POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Path Planning Algorithm for South Pole Lunar Rover Missions

FENG YANG CHEN
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Décembre 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Path Planning Algorithm for South Pole Lunar Rover Missions

présenté par **Feng Yang CHEN**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

DEDICATION

To my family, my biggest source of motivation and inspiration...

ACKNOWLEDGEMENTS

Je souhaite tout d'abord remercier professeur Giovanni Beltrame pour son mentorat et sa grande flexibilité pour m'avoir accomodé.

Merci également au Dr. Marie-Josée Potvin pour avoir continué à croire en mes capacités malgré un début turbulent.

Enfin, merci à ma famille et à Mlle Shao-Shang Ly (Celina) pour leur appui inconditionnel à travers cette aventure.

RÉSUMÉ

Les astromobiles lunaires auront un rôle crucial pour le retour de l'humanité sur la Lune. En effet, l'environnement lunaire est hostile, hasardeux et dangereux pour les humains. Ainsi, envoyer des astromobiles est une solution viable pour son exploration. avec le Programme d'accélération de l'exploration lunaire (PAEL), l'Agence spatiale canadienne (ASC) supporte l'initiative de retour sur la Lune en offrant divers contrats pour développer les technologies spatiales. Un livrable de ce programme est un astromobile lunaire qui sera lancé au pôle sud de la Lune. Cette zone particulière contient des éléments de réponse en lien avec la présence d'eau et avec la formation du système solaire. Ainsi, l'envoi d'un astromobile lunaire aura des retombées scientifiques importantes tout en minimisant le danger d'exposition aux humains.

Malheureusement, la planification de trajectoire pour un astromobile lunaire est difficile et complexe. Tout d'abord, la topographie de la Lune n'est pas propice à la navigation. Elle contient des pentes abruptes qui font en sorte que certaines régions ne sont pas accessibles. De plus, le mouvement relatif entre la Lune et le Soleil, et entre la Lune et la Terre génèrent des zones qui varient dans le temps sur la Lune où l'astromobile n'aura pas accès au Soleil ou à la Terre. Un astromobile lunaire devrait toujours rester dans des zones où ce dernier est exposé au Soleil pour charger ses batteries et en contact avec la Terre pour la communication de commande. Pour ce faire, des données existantes sur la topographie lunaire fournies par la National Aeronautics and Space Administration (NASA) seront utilisées. L'ASC a généré, à partir de ces données, une banque d'information sur les obstacles dynamiques représentant l'état du Soleil et de la Terre par rapport à la Lune. En combinant cette dernière avec la topographie, un modèle adéquat de l'environnement lunaire est généré.

Additionnellement, l'astromobile lunaire recevra de nombreux points de cheminement à explorer avec un temps limite. Par conséquent, le problème peut être décomposé en deux étapes: premièrement, l'ordre d'exploration doit être trouvé et deuxièmement, avec cet ordre, une trajectoire faisable et sans obstacle sera générée. Le travail de ce mémoire propose un outil de planification de trajectoire pour un astromobile allant au pôle sud de la Lune en résolvant précisément ce problème. L'outil contient deux composantes: le *2-step genetic algorithm* et le *Multi-Directional Rapidly-exploring Random Trees (MD-RRT*)*. La première composante est une variante de l'algorithme génétique qui retourne un ordre d'exploration pour les points de cheminement donnés en entrée. Cette dernière maximise la pertinence scientifique de la trajectoire potentielle et minimise sa distance parcourue. La deuxième composante reprend l'ordre et s'assure qu'une trajectoire faisable existe en tenant compte des obstacles statiques et dynamiques en lien avec la topographie lunaire, l'accès au Soleil et l'accès à la Terre.

Puisqu'il n'y a pas de comparatifs disponibles actuellement pour l'outil de planification, le meilleur moyen de valider ses résultats est de comparer avec des trajectoires générées manuellement par des opérateurs de l'ASC. Ces opérateurs ont planifié au meilleur de leurs habiletés et leurs résultats sont utilisés pour évaluer l'outil. En général, l'outil peut trouver des trajectoires plus courtes en termes de distance et de temps de traversée beaucoup plus rapidement que les opérateurs. De plus, ce genre de comparaison a été effectué sur différents sites lunaires, ce qui démontre le potentiel de cet outil pour des missions futures. Malgré cela, plusieurs limitations et aspects de recherche futurs sont identifiés pour améliorer la robustesse de ce planificateur pour des missions d'astromobiles plus complexes et avancées.

ABSTRACT

Lunar rovers will play a crucial role in the initiative of returning humans to the Moon. This is due to the harsh environmental conditions making it hazardous and dangerous for humans. Thus, sending rovers is a viable solution for exploration purposes. Following the Lunar Exploration Accelerator Program (LEAP), Canadian Space Agency (CSA) is supporting the return of humanity by giving out various space contracts for developing spatial technologies. One deliverable of the LEAP is a lunar rover that will go on the South Pole of the Moon. This particular area contains key answer elements about the presence of water and the history of the formation of the solar system. Therefore, sending a rover there will have important scientific relevance while minimizing human exposure.

Unfortunately, lunar path planning for a rover is complex and no easy feat. Firstly, the topography of the Moon is not exactly "navigation-friendly". It contains steep slopes and rough edges which makes some areas impossible to access. On top of that, the relative motion of the Moon with respect to the Sun and with respect to Earth creates areas that vary through time on the Moon where there is no access to the Sun or the Earth respectively. A lunar rover should always stay in zones where it can have exposure to sunlight for charging and have contact with our planet for communication purposes. To do so, existing lunar topography datasets provided by NASA will be used. CSA processed those datasets to generate dynamic obstacle information. By combining both, an adequate lunar environment model is created.

Furthermore, the lunar rover will receive multiple waypoints to explore given a certain time limit. Thus, the problem becomes two fold: first, the order of exploration needs to be decided, and second, given the order, a feasible obstacle-free path will be generated. This work proposes a path planner tool for a lunar rover going to the South Pole of the Moon by solving precisely this problem. This tool has two components: the 2-step genetic algorithm and the Multi-Directional Rapidly-exploring Random Trees (MD-RRT*). The first component is a variant of the Genetic Algorithm which proposes an order of exploration for the given waypoints. It focuses on maximizing the science score and minimizing the total distance traveled of the resulting trajectories. The second component then takes the order and ensures that a feasible trajectory can be found by taking into account all the static and dynamic obstacles (topography, access to the Sun and access to Earth).

Since there are no tools available publicly at the time of writing this thesis that does something similar, the best way to compare is with manual operators from CSA. Indeed, they manually computed trajectories to the best of their abilities, and their results served as a benchmark for the tool. Overall, the latter can generate in a much shorter time trajectories that are shorter in terms of time and distance. Additionally, comparisons were done on different lunar sites, which goes to show that this tool holds promising potential for the future of space exploration. Despite that, several limitations and future research areas can be identified to ensure that this tool will be robust enough for more complex and advanced space rover missions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

ACO	Ant Colony Optimization
ASC	Agence spatiale canadienne
B&B	Branch and Bound
B&C	Branch and Cut
B-RRT	Bidirectional Rapidly exploring Random Tree
B-RRT*	Bidirectional RRT*
CSA	Canadian Space Agency
DEM	Digital Elevation Model
DP	Dynamic Programming
HiRISE	High-Resolution Imaging Science Experiment
IB-RRT*	Intelligent B-RRT*
ILP	Integer Linear Programming
IMOMD-RRT*	Informable Multi-Objective and Multi-Directional RRT*
ISE	Incremental Search Engine
LEAP	Lunar Exploration Accelerator Program
LOLA	Lunar Orbiter Laser Altimeter
LP	Linear Programming
LRM	Lunar Rover Mission
LRO	Lunar Reconnaissance Orbiter
MFA*	Multi-cost Fast A*
MIP	Mixed Integer Programming
MOLA	Mars Orbiter Laser Altimeter
MST	Minimum Spanning Tree
NASA	National Aeronautics and Space Administration
PSO	Particle Swarm Optimization
PSR	Permanently Shadowed Regions
PAEL	Programme d'accélération de l'exploration lunaire
RRT	Rapidly-exploring Random Trees
RRG	Rapidly-exploring Random Graph
r-TSP	relaxed Traveling Salesman Problem
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem

CHAPTER 1 INTRODUCTION

Nowadays, the initiative of returning humans on the Moon is quite popular. With the announcement of the Artemis missions, the Moon became once again an objective for humans to land on. Additionally, this time, humans will attempt to settle on Earth's natural satellite in an attempt to create a gateway for future crewed missions reaching even further beyond [1].

1.1 Context and Basic Concepts

Canada will play an active role for this initiative by providing *Canadarm3*, the third generation of the robotic arm for the Gateway lunar station and by providing financial support through the LEAP initiative. The latter has many deliverables, including a Canadian lunar rover that is planned to be launched on the South Pole of the Moon, tentatively in 2027 [2]. This initiative is called the Lunar Rover Mission (LRM).

1.1.1 Lunar Environment

Sending a rover on the Moon is no easy feat. The lunar environment is extremely harsh compared to our friendly Earth environment [3]. Among the numerous environmental challenges, this paper will tackle the ones related to the static lunar topography and the dynamic lunar environment on the South Pole.

The surface of the Moon is quite rough. Some areas may be steep and inaccessible. Fortunately, with the Lunar Reconnaissance Orbiter (LRO) mission, the Lunar Orbiter Laser Altimeter (LOLA) dataset was collected, and it became possible to model the surface of the Moon [4]. This dataset will help the rover navigate the lunar environment.

Furthermore, the LRM is a mission planned specifically on the South Pole of the Moon. This is due to the presence of Permanently Shadowed Regions (PSR) which may contain important historical evidence of the Moon's thermal state and geological processes, not to mention the presence of water [5]. Unfortunately, access to the Sun for power and to Earth for communication will be limited and will be varying through time depending on the relative positioning of the Moon, the Sun and the Earth [4,6]. This will be quite problematic for the LRM, since it needs access to both to complete long-term missions.

Similarly to the static topography of the Moon, the dynamic environment of the Moon's South Pole can be modeled with the ray-tracing technique [5,6]. With that information, it becomes possible to ensure that the rover will stay in areas with access to the Sun and Earth.

1.1.2 Multi-Directional Path Planning

As mentioned above, there are numerous areas of interest on the South Pole of the Moon. Conventional path planning from point A to point B may not be appropriate anymore for a lunar rover. Instead, a multi-directional path planning approach is more efficient.

For precision, this thesis refers to multi-directional path planning as the process of planning trajectories through multiple waypoints [7]. Some may interpret this as planning a trajectory in multiple dimensions, which simply implies that the rover (or agent) can go in multiple directions [8], but that is not the case here.

1.1.3 Ordering Waypoints

When given multiple interesting areas to explore, one thing to consider before starting the path planning is the order of exploration. The LRM will have many zones of scientific interest to explore in the lunar South Pole with a finite amount of time; therefore, one challenge is to determine the order of exploration while respecting the maximum operation time.

Assuming that all the different waypoints are of equal scientific interest (the waypoints are all of equal priority), the factors that need to be considered for computing the order of exploration are the accessibility, the traverse time, and the stopping time.

Accessibility is if the specified location is available at the time of exploration. Recall that the lunar environment is dynamic; thus, a zone of scientific interest may have sunlight and access to Earth at one point in time, but not later (or earlier). This implies that every waypoint will have a time window in which exploration is possible. Anytime outside that time window means that the location is missing either sunlight or a direct access to Earth communication.

Traverse time represents the time it takes to go from the current location to the specified waypoint. Obviously, this value will change if the current location changes; therefore, the starting location and the desired next location need to be specified.

Lastly, the stopping time is the time the rover will stop moving when a particular destination is reached. This usually means that the rover will perform scientific activities: taking measurements, drilling on the lunar surface, collecting samples, etc. The time will change depending on the waypoint the rover is at, but this value is usually known beforehand.

1.2 Problem Statement

In brief, in terms of operation, a lunar rover will receive multiple areas of interest to go to with no specific order, a starting location, a starting time, an ending location and a time limit. The goal will be to find a feasible trajectory through all the waypoints if possible or as many as possible, while respecting the time limit to reach the ending location.

Notice that there are two parts to this problem: firstly, finding an order of exploration and secondly, planning the trajectory with the order.

1.2.1 What to Explore and When?

Since the lunar environment is dynamic, time will need to be taken into account. The difficulty is to find an order of exploration that ensures that all the waypoints are available at their respective time of exploration. In other words, on top of finding what to explore, it will be necessary to specify when each waypoint will be visited.

Another consequence of taking into account time is that a path leading up to a waypoint at one point in time may not work at another time. This implies that as soon as the time stamps at which the rover visits the different waypoints change, a new path will need to be planned, since the environment has changed.

1.2.2 Static and Dynamic Obstacles

With the order of exploration confirmed, the next step is to find detailed trajectories between the waypoints. This step needs to take into account both the static and dynamic obstacles of the Moon for the trajectory to be considered feasible. For dynamic obstacles, this may be challenging, since it is constantly changing. Fortunately, it is possible to model them with the LOLA dataset combined with the ray tracing method mentioned above.

1.3 Research Objectives

The research work pursued in this thesis can be translated into the following objectives:

- Model the lunar environment to find the time windows of each waypoint
- Develop a model capable of returning the best order of exploration of waypoints with different time windows
- Develop an algorithm that can perform multi-directional path planning, given the order of exploration

The proposed solution used to perform path planning is presented in Chapter 4, alongside the results and discussion.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the current state of the art for the topic of multi-point navigation on the Moon. Chapter 3 explains the overall research approach and organization of the thesis. This will clarify the correspondence between the article and the research objectives. From there, Chapter 4 contains the article submitted to *Acta Astronautica* titled *Path Planning Algorithm for a South Pole Lunar Rover Mission* that details the proposed path planning algorithm. Its different components, results and analysis can be found there. Chapter 5 then presents a general discussion on the results of the previous chapter. Lastly, Chapter 6 concludes this thesis by summarizing the findings, presenting the limitations and exploring the future avenues of this tool.

CHAPTER 2 LITERATURE REVIEW

Multi-point path planning on the Moon is certainly no easy feat. There are multiple topics that needs to be considered, namely path planning for space rovers, multi-directional path planning, and optimizing the order of exploration.

This chapter is complementary to the literature from the article in Chapter 4, although it will go more in depth for certain topics to help readers grasp what exists at the time of writing this article. This chapter has three sections: one for each topic mentioned above.

2.1 Path Planning for Space Rovers

Path planning has been an extensive research topic over the years. As robotics is becoming more ubiquitous in our daily lives, path planning evolved as well to ensure its proper integration. A lot has certainly been done [9], but bringing autonomous robots to space, particularly rovers, is still a relatively new challenge [1]. That being said, work has been done. Many techniques and/or algorithms exist for extraterrestrial rovers.

2.1.1 Lunar Rovers

Let's begin by reviewing path planners for lunar rovers specifically. There are many types or categories of algorithms for path planning [10], and the most popular ones applicable to lunar rovers are the C-space search category and the soft computing category. The former includes graph search and sampling-based algorithms, whereas the latter include artificial intelligence and evolutionary computation techniques.

Graph Search

Graph search algorithms model the search space of the problem as a graph. This means that the environment needs to be modeled in a graph-compatible format such as a grid [10].

