et al*. did note that the B&C algorithm may be unable to solve problems of larger size due to machine memory issues. The MIP tests were able to achieve the same optimality but in a much shorter period. It is due to the findings of Kara and Clímaco *et al.* that we first considered linear programming or MIP for the Astro-TSP.

3.2.2: Genetic Algorithms

A Genetic Algorithm (GA) is a type of metaheuristic. Metaheuristics are a class of algorithms that guide the search process to find high-quality solutions to complex problems. These are general and flexible algorithms that iteratively explore the solution search space.

Metaheuristic algorithms include but are not limited to Ant Colony Optimization [7] [8], Tabu Search [9], and Genetic Algorithms [2]. Several metaheuristics, including genetic Algorithms, are non-deterministic. Thus, there is no guarantee of finding the most optimal solution. For Astro-TSP the most optimal solution is not required, rather a near-optimal solution is sufficient for use with the NGPS.

Potvin explains that GAs are a technique inspired by natural evolution where each solution is represented as an individual in a population. Potvin demonstrated the ability of GAs to produce competitive schedules for TSP problems of medium to large sizes up to a few hundred cities [2]. Additionally, they explain the inherent ability of GAs to be parallelized to allow for solving larger, more complex problems.

Deng *et al*. showed the importance of initialization methods in GAs. By using a k-means clustering algorithm to initialize the population compared to random or greedy technique, k-means can decrease the average error, the percent by which the found solution exceeded the best-known optimal, by approximately 25.16% and 34.39%, respectively [10]. This demonstration of the influence of a strong initialization method is one of the factors that led to a GA as a solution for the complete Astro-TSP as seen in Chapter 6. While k-means clustering may not be the best fit for Astro-TSP, a natural heuristic like k-means can be used. For Astro-TSP, a strong initial method can be done using the west-to-east pattern astronomers may currently use.

While Astroplan is currently the closest off-the-shelf tool available for scheduling observations, it lacks the ability to handle time dependency and does not feature any scheduling algorithms beyond basic heuristic processes; these limitations motivate the need for other solutions. In the field of operations research, the TSP and its variants are well studied; however, they often rely on ideal circumstances such as perfectly linear travel time and visiting all cities.

Additionally, the methods discussed from operations research may also suffer and be ineffective due to the vastly larger search space of Astro-TSP.

CHAPTER 4:

RESEARCH FRAMEWORK AND OBJECTIVES

Our research explores algorithms to generate schedules for the Astro-TSP. As mentioned in Chapter 2, the requirements that make-up Astro-TSP were set for a potential future implementation for the NGPS or use by other astronomers faced with the same scheduling problem. As such, we wanted to create an easy-to-use Python package that would allow astronomers to represent their observations and create schedules in a manner similar to Astroplan.

4.1: Research Problem

The Astro-TSP can be stated as follows: Given a set of observations to be performed by an observatory. Each observation having a time-dependent observation length, being only available within a fixed time window, and an assigned priority. Determine the ordered set or ordered subset of observations, which will first maximize the total priority gathered, then, if possible, minimize the total observing time.

Several aspects should be noted about the requirements for solving the Astro-TSP for potential future use at the Palomar Observatory. Excluded from the problem statement above is the need to consider stabilization periods, downloading times, and complex or nonlinear telescope slew rates. It is also important to note that targets with similar positions will often have similar ideal observation times during the night. However, this correlation is not absolute, allowing for exceptions. Despite not being absolute, this relationship is considered when developing a solution for Astro-TSP.

4.2: Research Objective

The purpose of this research is to develop an algorithm for scheduling observations at observatories. To do this we investigated a variety of scheduling techniques and implemented several algorithms for the Astro-TSP. For measuring the strength of a schedule, the 'best' schedule will maximize the total priority first and secondly minimize the total time spent observing. The key criteria are as follows:

- For a solution to fully consider all aspects of Astro-TSP, it must consider that each observation will have a:
    - Position
    - Time-Dependent Observation Length
    - Time Window of Availability
    - Priority

In addition to the requirements above, we start with the objective of keeping the runtime below 30 minutes when considering 65 observations. A fully deployed solution for the NGPS will require a runtime of only a few minutes and may need to rely on additional parallel computing. This work was designed to be used on a computer characterized by a single-socket machine, which is equipped with no more than 24 threads and 32GB of memory.

4.2: Research Challenges

While the Astro-TSP can mostly be represented as a combination of TSP variants, developing a solution to the Astro-TSP is not as straightforward. With no preexisting solutions, the development of a Simple Sort and Look Ahead Greed, which are discussed in Section 7.2.1 and 7.2.2 respectively were simple heuristic-based approaches. The first more advanced approach we attempted to solve for Astro-TSP was a linear programming approach like those presented in Section 3.2.1.

The solutions for the TSP based on B&C, MIP, and linear programming were shown to be promising and effective for both the TSP and several of its variants. This motivated us to consider Linear programming as a possible solution for Asto-TSP. To begin with, we chose to start with the formula from Kara [5] for the TSPTW. We successfully implemented the formulation for TSPTW using the Gurobi optimizer[4]. We were unable to adapt the formulation to consider time dependency and priority for Astro-TSP. This was due to several unique characteristics of the Astro-TSP. For Astro-TSP, not all observations must be visited. The formulations presented by Kara and Clímaco both constrain their respective problems using this aspect of the TSP; removing the requirement to visit every city significantly increases the search space. Additionally, removing the respective constraints from an ILP formulation may overrelax the mathematical model and introduce invalid solutions. Furthering the difficulties of using MIP or pure B&C method is that the time dependency of Astro-TSP removes any usefulness of sub-paths. This is because the same sequence of observations can take drastically different amounts of time depending on when the sequence is started.

Due to these problems, we ultimately made the choice not to continue efforts to create a complete formulation for the Astro-TSP. Furthermore, it was noted that with a linear programming approach adding additional features, such as the ability to schedule across multiple nights, may require a complete reformulation. This then led to the choice to consider using a metaheuristic, specifically a GA, as discussed in Section 6.1.

[4] Gurobi optimizer: https://www.gurobi.com/

CHAPTER 5:

TECHNICAL BACKGROUND

The development of any solution for the Astro-TSP requires a basic understanding of a few astronomy concepts. This is required to allow for the application of TSP solutions to the real-world requirements that are needed for Astro-TSP. The concepts essential for the Astro-TSP are described below in this chapter. Additionally, a general understanding of GAs is beneficial before attempting to formulate a GA for a specific problem such as the Astro-TSP. A brief introduction to GAs is also provided in this chapter.

5.1: Astronomy Basics

To develop a solution for the Astro-TSP several aspects of Astronomy should be understood at a basic level. For the purposes of Astro-TSP, we will state that each observation has a position. The position is provided with the Right Ascension (RA) and a Declination (DEC) of the target. If one were to draw latitude and longitude lines on the sky, RA is like the longitude. While DEC is like latitude, measuring how far north or south an object is from the celestial equator, with the north star at 90 degrees. As the Earth rotates targets appear to travel around the North Star on these latitude lines. The Hale Telescope has an equatorial mount allows it to rotate independently for RA and DEC. This allows us to treat RA and DEC like rectilinear coordinates X and Y for the purposes of calculating travel time between pointing. Figure 2 shows a set of positions for 21 observation positions.

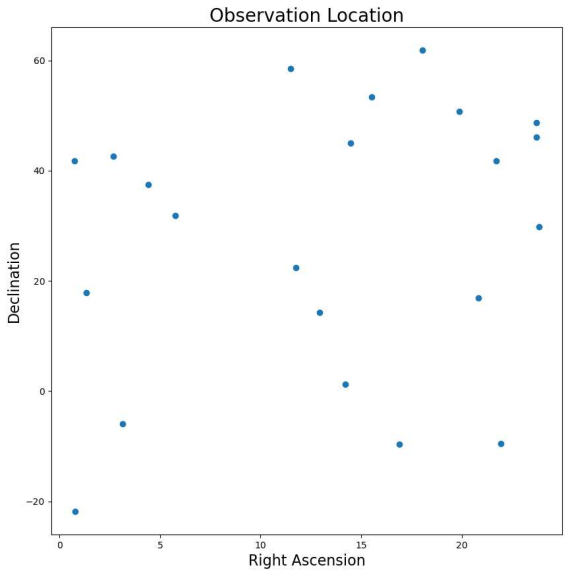


Figure 2: Sample Observation Locations

The slew time, which is the time the observatory takes to reposition the telescope between two positions, is calculated by assuming a simple linear movement speed in either direction. An additional settling time must also be added. This calculation, while not completely realistic, servers as good approximation for the scope of this project, and the current stage of development for the NGPS.

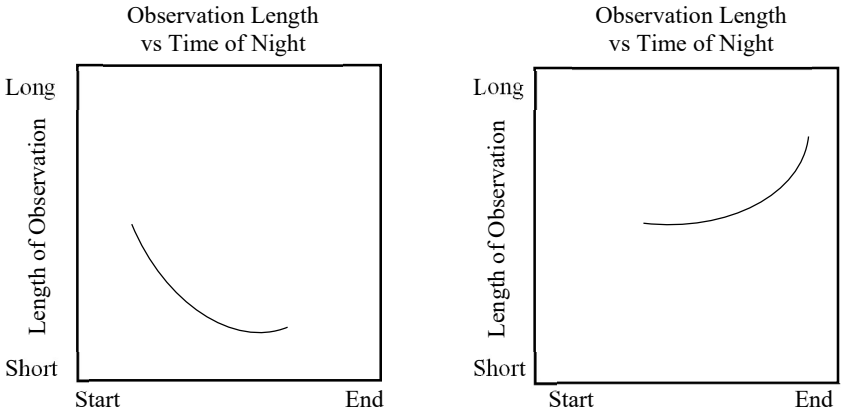


Figure 3: Two Possible Observation Length Requirements

As mentioned earlier and shown in Figure 3, the length of each observation is dependent on the time of night that the observation is started. This stems from each observation targeting a fixed signal-to-noise ratio that is set by the astronomer. "The faint signals must compete with background noise from the cosmos, the atmosphere, the earth's surface, and the detectors themselves" [1]. The detector in this case being an image sensor or instrument such as the NGPS. Of these variables the atmosphere can be controlled by scheduling the observation when it is ideally, directly above the head as shown in Figure 1. Determining the time required for achieving the desired fixed signal-to-noise ratio is considered outside the scope of the Astro-TSP and this research.

5.2: Genetic Algorithm

As mentioned in Chapter 3, a GA is an algorithm inspired by the theory of evolution. The following equivalencies in Table 1 may be helpful to understand better the relationship between traditional operations research in combinatorial optimization, genetic algorithms, and the Astro-TSP.

Table 1: Terminology [2]

Combinatorial Optimization	Genetic Algorithm	Astro-TSP
Encoded Solution	Chromosome	Permutation of all observations
Solution	Decoded Chromosome	Schedule of observations
Set of Encoded Solutions	Population	A set of permutations of all observations
Objective Function	Fitness Function	A method to evaluate how strong a schedule is.

Being inspired by the theory of evolution means a population of individuals is needed. Each individual in this population represents a possible solution. The specific solution they represent is encoded in their chromosome; for Astro-TSP this is a permutation of all

observations. A decoding method must be used to determine the solution an individual chromosome represents. A fitness function is used to assign a numerical score to each individual or chromosome. While not always required, in the case of Astro-TSP the chromosome must be decoded prior to performing the fitness calculation.

As seen in Figure 4, a GA is made of five key steps: population initialization, fitness evaluation, selection, crossover, and mutation. The stages of fitness evaluation, selection, crossover, and mutation are done once to the population for each generation.