Many examples of application for graph search methods exist. For example, Bai and Oh [11] model the environment first with a Digital Elevation Model (DEM). With that, they managed to recreate the lunar topography and shadowing situation on the Moon. They then used the A* algorithm for path planning. Note that for the dynamic elements of the Moon, access to Earth for communication and access to Sun for power are both considered. On another note, in [12], a slightly different approach is proposed. The same grid-based separation is applied from a DEM, however, they estimated the cost of going from one node of the grid to another. This estimation is based on three elements: the distance, the rover internal elements, and the rover external elements. Internal elements corresponds to wheel and track mechanisms of the rover. The idea was to quantify wheel slippage in the lunar regolith, and the values obtained came from testing. On the other hand, external elements corresponds to insolation (areas with Sun illumination). The idea is to quantify the amount of power generated by the rover. By combining these values together, the authors used Djikstra's algorithm to find the best route. Note however that this method can also be used with other path planning algorithms, such as D* [12–14]. Finally, another approach of research is to improve the existing path planning algorithms just like in [15] where Yu and al. proposes an improvement of the A* algorithm called the Multi-cost Fast A* (MFA*). While considering an environment with steep slopes and Sun illumination, the MFA* has a different heuristic function compared to the A* algorithm. It ensures that every node is only explored once. This allows the convergence of the search algorithm to speed up drastically.

Sampling Based

Sampling-based algorithms go one step further as they do not need a graph to exist. These methods will create the nodes of that graph as they explore. They are usually meant for high-dimensional searches, since they can keep on creating new nodes, until the user specifies a stop condition [10].

A very popular sampling-based algorithm is the Rapidly-exploring Random Trees (RRT). From a starting point, the idea is to find a path with a random tree that grows until the target point is also on that tree [16]. This algorithm serves as a building block for many improved versions such as the Rapidly-exploring Random Graph (RRG). From there a tree version of the RRG is introduced called the RRT* which guarantees asymptotic optimality [17]. This is done through the rewiring function which ensures that all connections in the trees will eventually be rewired or updated to be the minimum cost of traveling. Still building on this work, a Bidirectional RRT* (B-RRT*) is introduced in [16, 18] where two search trees stemming from the starting point and the target point respectively aim for each other. In other words, the search is performed in both directions, which help with the convergence of the algorithm in complex environments. Further, an intelligent version of Intelligent B-RRT* (IB-RRT*) is proposed in [19]. The improvement here is a heuristic that allows faster convergence by improving the sample insertion and tree connection steps of a traditional B-RRT* algorithm. Note that there are many other improvement variants of the RRT algorithm, and that what has been listed here is not an exhaustive list.

Due to the complex environment of the Moon, sampling-based algorithms have made it to applications for lunar rovers. For example, in [20], a progressive RRT algorithm is introduced, where sampling is focused on a region that it calculates, and the random tree growth is also accelerated with a dynamic forward step. These improvements all help the algorithm find a trajectory faster in the context of the Moon. Similarly in [21], a Bidirectional Rapidly exploring Random Tree (B-RRT) is used with combination of information on the lunar surface. A grid map is first built containing information on the obstacles. Then, a repulsive and an attractive potential fields are created based on the obstacles and targets respectively. Lastly, the B-RRT is then modified to use those fields to help the sampling of the algorithm.

Artificial Intelligence

This category is without any doubt quite broad; therefore, it will be limited to artificial intelligence applied to lunar rovers.

In [22], a reinforcement learning method is proposed. In a grid-like world in 2D, the authors proposed eight possible actions for the rover to move. They then created a policy-gradient algorithm that returns the probability distribution for the possible actions of the rover based on its current situation. Similarly in [23], a Q-network is used to return a probabilistic distribution of the possible actions. This time though, the input is a feature map representing the lunar environment, which includes slope and illumination information. A ResNet model first extracts the information from that map, and passes the information to the Q-network. The authors noted that their work is applicable not only to lunar rovers, but to planetary rovers in general. Furthermore, a different approach can be found in [24] where an artificial intelligence technique is proposed to compensate for the lack of information on far away obstacles. Due to the limitation of the depth cameras, the authors propose a method to use the images and estimate the obstacle size and positions with a degree of uncertainty. With this information, the detected obstacles can then be included in the global map and prevent any paths that collide with them.

Evolutionary Computation

Also known as nature-inspired, these algorithms will return a trajectory through the evolution of a population [10]. There are two categories, namely the genetic methods which uses chromosome models, and swarm optimizers which imitates the behavior of animals.

On one hand, genetic methods encode trajectories as individuals with chromosomes. Essentially, every waypoint or every grid explored by the algorithm is a chromosome, and by combining them together, an individual is formed (which represents a trajectory). Incidentally, a group of individual is called a population, which boils down to a grouping of possible trajectories [25]. After initializing the population randomly, it evolves through three processes called the genetic operations: reproduction, crossover and mutation. Briefly, reproduction is about creating new chromosomes based on the best ones in the population. Then, crossover is about interchanging chromosomes within the same individual or between two individuals. Finally, mutation is about random change in the chromosomes of a single individual [26].

After running the population through these operations, the population has evolved one generation (or one cycle). The overall quality of all the individuals can then be evaluated with a fitness score, which usually characterizes the trajectory with respect to time, distance, energy consumed, or another relevant attribute. After a certain number of generations, specified by the user, have passed, the algorithm should have converged to a promising trajectory with respect to the fitness score [10, 25, 27]. Most problems solved with this method have a static environment, however, research have been done on environment that change through time [28–30]. Lastly, work has been done to improve the genetic methods for path planning applications by modifying the crossover operator, the mutation operator and the fitness score [31–33]. On the other hand, swarm optimizers use agents that imitate animal behavior. These agents move in free space (unlike the genetic methods), and their motion gradually creates a pattern that will converge to the final trajectory [10]. Many types of methods exist, each imitating a different animal, though, the most popular ones remain the Particle Swarm Optimization (PSO), and the Ant Colony Optimizer (ACO).

PSO involve agents that imitates the behavior of groups of animals. The different agents will each explore randomly the search space or the environment. A fitness score will be associated to each of the agents [34], and at the end of every iteration, the best agent is selected to be improved upon at the next iteration. In other words, the agents can communicate with each other and share previous experience [10]. Similarly, the ACO imitates the behavior of another group: ants. Ants leave behind them a trail of pheromones during their search for food that other ants can detect. Generally speaking, the higher the concentration of pheromones on a path, the more chances another ant will follow it. Using this behavior, the places and areas containing the most pheromones will by default become the best trajectory [10, 35]. More details on these two algorithms can be found below in section 2.3.3.

2.1.2 Mars Rovers

In theory, all the techniques seen for lunar rovers are applicable to Mars rovers. The only difference would be to use the models of the martian environment instead. This is possible thanks to the Mars Orbiter Laser Altimeter (MOLA) dataset [36]. However, martian rovers that were sent require more autonomy than lunar rovers in general, since they are further away from Earth. Thus, their latency is even higher. This is why some path planning tools implemented in Mars rovers are worth mentioning.

For instance, in [37], NASA introduces TEMPEST, a planner that is using the Incremental Search Engine (ISE). The latter is a heuristic search algorithm based on graph-theory specifically conceived for planning and re-planning in 3D. The novelty is that its efficiency eclipses that of A* through the expansion phase. ISE expands a state through back-simulation of the possible actions from that state starting from the goal location. TEMPEST considers the static and dynamic obstacles on Mars, namely martian topography, access to the Sun and access to Earth. When faced with dynamic obstacles, the planner can easily perform re-planning activities by identifying the portion of the path that needs to be changed. This is also something that A* alone cannot do.

A different approach is to combine path planning with environment classification with machine learning. In [38], the A* algorithm is used in combination with a classifier trained with machine learning that can identify the type of terrain the rover is going through. The authors proposed five classifications that each have a different impact on the dynamics of the rover for path planning. They are named: sand, bedrock, loose rock, embedded pointy rock and embedded round rock. The A* algorithm is then optimized with respect to different objectives such as distance, time, tilt and the type of terrain traversed.

Further, another interesting approach is to model the wheel slip dynamic and to consider that into path planning. In [39], a wheel-soil contact model is proposed where the wheel slippage is quantified through the slip ratio and slip angle. This then becomes a control problem. A path is first specified by the user or a path planning algorithm. Then, a dynamic simulation of the rover is performed on the candidate path. Finally, this candidate path is evaluated on wheel slippages, vehicle postures, and elapsed time and total distance traveled. A control law can then be constructed to reduce the errors as much as possible.

Lastly, in [40], the authors proposes a martian path planner for multi-robot based on two major steps. The first step is all about generating the environment and finding the target waypoints to explore. Specifically, a 3D terrain model is first generated based on the instrument on a previous Mars orbiter mission called the High-Resolution Imaging Science Experiment (HiRISE). By zooming in on a particular location on Mars with this model, a detailed traversability analysis is performed to find all the smooth or flat terrain, while also highlighting the steep or rough terrain to avoid. This is then translated into a probability distribution map that will be the primary mean to find all feasible target waypoints for a martian rover mission. The second step involves the path planning of the different waypoints. It is done with the RRT* and takes into account four criteria, namely the path length, the rover roll, the rover pitch, and the required turning angle the rover needs to perform to go from one waypoint to the next. Finally, the coordination between the different rovers is done with prioritized planning which is a 4D coordination methodology. Each rover is given a priority index, and each will have its path planned according to their priority. This prevents any form of dynamic collisions.

2.2 Multi-Directional Path Planning

To build on what was mentioned in the introduction, multi-directional path planning here refers to path planning through multiple waypoints or areas of interest. This is different from the work in [8, 41, 42] for example where a multi-directional A* algorithm is introduced for path planning in higher dimensional space, or improving on Djikstra's algorithm for 4-point and 8-point directional search. The reason is that fundamentally, it is still for two waypoints: a starting point and a goal point.

For multi-waypoint path planning, at the time of writing, there is not a lot of research as this is a fairly novel or to some extent overlooked field. In [43], addressing multiple goal configurations with A* for a 6 degrees-of-freedom robotic arm is proposed. The authors leverage goal switching and bi-directional search to find the best possible path for the arm. They first quantify with a distance function the cost of going to each final arm configuration from the current one. Once the less costly one is returned, bi-directional search kicks in to help convergence of the search. Even if this work is not for a lunar rover, there is the concept of multiple goal state present (which are arm configurations in this case).

Another example is in [44], where a multi-directional approach is proposed to avoid complex obstacles a rover can encounter. Whenever stuck, the rover's planner creates a new node away from the current position in the direction of the goal destination. The planner then attempts to connect to the newly created node from the currently blocked one. The authors argue that this approach helps conversion speed and reduces memory usage by drastically limiting inefficient exploration of the environment. Despite being quite an effective method, this approach is not multi-directional per our definition. Here, the goal is still to start from a starting location and reach an end location.

Lastly, in [7], a truly multi-directional method per the definition above is proposed. The problem to solve is the exploration of numerous cities in the shortest possible trajectory. Given those cities as waypoints to explore, the authors introduces the Informable Multi-

Objective and Multi-Directional RRT* (IMOMD-RRT*) algorithm, where a search tree is created at every city. Through multiple iterations, sampling is performed at all the trees in an attempt to connect the trees together. When a sampled node belongs to two or more distinct trees, a connection is formed between the respective trees. In parallel to this, an enhanced cheapest insertion algorithm and a genetic algorithm is attempting to find the order of exploration of the cities that will result in the shortest distance. This is accomplished by modeling the different cities as a relaxed Traveling Salesman Problem (r-TSP) and solving it. Note that this problem is NP-hard due to its similarity with the TSP. This work distinguishes itself from the previous ones by solving two problems at the same time: generating trajectories and returning the least costly exploration order. The combination of the two makes it multi-directional per our definition.

2.3 Optimizing the Order of Exploration

Generating the best exploration order is usually related to the TSP. This problem is NP hard (similarly to path planning [32, 45]) in the combinatorial optimization field. The traditional TSP involves finding the shortest path possible through a graph, where every node must be visited exactly once with the exception of the starting city because the salesman (or the agent) start and end at the same city [46].

2.3.1 TSP variants

There exists many variants to the TSP to better model real world problems. The following non-exhaustive list presents some along with a description.

- Symmetric / asymmetric TSP: The distance from city A and B is the same as the distance from city B and A for the symmetric TSP. Those distances are not the same for the asymmetric version. [47]
- Time-Dependent TSP: Travel times and/or distances can change over time. [47]
- Multi-Objective TSP: There are multiple criteria to optimize when solving the TSP. For example, distance and safety of the trajectory. [48]
- Multiple TSP: There are more than one salesman, and together they must visit all the cities exactly once. [49]
- TSP with Time Windows: Each city has a time window when it can be visited. [50]

- Capacitated TSP: Each salesman has a capacity constraint, which is often related to the number of items or weight transported. [51, 52]
- TSP with Profits: Each city has a profit associated, and the salesman must maximize profits within a certain time limit. [48]
- Relaxed TSP: The start and goal cities are not the same. [7]

One note about the r-TSP. It is a better representation of the reality of lunar rovers [7]. This is because the constraint of having the agent begin and end at the same location is not always valid for lunar exploration. Since, the rover will begin at the lunar lander position, it is not always necessary or even beneficiary for the rover to return to that lander after accomplishing its mission. One can simply think of the extra amount of energy required to make the trip back to the lander. However, this does not mean that the problem is any easier to solve. Indeed, by making a copy of the starting waypoint and making it the ending waypoint, we can derive the r-TSP from the TSP. Therefore, in terms of complexity, they are both equivalent.

Now, since the TSP and its variants are extensively studied problems, numerous approaches exist to solve them. Conventionally, they can be broken down in two themes or categories: exact methods and heuristic methods [53–55]. The popular ones will be presented below.

2.3.2 Exact methods

Generally speaking, the exact methods guarantee finding the optimal solution; however, they can be computationally expensive for larger problems.

Brute force: As the name implies, this method simply tries every possible permutations of the different cities to find the minimum route. This approach becomes highly impractical for larger problems with a time complexity of $O(n!)$. [53, 56]

Dynamic Programming (DP): With DP, it becomes possible to break the problem into smaller, overlapping sub-problems. This is an improvement from the brute force method, since redundant computations can be avoided by storing the results of intermediate steps. This approach of applying DP to the TSP is called the Bellman-Held-Karp algorithm. [57,58]

In essence, we need to define the state of the problem as such. Let V be a set of cities, and S be a subset of V that includes the starting city (let's call it city 1), and any other cities to be explored. Let $g(i, S)$ be the minimum cost for exploring all the cities in S exactly once, starting from city 1 and ending in city i.

The recurrence relation is thus:

$$g(i,S) = \min_{j\in S, j\neq i}[g(j, S\backslash\{i\}) + d(j,i)]$$

Here, $d(j,i)$ is the distance between cities j and i. Moreover, $S\backslash\{i\}$ is the subset of cities with city i removed. Lastly, we have the boundary condition $g(1,\{1\}) = 0$

To solve the problem, the algorithm will compute the different values of $g(i,S)$ until all the cities are considered. This means that the final solution will be

$$\min_{i\neq 1}[g(i, V\backslash\{1\}) + d(i,1)]$$

This gives the minimum cost of visiting all the cities in V once and coming back to the starting city again. [56–58]

The time complexity here is $O(n^2 2^n)$, which is exponential. [59]

Integer Linear Programming (ILP), Mixed Integer Programming (MIP) and Linear Programming (LP): The TSP formulation for these three methods are similar, with the only difference in variable and constraint definition.

ILP, as the name suggests, only deals with integer variables. Therefore, the TSP is formulated using binary decision variables which represent whether or not an edge is part of the cycle. So we can define $x_{ij} \in \{0,1\}$, where $x_{ij} = 1$ means that the salesman travels from city i to city j, and $x_{ij} = 0$ otherwise. Let us also define the cost function c_{ij} which quantifies the distance between the two cities [60]. The goal is then to minimize the total travel cost $\Sigma_{i=1}^{n}\Sigma_{j=1}^{n} c_{ij}x_{ij}$, with n being the number of cities.

In terms of constraints, we need to ensure that every city is only visited once, and that no sub-loops are formed.