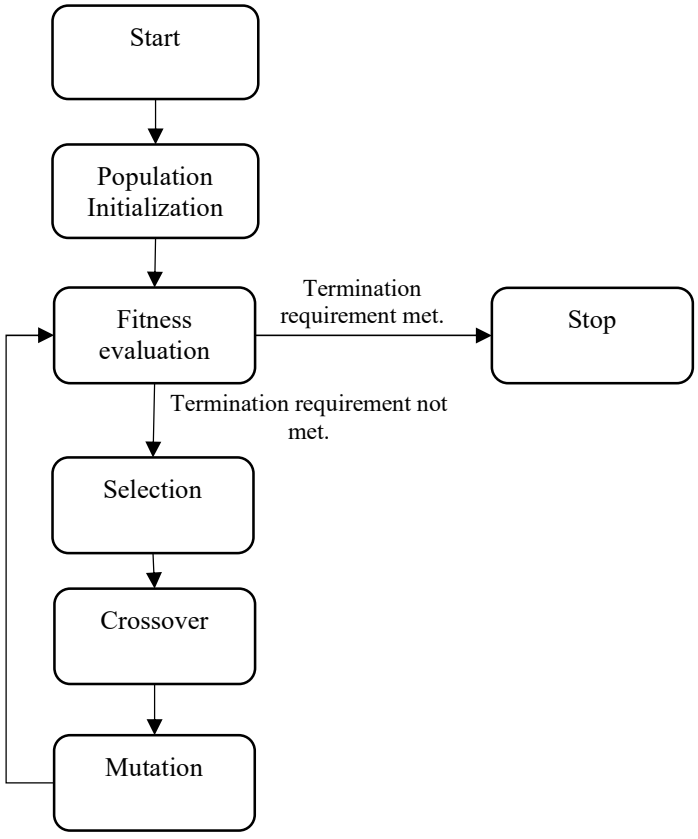


Figure 4: Genetic Algorithm Overview

Population initialization is the process of creating a set of initial individuals. Fitness evaluation is the process of assigning a fitness score to each individual. This fitness score is a

representation of how strong of a solution the individual represents. The fitness evaluation stage tends to be the most computationally expensive [11].

In the selection stage, the now calculated fitness values are used to determine which individuals will pass part of their chromosome onto the next generation via offspring. This is done by crossover; crossover takes two or more individuals that have been selected and combines them to create a new offspring that is hopefully stronger than the originally selected individuals. Potvin notes that special considerations must be made for TSP-like problems to ensure the chromosomes and thus the solutions remain a permutation [2].

The last step before the new individual can be put in the next generation is mutation. A mutation is applied randomly to some offspring after crossover. Mutation results in the random changing of the chromosome. This is done to help ensure diversity is not lost and to prevent convergence on a single solution from occurring too quickly. A new population is made by repeating these steps till an entirely new generation is formed, which should have individuals who on average have higher fitness scores and thus represent stronger solutions.

CHAPTER 6:

METHODOLOGY

For the Astro-TSP, two simple scheduling algorithms were first created, Simple Sort and Look Ahead Greedy; these two algorithms are discussed in Section 7.2.1 and Section 7.2.2 respectively. These two solutions serve as a baseline and provide a point of comparison due to their similarities with the scheduling algorithms used in Astroplan and the methods astronomers at the Palomar Observatory may use. As discussed in Section 4.3 linear programming proved too complex to fully represent the needed constraints for the Astro-TSP and formulate a solution. A GA was then created to meet the requirements for the Astro-TSP and produced satisfactory results. This then prompted a transition to the use of a GA which was our best performing algorithm as demonstrated in Chapter 7.

6.1: Genetic Algorithm

As discussed in Section 3.2.2, GAs are a versatile metaheuristic that have been shown capable of producing near optimal results for a variety of problems. However, they are not without their limitations. Due to their evolutionary nature, GAs are non-deterministic, and thus there is little chance of finding the most optimal solution. Additionally, it is possible to converge on a non-optimal solution early. Despite these drawbacks we chose to implement a GA as a method for solving Astro-TSP. As discussed in Section 5.2, a GA must be tailored specifically for the problem at hand. In this section we outline the formulation and reasoning for the GA we implemented for the Astro-TSP.

6.1.1: Encoding, Decoding, and Fitness Function

The first step in developing a GA for solving a real-world problem, such as the Astro-TSP, is to select an encoding strategy. Encoding is the representation of a solution as a chromosome. Potvin

explains in his survey that for encoding TSP-based problems, a path representation is a natural way to encode the solution [2]. While typically this would have the disadvantage of a single tour being represented by 2N practical solutions, this is not a concern for the Astro-TSP as a path is required. The path for Astro-TSP does not return to its starting node as required in the classic TSP. For this reason, our GA for the Astro-TSP used a path representation for encoding. A chromosome for the Astro-TSP was defined as a permutation of all observations the astronomers wish to consider. A decoding method is used to remove unperformable observations from this permutation. The logic for the decoding strategy can be seen in Figure 5.

DECODING ALGORITHM

```
    Input: Observation_List # An ordered  list of all Observations
    Output: Schedule # Ordered set or subset of Observation_List
    # Create an empty Schedule
1   Schedule
    # Loop Through each observation in Observation_List
2   For Obs in Observation_List:

        # Check if Observation can be performed
3       If can_perform_observation(Obs):
            # Append Obs to the schedule
4           Schedule.append(Obs)

            # Update time and telescope position
5           update_time_and_position(Obs)
    Return Completed Schedule
6   End
```

Figure 5: Decoding Algorithm

After decoding the fitness (F) of the now decoded, a solution was found by the following equation.

$$F = \sum \mathrm{P} + \frac{1}{T_o} \tag{1}$$

Where P is the priority for each observation, and T_o is the sum of all the observations lengths that are performed.

6.1.2: Population Initialization

Population initialization is the process of generating the first population. The objective of population initialization is to create enough diversity among individuals to ensure that the search space is sufficiently covered to prevent premature convergence. Population initialization can be done for the TSP by randomly ordering the cities to be visited. This method ensures the creation of a diverse initial population. Alternatively, methods like those discussed by Deng in Section 3.2.2 can be used. The initialization for the Astro-TSP GA employs a special process to utilize the relationship between the position of observation and the ideal time of night. The population initialization is performed by creating two initial chromosomes, one by ordering the observations by their ideal observation time, and the other by using a modified greedy algorithm. The two chromosomes then have their fitness values calculated and the one with the higher fitness score is selected as the base chromosome. The base chromosome is then duplicated, and a local shuffling algorithm is applied to these copies to introduce diversity to the population. This local shuffling is controlled by the initialization shuffle ratio from

Table 3 where a ratio of 0 means no shuffling occurs and a ratio of 1 would lead to a completely random population.

6.1.3: Selection

With a population of individuals having their fitness values determined. Selection is the next stage in a GA. Selection is the process of deciding which individuals will pass their genetic information on to the next generation. A range of selection strategies exists for GA-based solutions to the TSP. each selection strategy has pros and cons. [12] Tournament selection is one of the

simplest to implement. To perform tournament selection, we choose a random subset of individuals and select the one with the highest fitness. By controlling the subset size, the convergence rate can be controlled. A larger tournament will lead to convergence on a common solution faster than a smaller tournament size. Tournament selection was also found to be computationally the quickest, requiring the least amount of time per generation [12]. However, if the only concern is maximizing the best individuals' fitness, ranked selection may result in better results. After two individuals have been selected, they can be used in crossover.

The Astro-TSP tournament selection was used as the selection strategy. This was done for its simplicity and the ability to control the convergence rate by simply adjusting the tournament size. The use of tournament selection also helps to minimize run time. Additionally the Astro-TSP uses elitism. Elitism is the process of selecting the best individual at the start of selection, and placing a copy in the next generation's population. This process ensures the most fit individual, and thus the best schedule is never lost due to not being selected.

6.1.4: Crossover

After we select two individuals to be parents, we can create an offspring to be an individual in the next generation. This is done through a process known as crossover. As discussed by Potvin, for TSP GAs that use a path-like encoding, special considerations must be made to ensure that the chromosomes remain a permutation [2]. To do this for the Astro-TSP, a modified one-point crossover method was used. As shown in Figure 6, a modified one-point crossover requires two parents' chromosomes and a random slicing point to be selected. The offspring then receives the front segment of the Parent 1s chromosome and the back of the Parent 2s chromosome. Due to this, the created offspring's chromosome may not be a permutation. To prevent this, an additional operation must be performed to correct the chromosome. The complete crossover algorithm is

provided in Figure 7. A second offspring can also be generated with the same parents by selecting a new slice point and alternating the parts of the parents' chromosomes the child is receiving.

Parent 1	:	**1**	**2**	| **3**	**4**	**5**
Parent 2	:	1	3	| 2	5	4
Offspring	:	**1**	**2**	2	5	4
Corrected offspring	:	**1**	**2**	3	5	4

Figure 6: Modified One-Point Crossover

CROSSOVER ALGORITHM

```
    Input: Parent_A_Observation_List
           Parent_B_Observation_List
    Output: Offprint_Observation_List
    # Select Crossover point
1   crossover_point = random_integer(0, length(Parent_A_Observion_List))
    # Assign Offspring each side of parent
2   Offspring_Observation_List = Parent_A_Observation_List[:crossover_point] +
                                 Parent_B_Observation_List[crossover_point:]
    # Correct offspring to ensure permutation
3   for index from 0 to length(Parent_A_Observation_List) - 1:
4       if Parent_A_Observation_List[index] not in Offspring_Observation_List:
5           Offspring_Observation_List[index] = Parent_A_Observation_List[index]
    Return Offspring_Observation_List
7   End
```

Figure 7: Crossover Algorithm

6.1.5: Mutation

Mutation is the last stage of the GA. Mutations are not performed on every offspring but instead only occur based on a pre-set mutation probability. This prevents convergence on a local minima or maxima by randomly altering the chromosome of an individual. This also reintroduces genetic diversity that may have been lost in previous generations. Compared to other GA applications, solutions for the TSP are often significantly modified compared to the original tour after a mutation occurs [2]. The Astro-TSP's GA implementation uses a modified swap mutation.

First, a gene is randomly selected, and then a second gene is selected within the maximum swap range (m_s). This maximum swap range is added to encourage observations to remain near their optimal observation times.

Individual Before Mutation	:	1	2	3	4	**5**	6	7	8	9	10
Considered Positions	:	1	2	**3**	4	**5**	6	7	8	9	10
Individual Post Mutation	:	1	2	**5**	4	**3**	6	7	8	9	10

Figure 8: Modified Swap Mutation, $m_s = 2$

6.1.6: Parallelization

Being population-based, GAs are inherently well-suited for parallel computing [2]. GA parallelization can be implemented in several manners. In one potential implementation, described by Whitney *et al*. the distributed algorithm GA uses subpopulations, each performing a typical genetic search except that copies of the best solutions are passed to neighboring subpopulations at regular intervals [13]. With this implementation, Whitney *et al*. solved an instance of the TSP with 105 cities to the known optimal solution in 15 out of 30 attempted runs. The 15 remaining runs reached schedules within 1% of the known optimal.

The GA implantation for the Astro-TSP also utilized a distributed processing system. Specifically for our implantation, this was done with the Python concurrent futures class utilizing a processor worker pool. Each process that is available is seen as one worker in this pool and can be run in parallel with one another, permitting the necessary hardware to be present. This allowed for easy interoperability between Windows and Linux based machines. At the start of the GA, an initial global population is created using the methods outlined earlier in Section 6.1.2. Each subpopulation simultaneously goes through the genetic algorithm process and are later combined to form a new improved global population. This new global population can then be improved again, or the most fit individual schedule can be used.