For the first part, we can then say that for every city i, $\Sigma_{j=1}^{n} x_{ij} = 1, \forall i = 1,2,...,n$. Similarly for city j, $\Sigma_{i=1}^{n} x_{ij} = 1, \forall j = 1,2,...,n$. This ensures that each city is only visited once. [61]

For the sub-loop constraint, we can leverage the Miller-Tucker-Zemlin formulation. Essentially, by making the first city after the start index $u_1 = 1$ and every subsequent city in the tour a succeeding integer index, we can eliminate sub-loops. [60, 62]

$$u_1 = 1$$
$$\forall i \neq 1 : 2 \leq u_i \leq n$$
$$\forall i \neq 1, \forall j \neq 1 : u_i - u_j + 1 \leq (n-1)(1 - x_{ij})$$

So by putting everything together, we have:

$$\begin{array}{ll}
\text{Minimize} & \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}c_{ij}x_{ij} \\
\text{subject to:} & \\
\Sigma_{j=1}^{n}x_{ij} = 1, & \forall i = 1, 2, ..., n \\
\Sigma_{i=1}^{n}x_{ij} = 1, & \forall j = 1, 2, ..., n \\
x_{ij} \in \{0, 1\} & \\
u_1 = 1 & \\
2 \leq u_i \leq n, & \forall i \neq 1 \\
u_i - u_j + 1 \leq (n-1)(1 - x_{ij}), & \forall i \neq 1, \forall j \neq 1
\end{array}$$

Moving on to MIP, the approach to formulate the TSP will be very similar. However, we can now introduce continuous variables on top of the binary variables. Typically, the continuous variables are used to represent continuous aspects such as time windows or capacities.

As an example, a TSP with time windows would be formulated as such. While keeping the binary variables from above, we simply need to introduce the time variable. Let $t_i \in \mathbb{R}^+$ be the continuous variable indicating the time the salesman arrives at city i. Now, let e_i and l_i be the earliest and latest time to arrive at city i. Finally, let s_i be the service time at city i [63]. So, in this case, we need to ensure that the salesman arrives at city i within an acceptable time range. This can be expressed through the following constraint: [64]

$$e_i \leq t_i \leq l_i, \forall i = 1, 2, ..., n$$

We also need to ensure that the travel between two cities respect the service and the travel time. This leads to the following constraint:

$$t_j \leq t_i + s_i + c_{ij} - M(1 - x_{ij}), \forall i, j \text{ and } i \neq j$$

We introduce a variable M here, which is an arbitrary big real number. The point is to ensure the timing constraint is applied only when the salesman travels from city i to city j. If it doesn't, then there is no need for this constraint and the equation becomes $t_j \leq -M$ which will always be satisfied. [65]

Now we can combine the continuous variables and constraints with the previous ones:

$$
\begin{array}{ll}
\text{Minimize} & \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}c_{ij}x_{ij} \\
\text{subject to:} & \\
\Sigma_{j=1}^{n}x_{ij} = 1, & \forall i = 1,2,...,n \\
\Sigma_{i=1}^{n}x_{ij} = 1, & \forall j = 1,2,...,n \\
x_{ij} \in \{0,1\} & \\
u_1 = 1 & \\
2 \le u_i \le n, & \forall i \neq 1 \\
u_i - u_j + 1 \le (n-1)(1-x_{ij}), & \forall i \neq 1, \forall j \neq 1 \\
t_i \in \mathbb{R}^+, & \forall i \\
e_i \le t_i \le l_i, & \forall i = 1,2,...,n \\
t_j \le t_i + s_i + c_{ij} - M(1-x_{ij}), & \forall i,j \text{ and } i \neq j
\end{array}
$$

Finally, for the LP formulation, we are essentially doing a linear relaxation of the problem i.e., the binary variables can now be allowed to take fractional values between 0 and 1. This would then make the problem to be solvable in polynomial time. However, the solution will be fractional; therefore, not exactly a valid solution for the TSP. We usually use the solution returned with LP to serve as a lower bound for the ILP solutions.

Mathematically, it means that $x_{ij} \in [0,1]$. Here, when $x_{ij} = 1$, the salesman travels from city i to city j, and otherwise $x_{ij} = 0$. Additionally, $0 < x_{ij} < 1$ is also possible due to the relaxation and indicate a partial inclusion of the edge.

One other difference with the ILP and MIP formulations is that the LP formulation does not include strict integer subloop elimination constraints (due to the LP relaxation that allows fractional solutions). Note however, that it is still possible to include the constraint which would require the cutting plane method. [66, 67]

Thus, we end up with the following:

$$
\begin{array}{ll}
\text{Minimize} & \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}c_{ij}x_{ij} \\
\text{subject to:} & \\
\Sigma_{j=1}^{n}x_{ij} = 1, & \forall i = 1,2,...,n \\
\Sigma_{i=1}^{n}x_{ij} = 1, & \forall j = 1,2,...,n \\
x_{ij} \in [0,1], & \forall i,j
\end{array}
$$

To solve the ILP and MIP formulations above in an exact manner, several techniques or methods exist (the LP formulation is not discussed, since it can be solved in polynomial time). Amongst the popular ones, we can find Branch and Bound (B&B), Branch and Cut (B&C), and the Cutting Planes method.

Branch and Bound: B&B is a method that explores all possible solutions, while avoiding the ones that are subpar. First, it solves the relaxed LP version of the ILP. If this solution satisfies the integer constraint, then the optimal solution is found. On the other hand, if the solution is fractional, then branching is performed.

Branching consists in splitting the problem in two sub-problems: one that assumes the fractional variable takes the value 1 and the other one 0. The two subproblems are solved separately and this creates a branching tree. Now to avoid solving everything, the relaxed LP is solved at every subproblem. The total cost of the solution to the TSP will serve as a lower bound (minimization problem for the TSP). If this bound is worse than the best-feasible solution up until now, then this subproblem will no longer be considered and will be pruned. The algorithm then backtracks to explore other nodes of the branching tree. This process continues until all nodes are explored or pruned leaving us with the optimal solution. [56,68]

Cutting planes: The cutting planes method will gradually refine the feasible region of an LP relaxed TSP. This method adds linear constraints to exclude fractional solutions; thus, iteratively reducing the solution space toward an integer solution.

First, we begin by solving the relaxed LP of the TSP. If the solution is fractional, then the algorithm identifies regions of the solution space to constrain to force the solution to be integer. Linear inequalities are then added to the LP to exclude fractional solutions. This step is known as cutting planes, and the goal is to remove part of the solution space that does not contain integer solutions.

One common example of cutting planes is the sub-loop constraint mentioned briefly above. The way to do it is to simply constrain the number of edges between any subset of cities to be smaller than the size of the subset.

Once the solution space has been cut, the algorithm resolves the LP, and the newly generated solution will have a smaller solution space than its predecessor. This process of cutting and solving the LP is repeated until no more planes can be cut or until an integer solution is found (which is optimal). Note that the cutting planes method may not find a solution. In those cases, this method needs to be combined with other methods to explore further. [56,69,70]

Branch and Cut: Branch and cut is the combination of the above two methods together, which makes it a very powerful exact solver for the TSP.

First, a relaxed LP version of the problem is solved. If the solution is fractional, then the cutting planes method is used. If the solution is still fractional at this point, then the algorithm branches by selecting a fractional variable and creating two subproblems, just like in B&B. For each subproblem, the relaxed LP with cuts is solved again, and if the solution is still fractional, then either more cuts are added, or branching continues. In the event that solution is worse than the best known solution, then that branch is pruned, and the algorithm backtracks. This is repeated until all branches are explored or pruned. [56, 68, 70]

The time complexity of B&B, B&C and cutting planes are unfortunately not better than the brute force method, i.e a factorial time complexity $O(n!)$. This is because in the worst theoretical case, it is possible that the algorithm will still need to explore the entire problem before pruning anything. Generally speaking though, B&C is more efficient than B&B.

Concorde: One very popular solver for the TSP is the Concorde. It is known as one of the most reliable and fastest exact solver, which makes it a very good benchmark for novel techniques or methods for solving the TSP. To do so, it uses several methods covered in section 2.3 such as B&C, cutting planes and even heuristics (covered below). Note that Concorde leverages cutting plane techniques that are specific to the TSP such as the comb and the blossom inequalities, and even some heuristics when the exact approach is too expensive. Lastly, the Concorde also makes excellent use of LP relaxation techniques to speed up the convergence towards an integer solution. It incorporates several stabilization techniques like pricing or dual to do so. [56, 68, 70–72]

2.3.3 Heuristic methods

Compared to the exact methods, heuristic methods will not guarantee an optimal solution. However, they compute much faster with results that are somewhat close to optimal. Several different methods exist, and the following list is by no means exhaustive, but it will provide an overview of the popular approaches at the moment of writing this thesis.

Nearest Neighbor: A very simple algorithm that leads to quick solutions by building a tour by adding the nearest unvisited city until all the cities have been explored. So, this method will start at a random city, find the nearest unvisited city and add it to the tour, and repeating this process until the tour is complete.

This is one of the most well known greedy algorithm for solving the TSP with a time complexity of $O(n^2)$. [73]

Cheapest Insertion: Another type of greedy algorithm that works with insertion. In this case, the method starts with a tour of two cities. Then, among the remaining cities, it inserts the city that would cause the least increase in total cost. This process is repeated until the tour is complete. Note that this algorithm attempts to insert the new city in all possibilities to find the lowest cost.

The time complexity for the cheapest insertion algorithm for solving the TSP is also $O(n^2)$, and it tends to find better solutions than the nearest neighbor method. [74]

Christofides: The Christofides method is an approximation solver that has a guarantee on the quality of the solution. In fact, this algorithm guarantees a solution that will be at most 1.5 times worse than the optimal solution. However, it is only applicable if the triangle inequality is respected, i.e, for any three cities A, B and C, the distance between A and C is never greater than the sum of the distances between A and B, and B and C.

First, a Minimum Spanning Tree (MST) is built where all the cities are connected with the lowest possible cost and no cycles (not a complete tour). Then the algorithm identifies all the cities with an odd number of edges connected to them, and pairs them up together in a way that minimizes the sum of the distances between paired cities. By combining these new pairings with the MST, we get an Eulerian graph, since every vertex (or city) now has even numbered edges. From the Eulerian graph, we can find the Eulerian circuit, which is a tour that visits every edge once. Lastly, we can generate a Hamiltonian circuit from the Eulerian circuit by removing the vertices that are visited more than once. The triangle inequality guarantees that this conversion is not increasing the total cost.

Overall, the time complexity of this algorithm is $O(n^3)$, which is higher than the greedy algorithms above. However, it provides an approximation guarantee and is still computationally less expensive than the exact methods. [75]

2, 3, k-opt: These methods are also known as local search heuristic, and they gradually refine the solution by making edge changes (or swaps) to improve the initial solution.

The 2-opt version is the simplest. We start off with a tour (generated randomly or with a simple heuristic like the nearest neighbor) and identify two edges to remove. The order of the cities within the removed edges are then reversed (which creates a new segment). This new segment is then reconnected with the other cities, and if the total cost has decreased, then this new solution is kept. This process is repeated with different pairs of edges, until no more improvement can be done. The time complexity of the 2-opt is $O(n^3)$.

The 3-opt version is simply an extension of the 2-opt. It will remove three edges instead of two, which creates three segments. The algorithm then explores all the possibilities of reconnecting those three segments. If the total cost is lowered by a solution, then it is kept, and the whole process is repeated again to find better solutions until no improvement can be done. The time complexity of the 3-opt is $O(n^4)$.

The k-opt version is the generalization of the 2, 3-opt. It works exactly the same way, while removing k edges, which will create k segments. The rest works exactly the same. The higher the value of k, the higher the time complexity ($O(n^k)$), but the better the solution that will be found. [72, 76]

A common problem to 2, 3 and k-opt is that there is chance for the algorithm to get stuck in a local optimum, since they are local search methods. Note also that the k value is fixed. The Lin-Kernighan brings novelty to tackle this challenge by using a dynamic k-opt. This means that this algorithm will start with a small k, usually 2, and then increase the k value for further iterations when necessary for more complex transformations. This helps the method escape local optima.

One other thing that the Lin-Kernighan method does differently is backtracking. If a significant improvement is not achieved, the algorithm will undo the associated move and explore other ones. This also helps to avoid local optima. Due to its dynamic nature, its time complexity can be roughly approximated to be between $O(n^2 \cdot \log(n))$ and $O(n^3)$. Though, in the worst case where every solution needs to be explored, it would still approach $O(n!)$. [72, 77]

Meta Heuristics

Meta heuristics are higher-level search strategies that balance out the notions of exploration and exploitation. They also tend to be more versatile, since they are not problem specific. There are many methods that exist, and the most important ones are listed below. Although, meta heuristics cannot guarantee optimality, they are capable of producing high quality solutions in a feasible time frame which is very beneficial for larger instances of the TSP.

Simulated Annealing: This method is inspired from the annealing process in metallurgy, i.e., a metal is heated to high temperature and slowly cooled off to strengthen its structure. This translates into allowing worse solutions early in the process to avoid local optima and gradually be more selective for better solutions.

Specifically, we begin with a random solution s_0 to the TSP, compute its cost c_0 and set it as the best one. We introduce the temperature parameter T, which is the probability of accepting worse off solutions. At the start of the algorithm, this parameter will be quite high to encourage exploration. Gradually, as the number of iterations go up, T will decrease following a cooling function $T_{new} = \alpha \cdot T_{current}$ with $0 < \alpha < 1$. Then, a new solution s is generated by modifying the current one (for example using the 2-opt). If the new solution or tour is better than the previous ($\Delta c = c_{new} - c_0 < 0$), then the new one is kept. If it is not ($\Delta c = c_{new} - c_0 \geq 0$), then it will be accepted with a probability of $P = e^{\frac{-\Delta c}{T}}$. [78, 79]

If the solution is accepted then $s_0 = s$, and we repeat this whole process with an updated temperature T. Once no more improvement can be made or when T reaches a very low value, the algorithm stops and returns the best solution.

The time complexity of the simulated annealing depends on the cooling schedule and the number of iterations. Typically, though, it is between $O(n^2)$ and $O(n^3)$. [78]

Ant Colony Optimization (ACO): The ACO is inspired by the behavior of ants. They are simulated visiting different cities based on two factors: pheromone and heuristic.

We first begin by initializing ants at random cities in the solution space. The pheromone levels between all the cities are also initialized to the same constant value. The goal of all those ants will be to explore all the cities exactly once to construct a tour. They will choose the next city according to the two aforementioned factors: the pheromone level $\tau(i,j)$ between city i and j, and the heuristic $h(i,j)$ which is usually the inverse of the distance between city i and j ($h(i,j) = \frac{1}{d(i,j)}$).

So, the probability of going to city j from city i is

$$P_{ij} = \frac{(\tau(i,j))^{\alpha}(h(i,j))^{\beta}}{\Sigma_{k\in\text{allowed}}(\tau(i,k))^{\alpha}(h(i,k))^{\beta}}$$

Here, α regulates the importance of pheromone, and β regulates the importance of the shorter distances. The denominator simply sums over all the k eligible cities not yet explored. [10,35]

Once all the ants have completed their tour, the pheromone levels need to be adjusted according to an evaporation rate:

$$\tau(i,j) = (1-\rho)\cdot\tau(i,j) + \Sigma_{\text{ants}}\Delta\tau(i,j)$$

Here, $0 < \rho < 1$ is the evaporation rate, and $\Delta\tau(i,j)$ is simply the sum of the pheromone deposited by the ants that traveled between city i and j. Typically, the amount of pheromone an ant deposits is inversely proportional to the distance it traveled for the tour. [80]

At each iteration, the best tour is saved, and a global best tour is maintained until the algorithm finishes. Lastly, the time complexity for this algorithm is $O(N \cdot n^2 \cdot T)$, where N is the number of ants, n the number of cities and T is the number of iterations. [35]

Tabu search: The tabu search is similar to the simulated annealing in terms of its functioning. It relies on local search, but introduces memory structures to keep track of previous moves to avoid in a tabu list. This is to escape loops and avoid local optima.

The algorithm begins with a randomly generated solution. Then it creates an empty tabu list. This list will contain the previous moves that the method tried and will not repeat. It is dynamic and short term, meaning that the older moves will be discarded from that list as the newer ones come in. Even if a move appears promising, the algorithm will not choose it if it is on the tabu list. The only exception is related to the aspiration criterion. Essentially, if a tabu move is really good to the point of proposing a better solution than the overall best, then this criterion allows an exception to be made.

Thus, starting from the randomly generated solution, the algorithm explores neighboring solutions (by using k-opt or switches for example). It then chooses the best neighboring solution and saves that move in the tabu list. The method proceeds by generating more neighboring solutions related to the currently chosen solution, but this time, it will select the best one while respecting the tabu list unless the aspiration criterion comes into play. As the tabu list reaches its maximum length, it is updated like a rolling memory (discard old moves and replace with newer ones in a first in last out manner). Lastly, the algorithm keeps track of the best solution found, and after a certain number of iterations have been done, or when no improvement can be made, it will return the best solution. [81]

Typically, the time complexity of the tabu search can be expressed as $O(n^2 \cdot I)$, where n is the number of cities and I is the number of iterations. [81, 82]

Particle Swarm Optimization (PSO): The PSO is based on the natural behavior of birds flocking or fish schooling. When applied to the TSP, it is using particles to represent a tour (solution). Each particle will have two attributes: position and velocity. The position represents the order of the cities in the tour. Velocity, on the other hand, represents a change in position, which in this case means a reordering of some sort (for example a swap, an insertion, k-opt, etc.).

The algorithm will maintain two levels of best solutions. Firstly, at the particle level, each particle will have its own personal best. Secondly, at the global level among all the particles, there will be a best solution. Therefore, the velocity will be influenced by both the personal best of the particle in question, and the overall best.

This method will begin by initializing the particles, and each of them will be a random solution (tour) to the TSP (random positions). For the particles' velocity, it will be randomized as well as a series of moves that changes the order of exploration. Then each particle is evaluated based on its tour length. Each particle will keep track of its own personal best, and the algorithm will track the best particle score as the overall best. Now, the particles' velocity will be updated based on the two principles of exploration and exploitation. Exploration will involve introducing random moves in the velocity, and exploitation will be about introducing moves that will bring the current particle closer to the personal best and the overall best.

After the velocity update, a new series of swaps, insertion or interchange will be applied to the position of the particle which generates a new tour. Once new tours are generated, the personal best score and the overall best score are updated. [34, 83]

This process goes on for a set number of iterations or again until no more improvement can be made. The best overall solution is then returned.

Its time complexity can be expressed as $O(P \cdot n^2 \cdot T)$, where P is the number of particles, n the number of cities and T is the number of iterations. [84]

Genetic Algorithm: Genetic algorithms are inspired from the principles of natural selection, i.e. passing on the "better" genes to the next generation. They work on iteratively improving the solutions based on the genetic operators (selection, crossover and mutation). Each solution (tour) is called an individual which is made up of chromosomes (the different cities themselves), and the goal is to have a population (group of individuals) that evolves through every generation by returning shorter tours.

Firstly, a random population is generated. It contains individuals (tours) that are potential solutions to the TSP. The fitness of each of these individuals is also computed (usually, it is simply their total cost). Then, selection happens where the best individuals are chosen to go through crossover, where their genes are passed on to the next iteration. The selected individuals (also called parents) are paired and will produce offspring or children that are based on portions of their tour. This means that the child produced will inherit tour portions from both parents. Lastly, the offspring will undergo mutation under a certain probability where its order of exploration may be changed. This is to prevent premature convergence and help avoid local optima. The newly produced individuals are then evaluated and will replace some individuals in the original population based on their fitness score. [27, 45]

This process is briefly summarized below.

- Initial Population:

$$\mathbf{A} \to \mathbf{C} \to \mathbf{D} \to \mathbf{B} \to \mathbf{E}$$
$$\mathbf{B} \to \mathbf{E} \to \mathbf{C} \to \mathbf{A} \to \mathbf{D}$$
$$\mathbf{C} \to \mathbf{A} \to \mathbf{B} \to \mathbf{E} \to \mathbf{D}$$
$$\mathbf{D} \to \mathbf{B} \to \mathbf{E} \to \mathbf{A} \to \mathbf{C}$$

- Selection (Chosen Parents):

$$\mathbf{B} \to \mathbf{E} \to \mathbf{C} \to \mathbf{A} \to \mathbf{D}$$
$$\mathbf{C} \to \mathbf{A} \to \mathbf{B} \to \mathbf{E} \to \mathbf{D}$$

- Crossover (Generate Child):

$$\mathbf{A} \to \mathbf{B} \to \mathbf{E} \to \mathbf{C} \to \mathbf{D}$$

- Mutation (Add Randomness):

$$\mathbf{A} \to \mathbf{D} \to \mathbf{E} \to \mathbf{C} \to \mathbf{B}$$

After a set number of iterations or when no improvement can be made, the algorithm returns the best individual from the current population. [32].

Genetic algorithms also have numerous variants that can improve its convergence speed, memory usage and simply overall effectiveness. They include modifications to the workflow of the traditional genetic algorithm such as parallelization to speed up computation time, introduction of chaos and randomness to avoid premature convergence, and combination of external search techniques to better adapt to specific environments. Recall also that work has been done to improve its effectiveness through the genetic operators and fitness score. Lastly, the genetic algorithm can also solve the more generalized Vehicle Routing Problem (VRP) [31–33, 51, 85, 86].

The time complexity for the genetic algorithm will depend on various factors such as the number of cities n, the number of generations G and the population size P. Typically, its time complexity can be expressed as $O(G \cdot P \cdot n^2)$. [31, 85]

CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION

This chapter introduces the overall approach of this thesis. The link between the objectives stated in section 1.3 will be made apparent here, along with the connection of the article in Chapter 4. A detailed organization of the thesis will also be presented for clarity.

There will be three sections, namely problem modeling, problem solving and validation.

3.1 Problem Modeling

To solve the problem presented in section 1.2, a specific type of data is required. Recall the LOLA dataset presented in section 1.1.1. It is used reliably to model the topography of the lunar surface [4]. Further, to model the dynamic environment of the Moon, we use ray tracing techniques. The idea is to verify if unobstructed lines-of-sight exist between two celestial bodies [5]. In the particular case of this problem, we are mostly interested in the lines-of-sight between the Moon and the Sun, and between the Moon and the Earth. This is because, we seek to model illumination zones on the Moon with respect to the Sun and to model communication zones on the Moon with respect to the Earth.

The combination of the two modeled data can be referred as the accessibility data, since it represents where (and when) the rover can go. It will be crucial for the path planner algorithm to take it into consideration when generating feasible trajectories.

The accessibility data is already available at CSA. Operators have manually generated it. All in all, information on slope, access to the Sun and access to the Moon are all available; therefore, the lunar environment can be properly modeled.

3.2 Problem Solving

As presented in Chapter 2, there are many techniques to tackle a path planning problem. Each with its unique strengths and weaknesses. In the context of this work, a variation of the Genetic Algorithm and a sampling-based algorithm are chosen to solve the problem (as presented in section 4.5).

3.2.1 Not a simple TSP

To better understand the decision of splitting the problem in two portions, and why a genetic algorithm is chosen, we need to understand its fundamentals. Contrary to the conventional

TSP the CSA is facing a unique challenge as described in section 1.2 that constitutes elements from several variants of the TSP. **In fact, one can argue that the problem is a relaxed, asymmetric, time-dependent, multi-objective TSP with time-windows and profits.** For clarity, the following differences can be made from the conventional TSP:

- The lunar surface is a dynamic environment which means that the problem needs to take into account the time dimension.
- For all the sites the rover will attempt to explore (with the exception of the starting and ending point), there will be time windows, i.e., times at which the sites will be available to explore.
- The paths planned to reach the different sites will vary based on when the rover performs the traverse. This means that going from site A to B may take path σ at time t_0. But if the rover leaves site A at time t_1 instead, it will take a different path σ'.
- There is a need to replan quickly at the operator's request either before the rover's departure or during the traverse. This may be because the operator added, removed or swapped one or multiple sites.
- Due to operational constraints, there is a hard time limit for the traverses, which means that the algorithm needs to explore as much as possible based on the scientific relevance of the available sites while reaching the ending location in that time limit.
- The starting and ending sites are different.

From all the different methods discussed in Chapter 2, there is none that can fully solve the current problem. Moreover, using exact methods would be unfeasible due to the factorial growth. For example, suppose the four sites to explore are called: **A**, **B**, **C**, **D**. The set of solutions would be this:

$\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C} \rightarrow \mathbf{D}$

$\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{D} \rightarrow \mathbf{C}$

$\mathbf{A} \rightarrow \mathbf{C} \rightarrow \mathbf{B} \rightarrow \mathbf{D}$

$\mathbf{A} \rightarrow \mathbf{C} \rightarrow \mathbf{D} \rightarrow \mathbf{B}$

$\mathbf{A} \rightarrow \mathbf{D} \rightarrow \mathbf{B} \rightarrow \mathbf{C}$

$\mathbf{A} \rightarrow \mathbf{D} \rightarrow \mathbf{C} \rightarrow \mathbf{B}$

$\mathbf{C} \rightarrow \mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{D}$

$\mathbf{C} \rightarrow \mathbf{A} \rightarrow \mathbf{D} \rightarrow \mathbf{B}$

$\mathbf{C} \rightarrow \mathbf{B} \rightarrow \mathbf{A} \rightarrow \mathbf{D}$

$\mathbf{C} \rightarrow \mathbf{B} \rightarrow \mathbf{D} \rightarrow \mathbf{A}$

$\mathbf{C} \rightarrow \mathbf{D} \rightarrow \mathbf{A} \rightarrow \mathbf{B}$

$\mathbf{C} \rightarrow \mathbf{D} \rightarrow \mathbf{B} \rightarrow \mathbf{A}$

$\mathbf{B} \rightarrow \mathbf{A} \rightarrow \mathbf{C} \rightarrow \mathbf{D}$

$\mathbf{B} \rightarrow \mathbf{A} \rightarrow \mathbf{D} \rightarrow \mathbf{C}$

$\mathbf{B} \rightarrow \mathbf{C} \rightarrow \mathbf{A} \rightarrow \mathbf{D}$

$\mathbf{B} \rightarrow \mathbf{C} \rightarrow \mathbf{D} \rightarrow \mathbf{A}$

$\mathbf{B} \rightarrow \mathbf{D} \rightarrow \mathbf{A} \rightarrow \mathbf{C}$

$\mathbf{B} \rightarrow \mathbf{D} \rightarrow \mathbf{C} \rightarrow \mathbf{A}$

$\mathbf{D} \rightarrow \mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C}$

$\mathbf{D} \rightarrow \mathbf{A} \rightarrow \mathbf{C} \rightarrow \mathbf{B}$

$\mathbf{D} \rightarrow \mathbf{B} \rightarrow \mathbf{A} \rightarrow \mathbf{C}$

$\mathbf{D} \rightarrow \mathbf{B} \rightarrow \mathbf{C} \rightarrow \mathbf{A}$

$\mathbf{D} \rightarrow \mathbf{C} \rightarrow \mathbf{A} \rightarrow \mathbf{B}$

$\mathbf{D} \rightarrow \mathbf{C} \rightarrow \mathbf{B} \rightarrow \mathbf{A}$

From this, we can see that there are 4! possible permutations with each 3 paths to plan. This then brings the total number of paths to explore to 72 paths. While it's true that there are some redundant paths that may be reused, the growth in number of paths to plan as the number of sites increases is factorial according to $n! \cdot (n-1)$.

The time required for planning a path between two sites is highly dependent on the complexity of the environment and the distance between the two sites. However, for argument's sake, we can assume a lower bound approximation of 10 seconds. This means that for $n = 7$ sites, it can take up to 3.5 days, and for $n = 8$ sites it would be more than 30 days in the worst case scenario. Based on those numbers, it would simply be unfeasible to find the optimal solution in a reasonable time frame. Consequently, this problem requires an approximation method approach for the nominal operations.

3.2.2 Why Split the Problem?

Additionally, the problem currently requires the order of exploration to be found. However, to find a good order of exploration, the exact distances between each waypoint need to be known. Unfortunately, this creates a loop, since to find the exact distance, we need to know the time at which they are being explored (which depends on the order).

Because of this, we need to start somewhere. This is why the problem has been separated into two components: figure out the order of exploration quickly, and then perform a detailed planning. We begin by considering only the static obstacles for estimating distances between the waypoints. Since we want to generate a solution for the order of exploration quickly, a heuristic approach is preferred. One note is that this problem needs multi-objective optimization: distance should be minimized, whereas scientific relevance should be maximized.

Genetic algorithms are well known to handle well multi-objective optimization. Through its population based search, solution diversity is maintained. On top of that, they are quite flexible and adaptable to changing priorities. [27,87–90] Therefore, they will be used to tackle the first part of the this problem, that is, returning an order of exploration. With this, time stamps will be assigned to the different sites, which constitutes the starting point for the second part of the problem.

The second part involves the detailed planning of the trajectories. This is where we bring the dynamic obstacles into the picture. With the given order of exploration and proposed time stamps for the different sites, we now validate if a trajectory can be planned. We use a sampling-based algorithm. This is because they tend to work well in complex environment, particularly the B-RRT*. In the presence of challenging obstacles especially with concave cavities, it is difficult to find solutions with graph-based methods. Even with unidirectional sampling-based techniques, it can be challenging [16, 18]. This is why the B-RRT* is chosen as building block for creating feasible trajectories.

Finally, there is an operational reason for splitting the problem like so. The operators would prefer having a quick "draft" version of the potential trajectory so that they can approve or modify the proposed trajectory by adding or removing sites. Thus, at first the genetic algorithm returns this draft version providing the operators the opportunity to review. After their approval, we proceed with the detailed planning.

3.3 Solution Validation

At the moment of writing this thesis, there is no known tool that does what the proposed path planner does. Hence, the best way for comparison is to rely on manually planned trajectory data from CSA's operators. Note that for larger instances, it will not be possible to find an optimal solution for the reasons stated above. However, for smaller instances, the brute force method with the proposed tool can be used to find the optimal solution (though it will take a while) as shown in section 4.6.

For information, an operator will gather the lunar environment data for a particular place and time, and manually connect waypoints of interest to the best of their capabilities. This is a lengthy process that can take hours and days.

The way to compare the path planner and the operator's work will be to use as inputs for the path planner the same waypoints as the operator did. Then, by comparing the generated trajectory of the path planner with the operator's, we can assess the differences, as described in section 4.6.

Special attention is required for the cases when the path planner cannot find a feasible trajectory. This usually means two possibilities, either the path planner does not work for an unknown reason, or the operator attempted a trajectory that is deemed impossible. These are the cases that are interesting, since the limitations of the path planner are being challenged. Clear and transparent communication are required to improve this path planner.

3.4 Document Structure

This thesis follows the thesis by articles structure in which the submitted article is directly integrated as a distinct chapter.

- Chapter 1 introduces the reader to the context and some relevant background information for the settings of this project.
- Chapter 2 presents a literature review of the relevant topics related to this work and serves as a complement for the literature review of the article in Chapter 4.
- Chapter 3, the current one, proposes the overall research approach.
- Chapter 4 presents a tool that can perform path planning for multiple waypoints for a lunar rover on the South pole of the Moon. This work has been submitted to the Acta Astronautica in August 2024.
- Chapter 5 contains a general discussion of the results presented in Chapter 4 and throughout this thesis.
- Chapter 6 summarizes the work presented in-so-far, the limitations of this work, and potential future work or research directions.