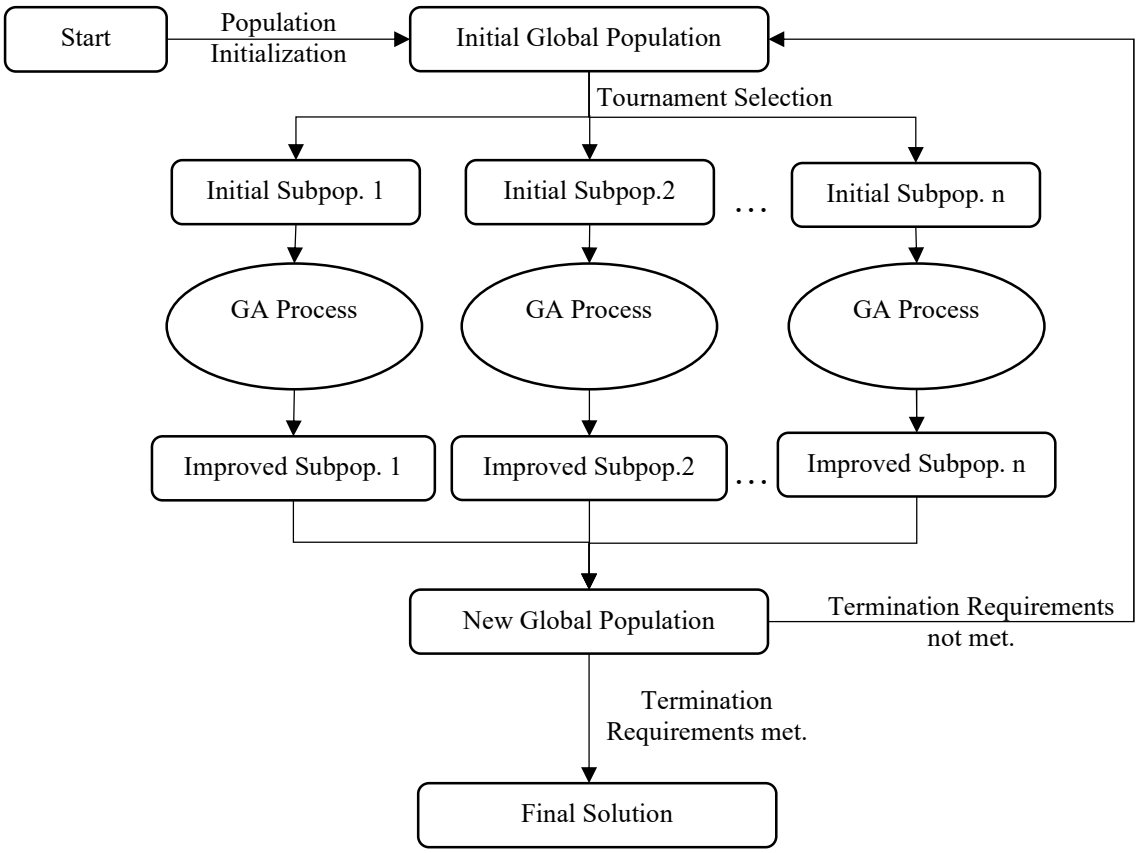


Figure 9: Astro-TSP Genetic Algorithm Parallelization Method

CHAPTER 7:

EXPERIMENTAL VALIDATION

This chapter presents the experimental validation process for the developed GA. Before discussing the results, we discuss the experimental setup processes. We then discuss how we will be evaluating the schedules to determine the effectiveness of the GA. Two algorithms are also introduced, which serve as a point of comparison and are presented.in the following chapters.

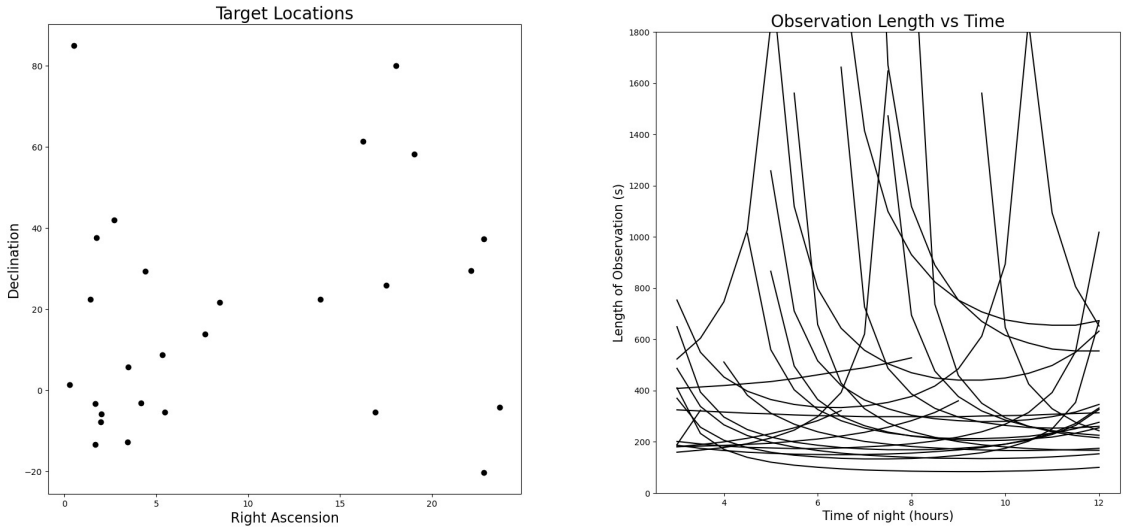
7.1: Experimental Setup

The experimental setup for this research and the unique real-world requirements were based on the needs of the NGPS. As mentioned in Chapter 5, observation lengths are calculated for a fixed signal-to-noise ratio. The specific target lists are examples from Zwicky Transient Factory[5], a current instrument at the Palomar Observatory. Observation times were calculated with the NGPS Exposure Time Calculator and provided by C. Shapiro (private communications). Each observation is also assigned a priority, and if the observation needs to be repeated, a repeat value can be provided as well. Repeats occur directly after one another. Both the priority and repeat values have a default and minimum value of one.

Table 2: Sample Observation Data for Single Observation

Name	RA	DECL	Observation Start Time	Required Observation Length (s)
ZTF22aaykjqo	5.77	31.92	2022-08-07T10:00:00	1009.04
ZTF22aaykjqo	5.77	31.92	2022-08-07T10:30:00	513.97
ZTF22aaykjqo	5.77	31.92	2022-08-07T11:00:00	355.45
ZTF22aaykjqo	5.77	31.92	2022-08-07T11:30:00	280.07

[5] Zwicky Transit Factory: https://www.ztf.caltech.edu/



(a) (b)

Figure 10: Sample Provided Observation Positions (a) and Time Requirements (b)

While the target positions provided by the Right Ascension (RA) and Declination (DECL) in Table 2 are not unique to the Palomar Observatory, the observation lengths are unique to the specific target and the NGPS. Additionally, the slew rate of the observatory is the time required to reposition the telescope between observations, which is unique to Palomar Observatory. Waiting periods were also implemented to account for the settling of the telescope after movement and downloading time of observational data.