CHAPTER 4 ARTICLE 1: PATH PLANNING ALGORITHM FOR A SOUTH POLE LUNAR ROVER MISSION

Feng Yang Chen, Natasha Jackson, Pierre Allard, Giovanni Beltrame (2024). Path Planning Algorithm for a South Pole Lunar Rover Mission. Submitted to *Acta Astronautica* on 18 August 2024

The other co-authors mainly had a mentoring or supervising role throughout this project. The bulk of the work was done by me.

4.1 Abstract

On the Moon, a rover needs to navigate around physical obstacles related to the topography such as boulders or steep slopes, to reach targets of scientific interest. In addition, it needs to avoid shadowed areas (that vary during the day), which prevent the rover from having access to the Sun for energy or to the Earth for communication. The combination of changing illumination and communication, as well as the rugged terrain make the path planning quite complex. Moreover, the distance between the Moon and the Earth causes a latency for commands to reach the rover. This makes it difficult to plan the rover's actions, especially with many areas to explore. To tackle this problem we propose a novel mission planner tool with two components: we first use a two-step Genetic Algorithm to compute a tentative order of exploration from a list of points of interest, allowing us to associate a time stamp for each explored waypoint. We then compute feasible trajectories between the ordered waypoints while ensuring that the rover is avoiding all static obstacles, is staying in contact with Earth and, is being powered by solar illumination. Simulation results show that this tool work well for different lunar sites and substantially reduces the workload for manual mission planning.

Keywords: Path Planning, Lunar Rover, Dynamic Environment, Genetic Algorithm, Exploration Order

4.2 Introduction

Humanity's return to the Moon has been announced with the Artemis missions and Gateway station. Canada is contributing to this return notably through the Lunar Exploration Accelerator Program (LEAP), where one deliverable will be a Canadian rover. At the time of writing, this rover is preliminarily scheduled to launch in 2027.

Just like any other rovers with minimal autonomous capabilities, the LEAP rover requires a navigation system, i.e., a tool or a program that allows the rover to know how to go from a starting location to a goal location. On Earth, this is quite trivial: one can simply activate the GPS function on a phone to know the directions to go to a goal from the current location.

However, on the Moon, the problem becomes drastically different. The lunar environment is quite harsh and for any rover to survive, it would need at least: access to sunlight to power itself and communication to Earth to receive instructions or to transmit data. On top of this, the surface of the Moon is quite rugged; thus, the rover must avoid obstacles caused by the challenging topography. Lastly, due to the distance between the Moon and the Earth, there will be a latency for commands on Earth to reach the rover on the Moon. This makes it difficult for the operator to plan the rover's daily activities long term, particularly when there are many areas of scientific interest to explore.

From the aforementioned challenges, it is clear that a simple path planner between a starting location and a goal location is not enough. Instead, a tool that can on one hand, return the optimal order of exploration given any number of waypoints to explore, and on the other hand, compute a feasible trajectory given the static obstacles and dynamic environment of the Moon is needed. This is exactly what we propose.

Our mission planner tool is separated in two main parts: the estimator and the detailed path planner. To achieve its objective, this tool first uses the estimator, which relies on a variant of the Genetic Algorithm, to return an order of exploration for the areas of interest and an estimate of a time of exploration for each waypoint. Then, the results of this first step are passed to the detailed path planner which finds feasible trajectories given the order of exploration, and the lunar environment.

The remainder of this article is organized as follows. Section 4.3 presents the current state of the art for the topic of multi point navigation on the Moon. Section 4.4 shows the background on the notation and the specific problem that we are trying to solve. From there, Section 4.5 describes the methodology of this tool. We first present the 2-step Genetic Algorithm, the first main component of the tool. Then we continue with the second main component, i.e., the detailed path planner called MD-RRT*. Lastly, Section 4.6 illustrates results and Section 4.7 provides discussions and a conclusion for this proposed tool.

4.3 Related Work

4.3.1 Path planning in space

Work has been done on path planners for extraterrestrial rovers. For lunar rovers, Bai and Oh [11] proposes the A* algorithm after modelling the environment with static and dynamic obstacles from a digital elevation model (DEM). For the dynamic environment, access to Earth for communication and access to Sun for power are both considered. On the other hand, Yu and al. [15] proposes a variant to the A* algorithm, the Multi-cost Fast A* (MFA*). In an environment that considers steep slopes and Sun access, the MFA* uses a different heuristic function which ensures that every node is only explored once, thereby speeding up the convergence of the search algorithm. Another approach consists of leveraging reinforcement learning. In the work of Yu and al. [22], a policy-gradient algorithm is proposed. It is optimized with the proximal policy method and returns a probability distribution for the possible rover actions. Similarly in Hu and al. [23], a Q-network is used to return a probabilistic distribution of the different actions possible for the rover. The input to this model is a feature map containing slope and illumination, and it is passed into a ResNet model to extract the features of the environment. Note that this work applies to planetary rovers in general and does not limit itself to lunar rovers.

For the mars rovers, NASA proposed the TEMPEST planner [37]. This planner leverages an Incremental Search Engine (ISE) which is a graph-theory based, heuristic search algorithm optimized for planning and re-planning in high-dimensional spaces. Its efficiency outclasses A* by expanding a state by backward-simulating the possible actions from that state. The planner considers the terrain, access to Sun and access to Earth for the obstacles. Another approach for the Martian environment is to combine path planning with classifying the type of environment the rover is going through. In [38], Ono and al. use the A* algorithm in combination with a machine learning model that classifies the terrain type in front of the rover. The A* algorithm is also optimized with respect to different objectives (distance, time, tilt, ...) and these optimization functions can be mixed together by assigning different weights to the individual functions. In the more recent work [40], a 3D terrain model is first generated based on the instrument on a mars orbiter called the High-Resolution Imaging Science Experiment (HiRISE). With this model, traversability analysis can be performed to favor smooth and flat terrain, which is lastly translated into a probability distribution map to help choose target waypoints for a mars rover to explore. The path planning is performed with the RRT* (discussed more in section 4.3.2), which uses a cost function that takes into account path length, rover roll, rover pitch, and required turning angle.

One can also note that the path planners proposed for extra-terrestrial rovers are for a starting location and a goal location. Therefore, given two points in space, they can provide a feasible trajectory for the rover to go from one to the other.

4.3.2 Multi-directional path planning

Path planning on Earth has been researched extensively in the past years, and many approaches exist nowadays. Sanchez and al. [10] propose a classification that encompasses well the current state of the art. The categories are Reactive computing, Soft computing, C-space search and Optimal control.

For lunar rovers, most existing planners and proposed work described in the section beforehand belong in the C-space search category, particularly the sampling based approach [10,91]. A very popular sampling-based algorithm is the Rapidly-exploring Random Trees (RRT), which also serves as a building block for many improved versions such as the RRT* who guarantees asymptotic optimality [17]. Then, a bidirectional RRT* (B-RRT*) is explored where there are two search trees aiming from the start to the goal, and from the goal to the start respectively [18]. They are essentially converging towards each other, and this helps with convergence in complex environment. An intelligent version of B-RRT* (IB-RRT*) further improves it with an unique sample insertion and tree connection heuristic, which allows for even faster convergence [19].

At the time of writing this article, not much work has been done in multi-directional path planning, i.e., planning a path through multiple waypoints. In Qian et al. [44], a multi-directional approach is used to avoid complex obstacles. Whenever the planner gets stuck, a new node is created further away in the direction of the goal and the planner tries to connect to that new node from the currently stuck one. This approach accelerates conversion and saves up memory by limiting drastically the number of nodes explored. Despite being quite effective, it is not quite multi-directional per se. Alternately, in Huang et al. [7], a truly multi-directional approach is presented. Given many cities to explore, the author proposes the IMOMD-RRT* algorithm, where a search tree is created at every city (starting point). A connection is formed when a sampled node belongs to two or more distinct trees. At the same time, an enhanced cheapest insertion algorithm and a genetic algorithm (ECI-Gen) is running to determine the best order of exploration for the different cities. This is done by solving the relaxed Traveling Salesman Problem (r-TSP). This work distinguishes itself by solving two problems bundled together, i.e., finding feasible trajectories and finding the best order of exploration.

4.3.3 Ordering waypoints

When it comes to finding the best order of exploration, one can easily think of the Travelling Salesman Problem (TSP). It is an NP hard problem in combinatorial optimization that involves finding the shortest path for a graph where all nodes are explored exactly once with the starting point and ending point being the same [46]. There exist a variant for this problem that has arguably better practical use: the r-TSP introduced above. The r-TSP is just like the TSP, but it does not impose the same starting and ending location [7]. For applications involving interplanetary rovers, it is likely that the mission operator specifies a different end goal than the starting point. Lastly, the r-TSP variant can still be considered a NP-hard problem. By making a copy of the starting location and subsequently making the copy the ending location, we logically end up with a r-TSP problem formulation that is equivalent to the TSP. Thus, the r-TSP still belongs to the NP hard problem class.

To solve the TSP, various methods exists as this problem has been tackled extensively [46, 92, 93]. The following list presents types of approaches to solve the TSP, but is by no means exhaustive: Heuristic approaches, Memetic algorithms, Ant colony optimizations, Simulated annealing, Genetic algorithms, Neural networks, etc.

The Genetic Algorithms (GA) category distinguishes itself from the others, since it can solve larger problems with very close to optimal solutions than its counterparts [27]. The GA also has many variants that can improve the effectiveness of the algorithm in terms of convergence speed and memory usage [85]. These variants modify the process or workflow of the GA, such as having parallel GA to speed up computation time, chaotic GA to prevent premature convergence and hybrid GA to combine the GA with other search techniques. Additionally, Dou and al. [86] worked on different crossover operators on the effectiveness of solving the TSP. Lastly, to further enhance the idea of the versatility of the GA, it can also adapt to a more generalized version of the TSP: the Vehicle Routing Problem (VRP) [51]. In this case, there may be various vehicles (instead of one traveler) to explore various cities. In other words, the TSP is a special case of the VRP where there is only one vehicle and no notion of vehicle capacity.

4.4 Background

This section will present the relevant terminology and notation.

4.4.1 Basic Definition

The lunar rover will be given a list of n waypoints $W = \{w_1, w_2, ..., w_n\}$, where $w_i \in \mathbb{R}^2$ is a waypoint containing cartesian coordinates. Each of these waypoints will have different availability depending on the time. Thus, $\delta(w_i, t_i)$ will provide a mapping between the waypoints and their availability, where t_i is the time waypoint w_i is visited. When the waypoint w_i can be explored at a specific time t_i, $\delta = 1$, otherwise $\delta = 0$.

$$\delta(w_i, t_i) = \begin{cases} 1, & \text{if available.} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

Note that there are two special waypoints for the start and goal: w_{start} and w_{goal}. These are assumed to always have $\delta = 1$.

Every waypoint also has a science score associated which represents the scientific interest of exploring it. This is simply an integer from 1 to 5, with 5 being a top priority and 1 being the lowest priority. The value is given by scientists and is known beforehand.

For practicality, we consider augmented waypoints as waypoints associated with a time stamp. Thus, they contain the Cartesian coordinates and a time stamp, meaning that they belong to $\mathbb{R}^3$. The notation for augmented waypoint k is $s_k = (x_k, y_k, t_k)$. We can then simplify the availability mapping by rewriting it as $\delta(w_i, t_i) = \delta(s_k)$, where the augmented waypoint s_k corresponds to the waypoint w_i and its time stamp t_i. Lastly, the augmented waypoint has the same science score as its waypoint counterpart.

Sequences

By grouping augmented waypoints together, we have a sequence. A sequence j is denoted by $S_j = \{s_{start}, s_1, s_2, ..., s_m, s_{goal}\}$. Here, the subscript m stands for the number of augmented waypoints in the sequence j excluding the start and the goal; therefore, $1 \leq m \leq n$ for the sequence S_j (assuming that sequences of just the start and the goal are impossible).

A sequence j is considered feasible if all its waypoints are available, i.e, $\delta(s_k) = 1, \forall k = 1, 2, ..., m$ (recall that by assumption the first and last waypoint of every sequence are always available, since they are the start and the goal).

There are two types of rewards for the sequences to quantify their quality: $r_{science}$ and r_{dist}. The former is the science score of the sequence, and the latter is the distance traveled by the sequence. For any sequence S_j, the science score reward is calculated by adding the science score of every waypoint visited besides the start and the goal.

$$r_{science} = r(S_j) = \sum_{k=1}^{m} r(s_k) \tag{4.2}$$

r_{dist} for the sequence S_j, on the other hand, is simply the distance traveled by going through the sequence (including the start and the goal).

$$r_{dist} = r(S_j) = \sum_{k=1}^{m-1} dist(s_k, s_{jk+1}) + dist(s_{start}, s_1) + dist(s_m, s_{goal}) \tag{4.3}$$

The *dist* function simply returns an estimated distance between the augmented waypoints. This is computed by a 2D version of the B-RRT* path planner presented below.

Sequences will be useful to determine rapidly the best order of exploration without worrying about the detailed trajectories between the waypoints. The 2-step Genetic Algorithm introduced below will be used to do so.

Continuous Path Planning

Whereas sequences contain augmented waypoints to explore, path planning takes care of planning the trajectory between these waypoints. Therefore, we define a set $O \subseteq \mathbb{R}^l$, where l is the dimension of the given space ($l \geq 2$). Moreover, this space is separated into obstacle and obstacle-free space, i.e., $O_{obstacles} \subset O$ and $O_{free} = O \backslash O_{obstacles}$. Note that the augmented starting waypoint and the augmented goal waypoint are assumed automatically to be part of the obstacle free space, that is $s_{start} \in O_{free}$ and $s_{goal} \in O_{free}$. Recall that O_{free} means two things on the lunar surface: no static obstacles and access to the Sun and the Earth ($\delta = 1$)

When performing a search, m is the number of augmented waypoints (excluding the start and goal) that the search is considering. For the path planning algorithm, the number of search trees required is $2(m+1)$, and the number of trajectories planned is $a = m+1$. This is because every augmented waypoint will have two search trees with the exception of the start and goal which have one. Figure 4.1 shows an example with $m = 4$, where the first waypoint and the sixth waypoint are the start and goal respectively. Thus, there will be 10 search trees in total, with 5 trajectories planned.

As an example, let $T_i = (V_i, E_i)$ where $V_i \subset O_{free}$ and $T_{i+1} = (V_{i+1}, E_{i+1})$ where $V_{i+1} \subset O_{free}$ be two growing random trees towards each other. Here V denotes the nodes and E the edges connecting the nodes. The trees T_i and T_{i+1} begin their search respectively at the state (or augmented waypoint) $s^i_{init} \in O_{free}$ and the state (or augmented waypoint) $s^{i+1}_{init} \in O_{free}$.

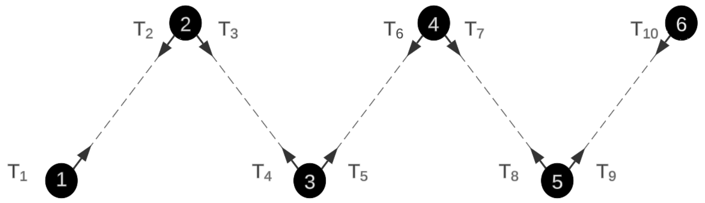

Figure 4.1 Example of B-RRT* with $m = 4$

Now, let the trajectory connecting two distinct augmented waypoints $s_1 \in O_{free}$ and $s_2 \in O_{free}$ be $\sigma(u)$ where $0 \leq u \leq 1$ and $\sigma(0) = s_1, \sigma(1) = s_2$. σ is thus a function that takes the parameter u to return points on a trajectory, thereby describing the path. As opposed to sequences which consist of a set of augmented waypoints, the trajectories are continuous and are what link those waypoints together via a set of points. However, note that the trajectories do include the augmented waypoints, since the goal is to pass through them. Finally, for a trajectory to be feasible, all of its points must belong to O_{free}, which means that $\sigma(u) \in O_{free}$ for $0 \leq u \leq 1$.

Now consider the connecting state $s \in O_{free}$ between the two trees T_i and T_{i+1}. The path trajectory starting from $s^i_{init} \in O_{free}$ to s is:

$$\sigma_i(u_i) \subset O_{free} \mid \sigma_i(0) = s^i_{init}, \sigma_i(1) = s \tag{4.4}$$

for $0 \leq u_i \leq 1$.

Similarly, the path trajectory starting from $s^{i+1}_{init} \in O_{free}$ to the same augmented waypoint s can be defined as:

$$\sigma_{i+1}(u_{i+1}) \subset O_{free} \mid \sigma_{i+1}(0) = s^{i+1}_{init}, \sigma_{i+1}(1) = s \tag{4.5}$$

for $0 \leq u_{i+1} \leq 1$.

These path trajectories are considered feasible respectively when $\sigma_i(u_i) \in O_{free}$ for $0 \leq u_i \leq 1$ and when $\sigma_{i+1}(u_{i+1}) \in O_{free}$ for $0 \leq u_{i+1} \leq 1$.

The end-to-end trajectory between s^i_{init} and s^{i+1}_{init} is then denoted by $\sigma_{f_a} = \sigma_i + \sigma_{i+1}$, where a marks the a^{th} end-to-end trajectory. In the context of this work, there are $m + 1$ end-to-end trajectories for m augmented waypoints; therefore, $1 \leq a \leq m + 1$. If the two trajectories σ_i and σ_{i+1} are feasible, then σ_{f_a} is feasible too.

The complete trajectory which combines all the $m + 1$ end-to-end trajectories σ_{f_a} would then be expressed as:

$$\sigma_f = \sum_{a=1}^{m+1} \sigma_{f_a} \tag{4.6}$$

Lastly, for the complete trajectory to be feasible, $\sigma_{f_a} \in O_{free}$, for $1 \leq a \leq m+1$.

4.4.2 Path planning with time

Before diving deeper, it is important to understand that the South Pole of the Moon is a dynamic environment. Thus, the path planner models the lunar environment in a specific way. When estimating distances for sequences, there is no notion of time; therefore, the path planning algorithm (B-RRT*) works just like in [18, 19] in 2D. There are two search trees that begin at two waypoints which converge towards each other, and a connection is made when an end-to-end trajectory is found. In the end, the returned complete trajectory is the sum of all the end-to-end trajectories found during the prescribed number of iterations.

In the case of trajectory planning, time needs to be taken into consideration. Thus, we augment the 2D environment by adding the time dimension as described in section 4.4.1. Dynamic environment factors such as access to Earth and access to the Sun can therefore be modeled. With this change, the path planning algorithm needs to consider the notion of time as well. This means that all the points in σ_f respects chronological order. This is accomplished by ensuring that the sampling step respects chronology, and that the connection of the different search trees does so as well. This will be detailed further below.

4.4.3 Problem Statement

The goal is to find a complete and feasible trajectory σ_f (not necessarily optimal, but that $\sigma_f \in O_{free}$) that will first maximize the science score $r_{science}$ but also minimize the distance r_{dist}, given a list W of n possible waypoints to explore in a time limit T. Note that this problem is NP-hard, as shown above.

This problem can be separated in two steps: the sequence part and the path planning part. Figure 4.2 shows the data flow of the tool.

For the sequence portion, we first need the distances between each waypoint. To do so, we build an adjacency matrix first by estimating the distances between every waypoint with a 2D version of B-RRT*. Note that only static obstacles are considered, since no notion of time is considered.

We then run a variation of a Genetic Algorithm, called the 2-step Genetic Algorithm. It first chooses waypoints from the list W that will form a feasible sequence S^* which maximizes

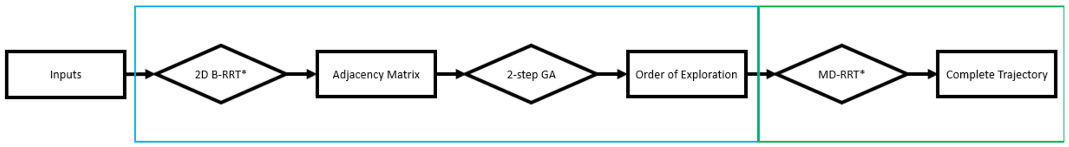

Figure 4.2 Data flow of the proposed tool. In blue, the sequence portion and in green the path planner portion

$r_{science}$ while respecting the time limit T. Notice that the augmented waypoints are created as a waypoint is chosen. This is because the time stamp can only be assigned as the sequence is built.

$$r_{science}(S^*) = r^* = r(S^*) = \max_S r_{science}(S) \tag{4.7}$$

Afterwards, with the same waypoints in sequence S^*, we rearrange the order of the sequence to get another feasible sequence S^{**} such that the total distance traveled is minimized. Here again, the augmented waypoints are created as the ordering of waypoints is confirmed.

$$r_{dist}(S^{**}) = r^{**} = r(S^{**}) = \min_S r_{dist}(S) \tag{4.8}$$

At the end of this first step, we have a feasible sequence of augmented waypoints which respects the time limit, maximizes the science score and minimizes the distance. This sequence will also have approximate time stamps for each augmented waypoint, which will be useful for the path planning.

For the path planning portion, the goal is to find a path σ_f in the obstacle free space, i.e., $\sigma_f \in O_{free}$ that passes through all the augmented waypoints of the sequence S^{**}. The algorithm used is called the Multi-Directional RRT* (MD-RRT*). If no path exists, failure is communicated.

4.5 Methodology

4.5.1 2-step Genetic Algorithm

The proposed 2-step Genetic Algorithm (GA) is the first steppingstone of this work and has the sole purpose of returning rapidly the sequence s^{**} introduced above.

Figure 4.3 shows the main phases of the 2-step GA with the genetic operators shown in blue. The upper box is the first step, and the bottom one is the second step.

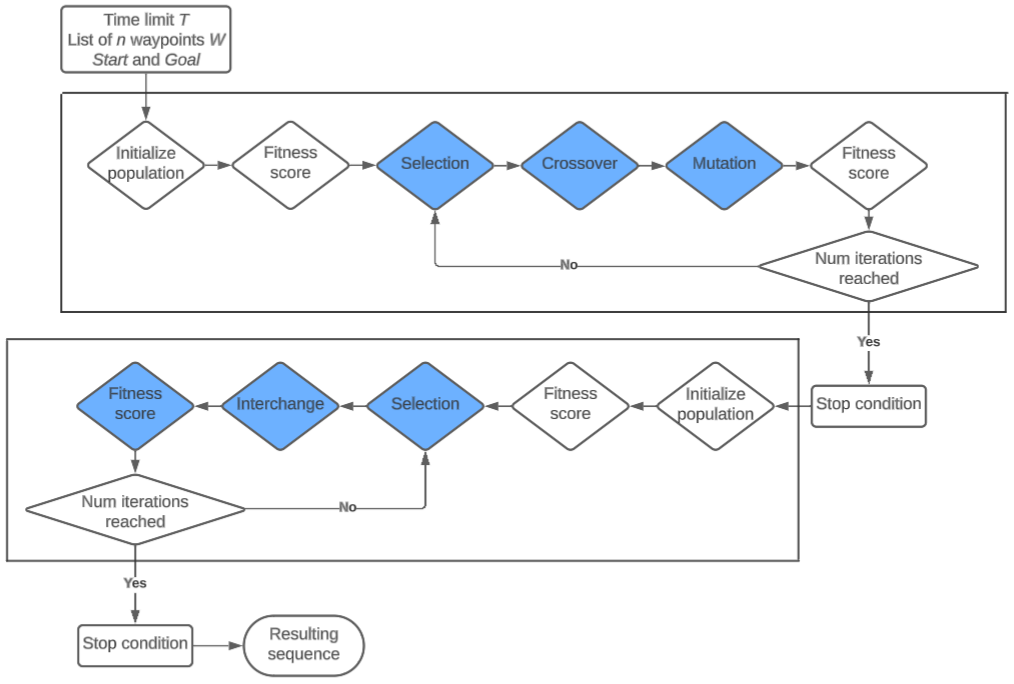

Figure 4.3 2-step Genetic Algorithm

Step 1

The first step of the 2-step GA involves maximizing the science score $r_{science}$ from equation 4.7, while respecting the time limit T. This will ensure that we have selected the most pertinent waypoints from the initial list W.

Every waypoint w_i from the list W is encoded as a chromosome. This means that initially a chromosome is composed of a set of coordinates x_i, y_i. By putting these chromosomes together, we start building a sequence, and each chromosome will also be assigned a time stamp t_i; thus, becoming augmented waypoints. The time stamp represents the time at which the rover will tentatively visit the waypoint.

Each sequence starts at the same waypoint coordinates x_s, y_s and ends at the same waypoint coordinates x_e, y_e. Note that the time stamp of the goal location t_e will differ from sequence to sequence, but they all respect T such that $t_e \leq T$. Additionally, besides having different orders, one peculiarity is that every sequence can have a different number of augmented waypoints in total than other sequences. Figure 4.4 illustrates this process.

Initialize population: This function starts the algorithm by generating a specified number of trajectories. When randomly adding waypoints to a sequence, an estimated time stamp is associated to each one; therefore, transforming them into augmented waypoints (see figure

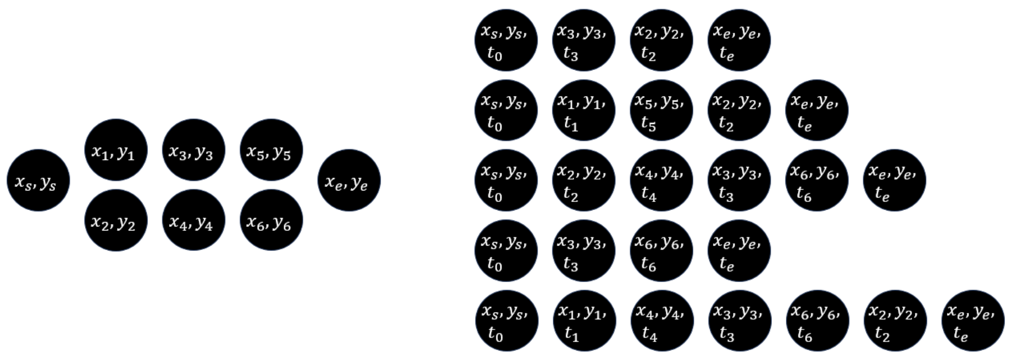
x_s, y_s
x_1, y_1
x_2, y_2
x_3, y_3
x_4, y_4
x_5, y_5
x_6, y_6
x_e, y_e
x_s, y_s, t_0 x_3, y_3, t_3 x_2, y_2, t_2 x_e, y_e, t_e
x_s, y_s, t_0 x_1, y_1, t_1 x_5, y_5, t_5 x_2, y_2, t_2 x_e, y_e, t_e
x_s, y_s, t_0 x_2, y_2, t_2 x_4, y_4, t_4 x_3, y_3, t_3 x_6, y_6, t_6 x_e, y_e, t_e
x_s, y_s, t_0 x_3, y_3, t_3 x_6, y_6, t_6 x_e, y_e, t_e
x_s, y_s, t_0 x_1, y_1, t_1 x_4, y_4, t_4 x_3, y_3, t_3 x_6, y_6, t_6 x_2, y_2, t_2 x_e, y_e, t_e

Figure 4.4 Left: 6 waypoints with the start and end waypoints. Right: Possible sequences generated from waypoints.

4.4). It is important to understand that the time stamps have no relation between different sequences (with the exception of t_0 being the same accross the different individuals), and that the indices do not represent chronology. For example, the first individual has $t_e > t_2 > t_3 > t_0$. The second one has $t_e > t_2 > t_5 > t_1 > t_0$, and so on. These time stamps serve to perform a check for availability such that $\delta = 1$ when the rover is planned to visit. If $\delta = 0$ for any augmented waypoint, then it is skipped, and the algorithm will choose another augmented waypoint instead.

Fitness score: The score used to quantify the quality of the trajectories here is the science score. To compute it, we simply add up the science score of each waypoint in the sequence as in equation 4.2. The goal is to maximize it according to equation 4.7.

The genetic operators used here are selection, crossover, mutation, and they all work similarly to what can be found in the literature [51,85]. A visual representation of the latter operators can be found in Figure 4.5.

Selection: This operator simply selects the best individuals from the population based on their fitness score. As an example from Figure 4.5, the top two individuals are chosen and the third one is discarded.

Crossover: The crossover operator serves to generate offsprings from a pair of selected individual, i.e., the parents. The tool takes all the chromosomes from both parents, removes the time stamps and builds a new individual by assembling the chromosomes sequentially while augmenting them (similar to the Initialize population step). The crossover also ensures that the time windows are respected such that $\delta = 1$ for all augmented waypoints, and that the time limit T is respected. From Figure 4.5, a third individual is generated based on the chromosomes of the first two.

Mutation: The mutation operator introduces randomness into the genetic algorithm to avoid premature convergence. Given a certain probability, a random chromosome in an individual will be changed to another one from the list of waypoints W. Naturally, by removing and inserting a new waypoint implies that the time windows check and time limit T check need to be performed again (just like in the crossover operator). From Figure 4.5, the offspring's second chromosome changed.

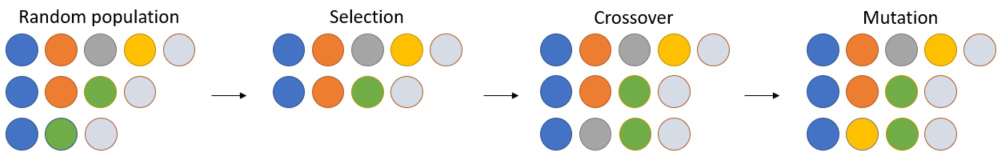

Figure 4.5 Genetic operators in step 1

After every operator has been performed, one iteration is complete (or one generation). Once the predetermined number of iterations is reached, the first step of the genetic algorithm is complete, and the best individual is passed on to the second step.

Step 2

In the second step, with s^* returned from the first step, we attempt to reorder the waypoints in s^* to return the combination of ordered waypoints that will minimize the total distance r_{dist}. This will lead to s^{**}.

Initialize population: The initialization in step two is exactly the same as in step one, except that the length of the individuals will be the same in the population. This is because we only want the waypoints in s^* from the first step. The idea is to simply rearrange their order to minimize the trajectory length. Note that the same checks for availability and time limit are performed as step one. Figure 4.6 illustrates this process.

The genetic operators used in step two are interchange and selection. The latter is the same as the one in step one, but with a different fitness score. An example of an iteration is shown in Figure 4.7.

Interchange: This is an operator that switches the position of two waypoints in a trajectory. It creates new trajectories without deleting the old ones. In Figure 4.7, the interchange operator creates individuals a, b, c by modifying individuals 1, 2 and 3. A time limit and a time window check are performed whenever a new individual is created.