No specific machine specifications were provided. As such, the code was written in a manner that is compatible with a variety of most modern hardware. For our testing and results the machine featured an AMD Ryzen 9 3900X 12-Core Processor and 32GB of DDR4 memory. The parameters used for the Genetic Algorithm can be found in

Table 3: Genetic Algorithm Configuration. A total population of 5,000 individuals were used for our testing purposes. Ten global improvements of 100 generations proved sufficient to converge on a solution for even our most complex problems.

Table 3: Genetic Algorithm Configuration

Parameter	Value	Description
Sub-population size	500	The number of individuals in a subpopulation
Number of subpopulations	10	The number of subpopulations
Number of Generations	100	The number of generations each subpopulation is improved per global improvement cycle.
Global Improvement Cycles	5	The number of global population improvements
Initialization shuffle ratio	0.3	The ratio of allowed shuffle distance to chromosome length
Tournament selection ratio	0.005	The ratio of population size to tournament size
Global tournament selection ratio	0.005	The ratio of the global population to tournament size
Mutation probability	0.2	The probability that an individual will be mutated
Mutation swap distance ratio	0.2	The ratio of allowed swap distance to chromosome length

7.2: Schedule Evaluation Performance Metrics

To evaluate a schedule, we consider two key metrics. First, is the amount of priority that the schedule gathers. Per the problem definition we aim to maximize this value before any other aspects of a schedule. Secondly, we consider the time spent observing. This is equivalent to equation (1), the fitness function used in the GA, and aligns with the problem requirements outlined in Section 4.1. While other factors such as the number of unique observations, time spent waiting, time spent repositioning, and the total length of the schedule could have been used in measuring the performance of a schedule, this would have shifted the problem to a multi-objective optimization problem and added further complexity. To provide a point of reference for our GA we also created two additional basic algorithms. These methods are Simple Sort (SS), and Look Ahead Greedy (LAG).

7.2.1: Simple Sort

The Simple Sort (SS) is the most intuitive of the algorithms and is similar to the sequential sort included in Astroplan. The SS algorithm orders the observations by their ideal observation time. The ideal observation time is the time at night when the observation has the shortest observation length. To handle the fact that not all observations will be possible in this order, the list is pruned by removing those that could not be performed due to their unique time windows. This pruning process is the same as the decoding algorithm for the GA presented in

Figure 5. The SS is not expected to perform the best due to zero consideration of priority being given. However, for problems in which all observations can be scheduled, the SS can still be effective.

7.2.2: Look Ahead Greedy

The look-ahead-greedy method (LAG) was developed as a method that would consider the priority. For the LAG method, at the start of the schedule, and after each observation, a value (V) is determined for all remaining observations:

$$V = \frac{P}{T_r} \tag{2}$$

Where P is the priority of the observation, and T_r is the time to travel to and complete the observation. To prevent excessive waiting time for observations that have high value, a look-ahead time is introduced. This look-ahead time limits the observations that can be scheduled next by requiring the observation to be available to start within the look-ahead time. If no observations are available, the look ahead time is increased. When one or more observations are found within the look-ahead time, the observation with the highest value is scheduled. The algorithm's logic can be seen in Figure 11: Look Ahead Greedy algorithm.

LOOK AHEAD GREEDY ALGORITHM OVERVIEW[6]

```
    Input: Observation_List   # A list of all Observations
           Look_Ahead_Time # The time the algorithm should look ahead
    Output: Schedule          #Ordered set or subset list of Observation_List

1   # Create empty Schedule and Available_Ahead Lists
2   Schedule, Available_Ahead

    # Where there are observations available
3   While length(Observation_List) > 0:
        #Check Availability of each observation
4       For Obs in Observation_List:
            #If no longer possible, remove from Observation List
5           If observation_is_impossible(Obs)
                Observation_List.remove(Obs)

            #If the observation will be available in look ahead time, append it
6           If Observation_avalible_in_time(Obs, Look_Ahead_Time)
7               Available_Ahead.append(Obs)
        #Select Max Value Obs and append to schedule
8       Max_Value_obs ← determine_max_value_obs(Available_Ahead)
9       Schedule.append(Max_Value_Obs)
10      Observation_list.remove(Max_value_obs)

        #Update time and telescope position
11      update_time_and_position(Obs)
    Return Completed Schedule
12  end
```

Figure 11: Look Ahead Greedy algorithm.

7.3: Results

As discussed previously, no algorithm for the Astro-TSP currently exists. This facilitates the need to create sample problems to simulate a range of possible real-world scenarios. Five problems were created and solved using the SS, LAG, and GA method. Due to not fully formulating a linear programming solution, no comparisons can be made to linear programming as a solution for Astro-TSP.

[6] Look-Ahead-Greedy Algorithm Overview excluded handling of increasing and resetting the look-ahead-time.

With an original problem size of 65 observations being expected for possible use for the NGPS, A1 at 78 observations most closely represents the original target size. A2 represents the effects of having variance in the priority among the observations. For problem A2, observations had their priority assigned randomly to be in the range 1-3. Problem A3 represents the effects of undersubscription; undersubscription occurs when all observations can be completed with significant time to spare. This creates a scenario that would be easier to solve by hand due to the significantly smaller search space. Problem A4 represents the effects of oversubscription. Oversubscription occurs when there are many more observations compared to what can be scheduled, this results in schedules that only contain a subset of the observations that were provided to be scheduled. A5 represents another case of oversubscription with the addition of variance in priority. Observations had their priority randomly assigned between 1 and 10. Additionally the observation lengths were increased by a random amount between 0s and 600s. For all instances, the Simple Solver (SS) and the Look-Ahead Greedy (LAG) algorithms produced schedules in less than one second. Due to the non-deterministic nature of GAs, the proposed GA solved the same problem instance five times, and an average is presented.