Fitness score: The fitness function used here is simply the total distance of the sequence shown in equation 4.3, and we seek to minimize it as shown in equation 4.8. As mentioned

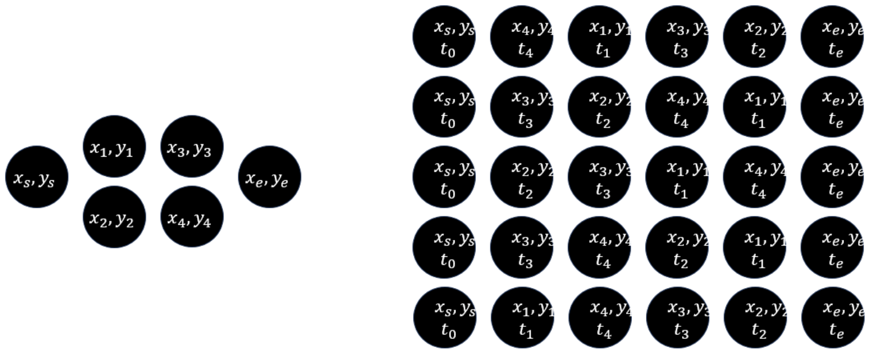
x_s, y_s
x_1, y_1
x_2, y_2
x_3, y_3
x_4, y_4
x_e, y_e
x_s, y_s t_0 — x_4, y_4 t_4 — x_1, y_1 t_1 — x_3, y_3 t_3 — x_2, y_2 t_2 — x_e, y_e t_e
x_s, y_s t_0 — x_3, y_3 t_3 — x_2, y_2 t_2 — x_4, y_4 t_4 — x_1, y_1 t_1 — x_e, y_e t_e
x_s, y_s t_0 — x_2, y_2 t_2 — x_3, y_3 t_3 — x_1, y_1 t_1 — x_4, y_4 t_4 — x_e, y_e t_e
x_s, y_s t_0 — x_3, y_3 t_3 — x_4, y_4 t_4 — x_2, y_2 t_2 — x_1, y_1 t_1 — x_e, y_e t_e
x_s, y_s t_0 — x_1, y_1 t_1 — x_4, y_4 t_4 — x_3, y_3 t_3 — x_2, y_2 t_2 — x_e, y_e t_e

Figure 4.6 Left: 4 waypoints from s^* with the start and end waypoints. Right: Possible sequences generated.

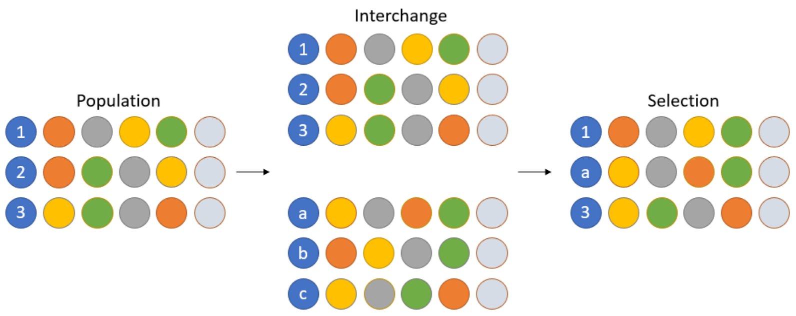

Figure 4.7 Step 2 of the 2-step Genetic Algorithm

above, this will not be the summation of Euclidean distances. Instead, we run the 2D B-RRT* to find feasible trajectories that avoids static obstacles only.

Once the interchange and selection operators have been performed, one generation is complete. Then, once the number of generations is reached, the algorithm returns the best sequence. The output of this second step will be the sequence s^{**} which contains the same waypoints that are scientifically relevant in s^*, but with a minimized distance.

Time assignment

Recall that the lunar environment is dynamic; therefore, a time stamp t_i is associated for every augmented waypoints of the sequence s^{**}. This is done by approximating the time it would take for a lunar rover to complete the sequence using its average speed. Note that the sequence's distance is not the actual distance, but rather an estimate of it based on static obstacles only.

The point of doing this is to have an approximate for the time the rover needs to reach the different sites to rapidly sort out the unfeasible sequences. The specific time will be confirmed by the detailed path planner in the next section by considering both static and dynamic obstacles.

4.5.2 Multi-Directional RRT*

This section describes the multi-directional RRT* (MD-RRT*) algorithm used to plan trajectories between several augmented waypoints.

The MD-RRT* relies on the RRT* [17] as a building block, and also leverages the B-RRT*'s workflow [18] to avoid complex obstacles. Its structure is illustrated in Figure 4.8.

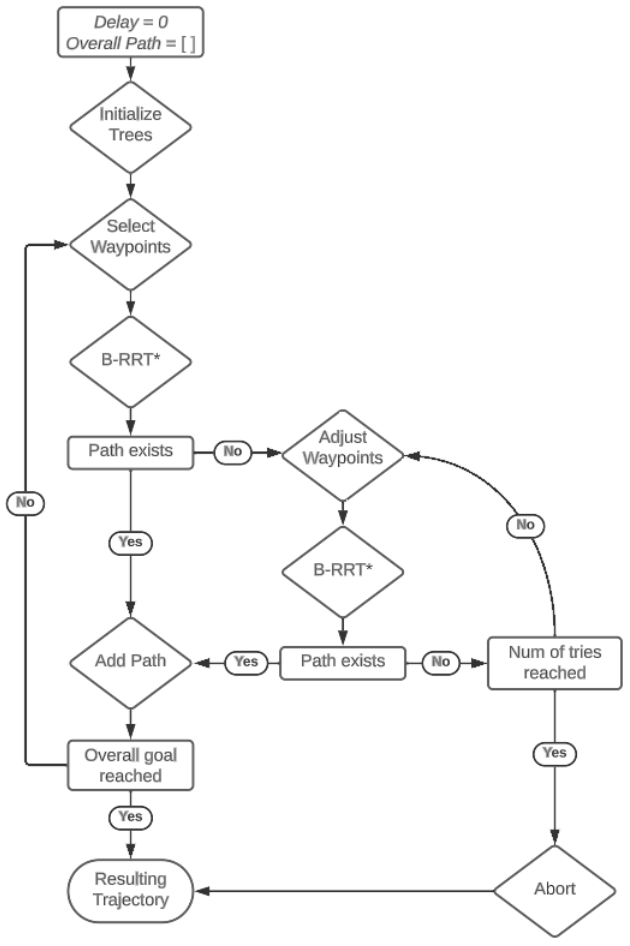

Figure 4.8 MD-RRT* data flow

The following subsections will present the different components of the structure of B-RRT* and of the MD-RRT* respectively.

B-RRT*

The B-RRT* algorithm [18, 19] performs two tree searches between an initial state s_{init} and an end state s_{goal} in order to find a feasible trajectory. The algorithm's flow is shown in Figure 4.9.

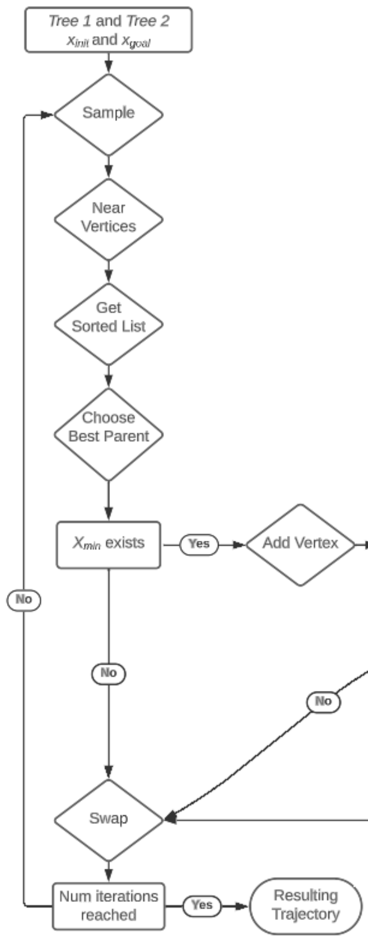

Figure 4.9 B-RRT* data flow

Its inputs contain two search trees with a start s_{init} in one tree and an end state s_{goal} in the other one. The overall cost is initialized at infinity.

Sample: Sampling returns the next point to explore starting from the current point in the obstacle free space, that is $s_{sample} \in S_{free}$. Recall that we are planning in the augmented state space; thus, time is being considered and chronology must be respected. This means that if the sampling is for the search tree T_1 containing s_{init}, the sampled point needs to have a time stamp that is later in time than the current point, i.e. time is moving forward. Conversely, if the sampling is for the search tree T_2 containing s_{goal}, time is going backwards, which means that the sampled point must have a time stamp earlier than the current point. Additionally, the sampling will be random in an independent and uniformly distributed way.

Near Vertices: From the literature [17–19], given a sampled point $s_{sample} \in S_{free}$, a search tree $T_i = (V_i, E_i)$ and a spherical region (ball-like) $B_{x,r}$ with radius r and centered at $s = s_{sample}$, we can define the set of the near vertices as $\{v \in V : v \in B_{s,r}\} \mapsto S_{near} \subseteq V$. Here, $S_{near} = \{v \in V : d(s,v) \leq \gamma(\frac{\log z}{z})^{\frac{1}{l}}\}$, where γ is a constant, z is the number of vertices and l represents the search space dimensions.

Get Sorted List: This function returns a list L_s containing tuples of the form (s_i, C_i, σ_i), sorted in ascending order of the cost C. s_i represent the near vertices, so $s_i \in S_{near}$. C_i represent the sum of the cost of going from the initial vertex of the current search tree to s_i and of the cost of going from s_i to the sampled vertex s_{sample}. In other words, C_i is the cost of the trajectory from the start to s_{sample} by passing by s_i. σ_i represents the trajectory from s_i to s_{sample}.

Choose Best Parent: With the L_s, $s_{min} \in L_s$ will be found. This s_{min} represents the vertex among the near vertices that will make the shortest collision free path from the start s_{init} to s_{sample} (by passing by s_{min}). The collision check will ensure that the trajectory σ_i does not contain any obstacles, that is, for $\sigma_i(u)$ to belong to S_{free}. In other words, $\sigma_i(u) \in S_{free}, \forall u \in [0,1]$. Note that it is possible that s_{min} does not exist.

Once the algorithm reaches this point, two possibilities can happen: either it finds s_{min} or it cannot. If it can, then the algorithm proceeds to add it as a vertex to the tree. If not, the algorithm proceeds with the **Swap** operation.

Add Vertex: Adding a vertex means that the current search tree will save the specified vertex s_{sample}, and save the edge connecting s_{min} to s_{sample}.

Rewire: The rewiring step is what allows this algorithm to be asymptotically optimal, i.e., it will eventually reach the best trajectory [17]. When a vertex is added, a check is performed to all near vertices in S_{near} to see if it is possible to reach any of them by passing by the new vertex in a less costly manner than the current tree configuration. If it is the case for any neighbor, then the previous edge is replaced or "rewired" by the new edge connecting the new vertex to the near vertex in question.

Connect: As the name implies, this step attempts to connect the two trees together. First, after adding the vertex s_{sample}, the **Near Vertices** operation is performed with the other tree (not the current one) to find the nearby vertices on the other tree to s_{sample}. These vertices are then sorted with **Get Sorted List**, and **Choose Best Parent** returns the best one. From there, if this best vertex can connect to s_{sample} without encountering any obstacles, and if the cost of the end-to-end trajectory is lower than the overall cost, then this newly found end-to-end trajectory becomes the best one and is saved. On the other hand, if one of these conditions is not satisfied, then the search continues by swapping with **Swap**.

Update Best Path: This function updates the overall cost to newly found best trajectory, and also saves the best trajectory.

Swap: This swapping function marks the end of an iteration and proceeds with the same operations on the other tree. So, if the current tree searches forward in time, after the swap, the current tree will the one searching backwards in time.

After reaching the specified number of iterations, the best trajectory that the algorithm found is returned as the resulting trajectory, that is, σ_{f_a}.

MD-RRT*

As shown in Figure 4.8, the B-RRT* algorithm is a building block for the MD-RRT*. The operations in the workflow of the MD-RRT* are explained below. It begins by initializing an empty set for the overall trajectory and a variable named *delay* introduced below.

Initialize Trees: This function initializes the trees before beginning the search, by creating them and inserting the right starting vertex. As illustrated in Figure 4.1, there will be $2(m+1)$ trees where m is the number of waypoints (excluding the start and the goal). Moreover, these m waypoints will each be the starting vertex of two trees: one going forward in time, and one going backwards in time.

Select Waypoints: With the trees initialized, a start and an end waypoints are selected for the search. In the event that a feasible trajectory is not found, the algorithm will try to delay the end waypoint in an attempt to provide more time to find σ_{f_a}. This delay is reflected in the variable *delay*, and it will be taken into consideration when selecting the right waypoints.

Now that the start and end waypoints are properly selected, the B-RRT* algorithm is called to perform a search between them. If a trajectory exists, then it is returned and added to the overall trajectory with the **Add Path** function. If not, the waypoints are adjusted based on **Adjust Waypoints**.

Add Path: When B-RRT* returns a feasible trajectory between two waypoints, that is, σ_{f_a}, the trajectory is appended to the overall trajectory set.

Adjust Waypoints: This function is called when the B-RRT* algorithm did not succeed in finding a feasible trajectory, and that the maximum number of retrial attempts has not yet been reached. The algorithm will try to delay the time stamp of the end waypoint, and this delay will be recorded into the *delay* variable as well.

If, on one hand, after adjusting the waypoints, σ_{f_a} is found, then it is added to the overall trajectory through **Add Path**. On the other hand, if the maximum number of retries has been exceeded, then the algorithm will abort the search, and return the resulting trajectory.

Abort: Aborting the MD-RRT* algorithm will simply end the current B-RRT* search, and the algorithm will return the overall trajectory that has been found up until now, i.e., all the σ_{f_a} found so far.

If σ_{f_a} has been added successfully with **Add Path**, then MD-RRT* will proceed with **Select Waypoints** for the next pair of start and end waypoints. This cycle will repeat until the overall ending waypoint is reached. If that is the case, then the algorithm will return the resulting trajectory.

4.6 Results

At the time of writing this article, there is no prior work on multi-directional path planning in a dynamic environment to the best of our knowledge. Hence, results returned by the proposed path planner tool will be compared to manually planned trajectories by operators from CSA.

Table 4.1 contains all the parameters that this path planning tool used to generate trajectories, namely for the number of generations, the population size, the number of strong candidates passed on, the mutation probability for the genetic algorithm, and the rover speed, the number of iterations for the MD-RRT*. Note that all simulations were run on a laptop with an 11th Gen Intel(R) Core(TM) i7-1185G7.

Figure 4.10 shows a path that is planned by an operator in a potential landing zone on the Moon. The different colors from green to red illustrate slope accessibility for the lunar rover. Thus, green implies that the rover will have no problem navigating the terrain. Yellow and orange mean that the rover can still access the zone, but with difficulty. Lastly, red means that the rover should not navigate the zones given the high risk of failure.

The map also has yellow waypoints that represent potential locations of scientific interest.

Table 4.1 Path Planning Tool Parameters and Hyperparameters

Genetic Algorithm				MD-RRT	
# gen.	Pop. size	# strong cand.	Mut. prob. (%)	Rover speed (m/h)	# iter.
500	50	25	100	10	500

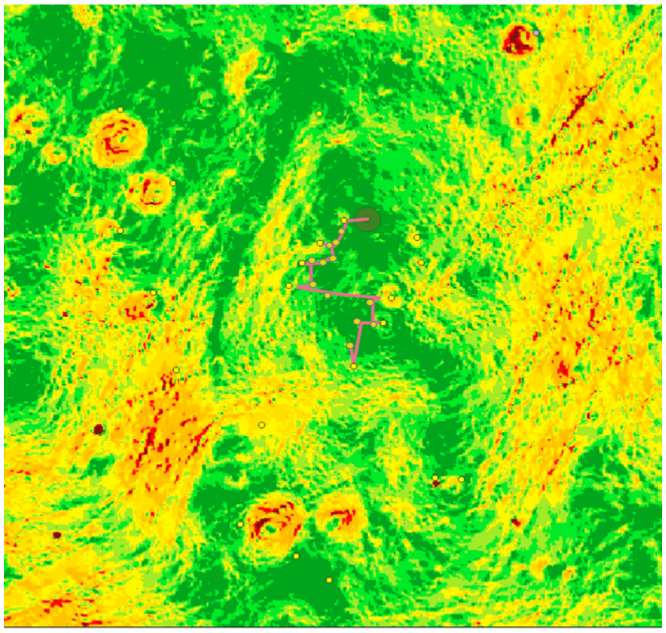

Figure 4.10 Lunar path planned by operator

The operator picked a handful of them and planned a trajectory, given the landing zone outlined by a shaded circle.