Table 4: Astro-TSP Solution Comparison

Problem Instances			Simple Solver			Look Ahead Greedy		
Name	Total Unique Observa-tions	Total Priority	Unique Observa-tions	Priority	Observing Time (s)	Unique Observa-tions	Priority	Observing Time (s)
A1	78	78	75	75	29233	68	68	28901
A2	93	191	88	181	30615	79	171	28977
A3	27	27	27	27	22124	27	27	25482
A4	168	168	96	96	3469	121	121	30092
A5	137	431	58	200	33924	78	292	32408

Problem Instances			Genetic Algorithm			
Name	Total Unique Observations	Total Priority	Unique Observations	Priority	Observing Time (s)	Run Time (s)
A1	78	78	78 ± 0	78 ± 0	24014.4 ± 531.8	762.8 ± 8.14
A2	93	191	93 ± 0	191 ± 0	24514.9 ± 13.6	886.6 ± 1.05
A3	27	27	27 ± 0	27 ± 0	9348.43 ± 1.35	267.5 ± 0.66
A4	168	168	145.2 ± 0.8	145.2 ± 0.8	31945.0 ± 280.9	1415 ± 2.4
A5	137	431	86.4 ± 0.6	336.8 ± 1.8	32899.0 ± 52.8	922.63 ± 5.6

For problem A1, our developed GA was the only solution that successfully scheduled all 78 unique observations, thus gathering the most priority. The GA did so while requiring the least observing time as well. On problem A2, which introduced priority, the GA was once again the only method capable of scheduling all the observations, and thus achieved the highest priority again. In both problems A1 and A2 the SS did perform better than the LAG. This is due to the LAG choosing to wait for observations that have a higher value. These waiting times prevent more observations from being performed.

Problem A3 measured the effects of undersubscription. All three algorithms were able to schedule all the observations and thus gathered the same amount of priority. Despite this the GA

still produced the best schedule. The GA only required a total of 9,348s (about 2 1/2 hours) to be spent observing in order to complete all the observations. This is less than half of what the SS, and the LAG both required over 20,000s (about 5 1/2 hours). While this significant difference may seem strange it is expected. Both the SS and LAG create the schedule from the start of the night looking forward. This led to both the SS and LAG to schedule an observation as soon as it was possible despite the longer observation length compared to waiting. This could be considered a flaw in the scheduling methodologies; however, this effect only appears when there is an undersubscription. Additionally, this could be prevented by limiting the time windows of the observations to ensure they are scheduled closer to their ideal observation time. In contrast, the GA creates the entire schedule at once. This allowed the same observations to be pushed later by the introduction of a long waiting period. This long waiting period resulted in a superior schedule.

Problem A4 represents the effects of oversubscription. Problem A4 has 168 total observations each worth one priority. For this case, the GA once again outperformed the SS and LAG. The GA was able to schedule an average of 145.2 observations, while the SS and LAG scheduled 96 and 121 observations respectively. The effects of oversubscription are also observed in problem A5. Problem A5 consisted of only 137 observations for a total priority of 438. Observations were also made artificially longer. For this case the SS gathered 200 priority, the LAG 292 priority, and the GA 336.8 priority.

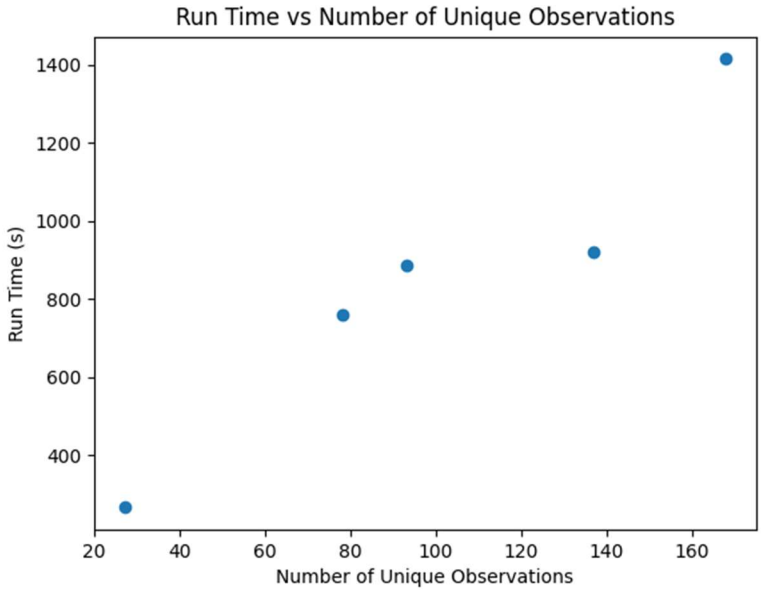


Figure 12: Run time vs Number of unique observations for GA

As demonstrated above, the GA regularly produces schedules that gather more priority or equal priority with less total observing time. This does come with one drawback. The GA takes substantially longer than the SS and LAG. The length of time is dependent on many factors, mainly the number of individuals in the population, the number of generations, and the number of observations available to schedule. While the number of individuals and the number of generations were kept constant, each problem did have a unique number of observations. As seen in Figure 12 there is a near linear relationship between the number of observations and the run time.

Table 5: Parallelization Analysis

Available Workers	Run Time (s)
24	1415 ± 2.4
10	1537 ± 3.0
8	2603 ± 95
5	2700 ± 103
2	6329 ± 157

For the results presented in Table 4, no limit on the number of workers used in the concurrent futures workers pool was used. Limiting the number of workers, like limiting the number of processes available, allows us to see how effectively the parallelization methods work. Table 5 shows the effects of limiting the number of works on problem A4. This problem was computationally the most expensive due to having 169 unique observations.

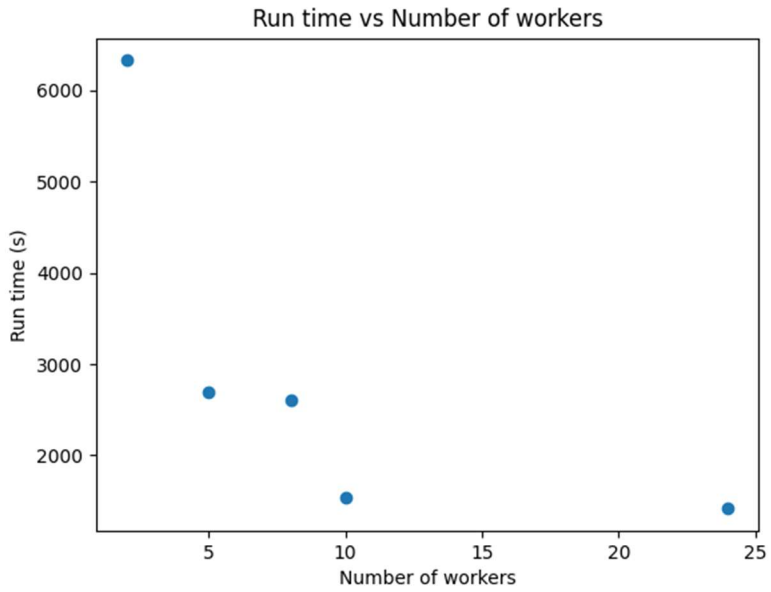


Figure 13: Run time vs Number of workers.

While it is expected that increasing the number of workers reduces the run time, it might seem unusual that we see little benefit from changing the number of workers from five to eight, as seen in Figure 12. However, this is expected. As seen in Table 3, the number of subpopulations was set to 10, and each subpopulation was improved for 100 generations. In the case of two workers, each worker will solve five subpopulations. For five workers, each worker must solve two subpopulations. For eight workers, six workers must solve one subpopulation, and two workers will be required to solve two subpopulations. Then, at ten workers, we see a further reduction as each process is then only required to solve a single subpopulation. Increasing

the number of workers beyond that yields little improvement as expected. The proposed parallelization methods had an effective speed-up of 4.47x when increasing the number of works available from two to ten.

CHAPTER 8:

CONCLUSION

The objective of this research was to create a solution for scheduling observations at an observatory. Motivated by the desire to increase the impact of the NGPS at the Palomar Observatory, we defined the Astro-TSP problem. We then considered a variety of solutions and implemented three: the SS, LAG, and GA. The GA required special considerations for the Astro-TSP and was designed to utilize the natural order of observations. Despite the non-deterministic nature of GAs, the implemented GA for Astro-TSP regularly outperformed the SS and LAG methods with the drawback of longer run times. We were able to minimize the required run time via parallel computing. While a deterministic solution may exist that outperforms our developed GA, we believe the implementation of our algorithm into an astronomer's workflow when working with the NGPS can help lead to more data collection and benefit the scientific community. The NGPS developers and some users are aware of these results and have shown interest in the future implementation of a solution for Astro-TSP as part of the NGPS software tools when appropriate funding and workforce can be identified.

8.1: Future Work

While our developed GA was shown to be effective for the Astro-TSP, we do acknowledge a deterministic solution may be desired, and as such, resuming efforts to formulate the problem via linear programming could yield a solution. , If this is not possible, improvements can still be found in our GA. Improvements to the GA could come in two forms. Considerations of alternative methods for initialization, selection, crossover, and mutation. These alternative methods could lead to the creation of better schedules. The GA can also have additional parallelization techniques applied. While our current GA does use parallelization, additional

performance could be found by utilizing a Graphical Processing Unit (GPU) to perform all fitness calculations in parallel. As noted previously, fitness calculations are the most computationally expense and thus may lead to a significant reduction in runtime if solved with a GPU.

Future work can also be performed by furthering the scope of the problem. Additional complexities could be added to better assist astronomers. These include, but are not limited to, multiple night scheduling, larger problems, generation of time dependency data, and more.

REFERENCES

[1] H. Bradt, Astronomy Methods A Physical Approach to Astronomical Observations, New York: Cambridge University Press, 2004.

[2] J. Potvin, "Genetic algorithms for the traveling salesman problem," *Ann Oper Res,* vol. 63, pp. 337-370, 1996.

[3] A. Punnen and G. Gutin, The Traveling Salesman Problem and Its Variantions, Kluwer Academic, 2002.

[4] E. Baker, "An exact algorithm for the time-constrained traveling salesman problem," *Operations Research,* vol. 31, no. 5, pp. 938-945, 2007.

[5] I. Kara and T. Cerya, "Formulations for minimizing tour duration of the traveling salesman problem with time windows," *Procedia Economics and Finance,* no. 26, pp. 1026-1034, 2015.

[6] G. Cl´ımaco, L. Simonetti and I. Rosseti, "A Branch-and-cure and MIP-Based Heuristis for the Prize-Collecting Travelling Salesman Problem," *RAIRO Operations Research,* vol. 55, pp. S719-S726, 2021.

[7] W. Deng, J. Xu and H. Zhao, "An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem," *IEEE Access,* vol. 7, pp. 20281-20292, 2019.

[8] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation,* vol. 1, no. 1, pp. 53-66, 1997.

[9] G. Fred and L. Manuel, "Tabu search I," *ORSA Journal on Computing,* 1999.

[10] Y. Deng, Y. Liu and D. Zhou, "An improved genetic algorithm with initial population strategy for symmetric TSP," *Mathematical Problems in Engineering,* pp. 1-6, 2015.

[11] M. Ca'mara, J. Ortega and F. Toro, "Approaching Dynamic Multi-Objective Optimization Problems by Using Parallel Evolutionary Algorithms," *Studies in Computational Intelligence,* vol. 272, pp. 63-85 , 2010.

[12] M. Razali and J. Geraghty, "Genetic Algorithm Performance with Different Selection Strategies in Solving TSP," in *World Congress on Engineering*, London, U.K., 2011.

[13] D. Whitley, T. Starkweather and D. Shaner, "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination," 2002.

[14] P. Hungerländer and C. Truden, "Efficient and easy-to-implement mixed-integer linear programs for the traveling salesperson problem with time windows," *Transportation Research Procedia,* no. 30, pp. 137-166, 2018.