Note that the operator did account for the dynamic obstacles in an approximate manner. They made sure that the waypoints had access to the Sun and Earth at the time of travel. Obviously, it is quite tedious to do the check for the entire trajectory; thus, the dynamic environment could only be considered at the waypoints. This shows in a way the value of having this tool perform the check automatically for both static and dynamic aspects of the lunar surface at all times.

Comparatively, Figure 4.11 shows the trajectory returned by the path planner tool. Note that the figure is shown in 2D, but the planning itself happens in 3D to take into account the dynamic environment. The trajectory planned is similar, but more precise than what the operator has planned. This is because of how the tool models the lunar environment, which allows it to generate trajectories with more confidence than the approximate approach of the operator. On one hand, this translates into a lower total distance of the tool-generated trajectory than the operator one. On the other hand, the tool also proposes a trajectory that takes less time to accomplish.

For illustrative reasons, the trajectory in 3D (the rover's position as a function of time) is shown in Figure 4.12. It can be seen that time chronology is respected, since from the start to the finish of the trajectory, the time component is monotonically increasing.

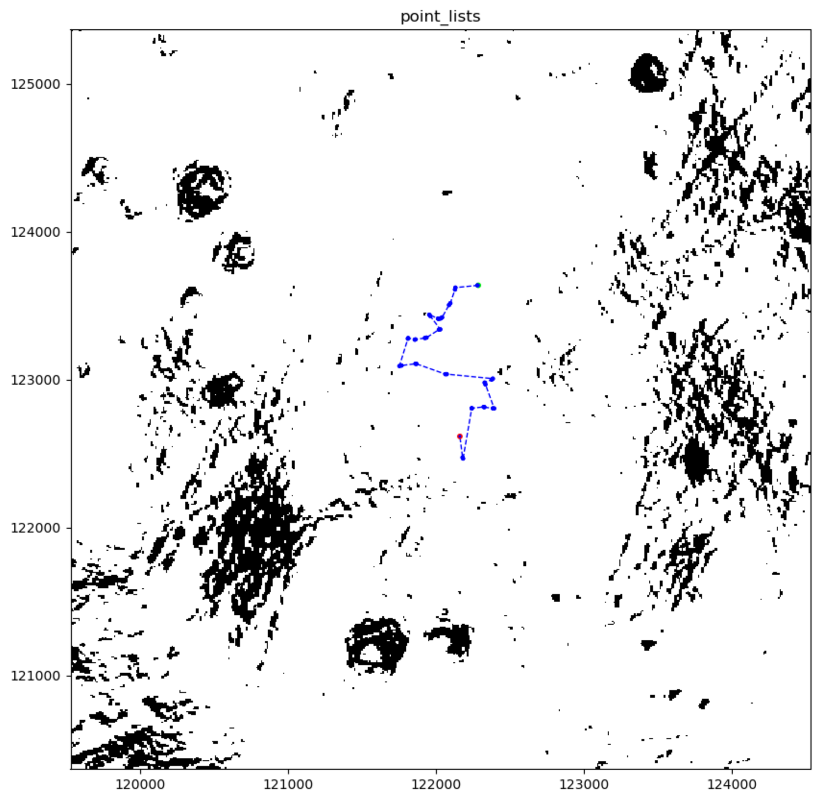

Figure 4.11 Lunar path returned by tool

The same comparison has been done for various potential landing sites and time of landing. The full results can be found in Table 4.4 in the Appendix (section 4.8).

From these results, we can see that the tool returns trajectories that have a lower distance and a lower time overall. Indeed, with the exception of the operator's traverses that are not feasible, all others have lower characteristics. To further support the latter observation, we performed a statistical test to confirm that the results are statistically significant.

To do so, we first subtract the operator's traverses' distance and time from the tool's traverses' distance and time. The data points are shown in Figure 4.13 for the difference in time and in Figure 4.14 for the difference in distance.

With these values, a T-test is performed to verify if the mean of these two differences is greater than 0. From there, it can be said that it is statistically significant that the tool produces better trajectories in terms of distance traveled (T-test, $p<0.0054$, $T=2.740$) and

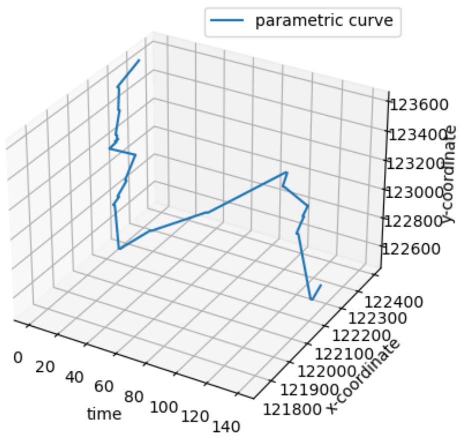

Figure 4.12 Proposed trajectory in 3D

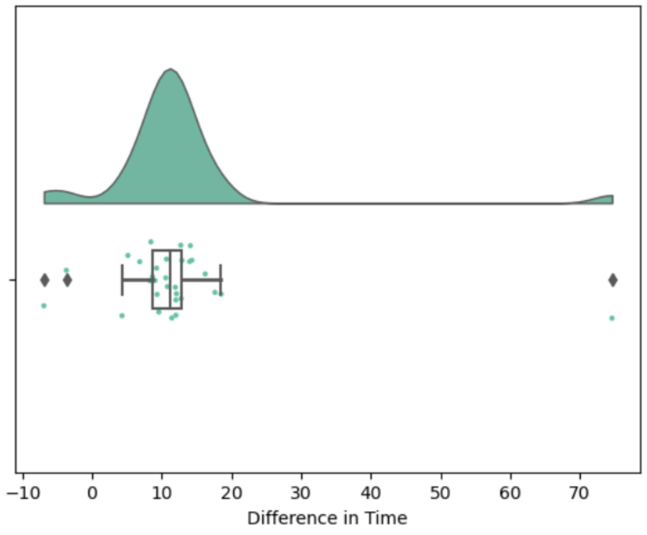

Figure 4.13 Data for the difference in time

time of travel (T-test, p<0.0001, T=5.106).

Lastly, some smaller instance trajectories were solved to optimality using a brute force method, i.e., by testing every possible permutation of the different sites with the current tool. To do so, the tool parameters have been changed to ensure that the returned solution is the best as shown in Table 4.2. Note that the only difference is that the number of generations and iterations have been increased. In addition, a condition is included that stops the search when the quality of the solution does not improve. It was possible to accelerate

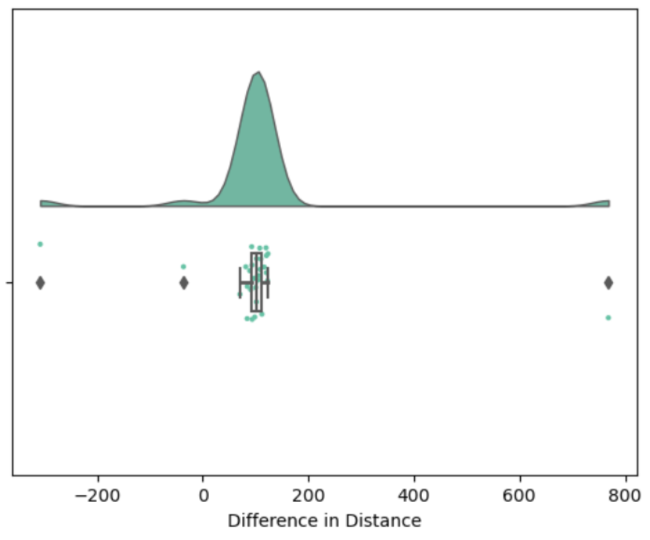

Figure 4.14 Data for the difference in distance

Table 4.2 Path Planning Tool Parameters and Hyperparameters for Optimal Solving

Genetic Algorithm				MD-RRT	
# gen.	Pop. size	# strong cand.	Mut. prob. (%)	Rover speed (m/h)	# iter.
1000	50	25	100	10	1000

the search by pruning the solutions that contained unavailable sites. Although it still took on average three days to get the results for each run, it is still worthwhile to see if the path planner returns something close to the optimal solution. The results are shown in Table 4.3.

From this optimal search, it can be shown that the tool returns the same order of exploration when solving for optimality. This means that the only improvements in the length and time of the path were between the different sites. This can be explained by the increased number of iterations for the MD-RRT* as it is based on the RRT* which guarantees asymptotic optimality. From the three traverses, the optimal solutions seem to have slightly lower distance and time. Although this does not directly translate to the larger instances in which the tool will be used, it is still a vote of confidence in the results that the proposed tool returns for lunar path planning.

4.7 Conclusion

In summary, despite not having any close comparative, we can still show that the tool performs well on cases that operators have done. The results of the tool are better in terms of distance and time than what operators have planned. Additionally, the tool will return feasible trajectories, even if the ones generated by the operator are not, albeit sometimes with a higher distance and time.

This paper introduces a tool for rover multi-point navigation on the Moon. Given the complexity of the task, not much public work has been found in this field. There is no adequate benchmark at the moment of writing this article to compare the work done. Even so, the tool successfully returns feasible trajectories that match the ones manually generated by mission operators of the Canadian Space Agency. In fact, the trajectories returned by the tool are better than the operator ones in terms of distance and time. Moreover, a few small instances have been solved to optimality to show that the tool is returning results that are legitimate. This increases its confidence against larger instances.

The work done in this article will likely serve as an initial benchmark for future endeavors to tackle this problem. Thus, here are some proposed way forward to improve this tool. One can consider including energy consumption in the implementation of this tool to track the rover's energy level. The latter has an influence on the rover's speed. Moreover, the path planning optimization is currently solely distance minimization. A more sophisticated tool could consider other factors such as path smoothness or energy required. Furthermore, the 2-step GA and the MD-RRT* will eventually become obsolete for returning the order of exploration and planning paths. Thus, it is worthwhile to investigate state of the art algorithms to perform these respective tasks. Lastly, the data used to model the lunar environment is currently what is available now. In the future with more precise data, the tool can be rendered that much more accurate.

4.8 Appendix: Experimental Data of Operator and Tool Traverses

Table 4.3 Traverses Solved to Optimality

Run	Optimal		Tool			
#	Dist. (m)	Time (h)	Dist. (m)	Time (h)	# sites	Same order?
14	359.16	128.72	371.29	134.13	8	yes
20	365.31	82.27	381.91	86.12	8	yes
22	288.18	26.14	303.53	28.50	7	yes

Table 4.4 Comparison Between Operator and Tool Trajectories

Run	Operator		Tool		Feasibility		# sites
#	Dist. (m)	Time (h)	Dist. (m)	Time (h)	Operator	Tool	
1	1165.35	161.04	1068.11	148.25	yes	yes	19
2	744.82	74.48	635.13	63.79	yes	yes	12
3	955.98	95.60	839.64	81.59	yes	yes	17
4	630.34	171.69	546.36	158.78	yes	yes	10
5	659.23	90.00	570.39	85.69	yes	yes	11
6	340.98	94.00	649.12	100.91	no	yes	13
7	670.14	50.81	568.41	38.69	yes	yes	12
8	1009.70	90.97	909.63	80.16	yes	yes	18
9	769.10	120.00	670.78	105.68	yes	yes	14
10	681.76	225.00	597.59	208.76	yes	yes	13
11	940.83	94.08	859.43	85.63	yes	yes	16
12	955.03	95.50	831.57	81.38	yes	yes	16
13	659.93	101.99	566.55	92.65	yes	yes	11
14	464.46	142.45	371.29	134.13	yes	yes	8
15	522.24	52.22	558.96	55.90	no	yes	11
16	743.60	118.17	634.91	99.64	yes	yes	15
17	695.63	116.17	625.38	98.53	yes	yes	12
18	1348.95	178.97	580.56	104.36	no	yes	15
19	610.75	125.3	506.64	114.71	yes	yes	10
20	485.69	91.27	381.91	86.12	yes	yes	8
21	874.38	132.89	765.63	123.30	yes	yes	15
22	411.38	39.94	303.53	28.50	yes	yes	7
23	614.56	84.67	513.10	77.82	yes	yes	11
24	1295.54	121.82	1183.61	112.78	yes	yes	20
25	945.38	154.54	824.96	142.50	yes	yes	17
26	642.25	64.23	549.71	54.97	yes	yes	11
27	826.44	82.64	706.76	70.68	yes	yes	14
28	726.91	83.69	636.34	75.02	yes	yes	13
29	1295.54	147.82	1172.97	135.08	yes	yes	21
30	945.38	153.54	824.96	141.50	yes	yes	18

CHAPTER 5 GENERAL DISCUSSION

This chapter elaborates further on the discussion of the different results presented in section 4.6 and section 4.7. The goal is to recapitulate the contributions, to address a new mission requirement, and to explore the impact of this work on the research community.

5.1 Contributions

Chapter 4 presents a path planning algorithm for a lunar rover going to the South pole of the Moon with complex dynamic obstacles when given multiple areas to explore. This work has no close comparatives yet; therefore, the results of the path planner are compared to manual trajectories generated from CSA operators.

From the different experiments, the path planner seems to return better trajectories in terms of time and distance traveled when given the same number of waypoints to explore. It is also noteworthy to mention that the path planner takes a few minutes to generate a trajectory, whereas an operator requires hours, if not days to complete the same workload. This attests to how complex path planning is on the South pole of the Moon. It has also been shown that when using the tool to solve to optimality the smaller instances (brute-force approach), the results are not so far off from what the tool usually returns. Particularly, the order of exploration does not change for the small instances. This increases the confidence of the results of the tool for the larger instances.

Lastly, there were a few cases when the path planner generates longer trajectories. However, this is due to the fact that the operator's manual trajectory was not feasible. The path planner had to figure out detour routes to achieve the original objectives, at the cost of a longer traverse both in terms of time and distance.

5.2 Mission Limitation

The original intent of use of the path planner was to have the 2-step GA run multiple times to deal with the potential re-planning in case of waypoints change. Hence, the path planner was designed to have the first component run quickly, whereas the second part (MD-RRT*) would take longer to compute and validate feasible trajectories.

Recently though, a new requirement came up for the path planner to compute feasible trajectories in around five second. This is due to a mission limitation. It has become clear that the method of operating the lunar rover requires multiple re-planning attempts of different trajectories. The idea is then to run very quickly the entire path planner (both the first and second component) several times to investigate the various possibilities. This will certainly change the dynamics of the path planner, and adjustments to both the 2-step GA and the MD-RRT* will be necessary to respect this requirement. Alas, this will be marked as future work and outside the scope of this thesis due to the timing of this limitation.

5.3 Impact on Research Community

The work presented in this thesis is first and foremost a steppingstone for research related to future space missions. As mentioned in section 1.1, the initiative of re-establishing human presence on the Moon has been rekindled. With the multiple lunar missions already scheduled, one can easily imagine the numerous possibilities for future missions on the Moon and further beyond [1, 2].

Undoubtedly, extra-terrestrial rovers will play a crucial role in those. Since there is no public work available at the moment regarding extra-terrestrial rover path planning of numerous waypoints in dynamic environments, this thesis may serve as a benchmark to develop more advanced path planners for more complex missions. Specifically, we can think of missions that require more automation since the rover will be less in contact with human operators, or missions that will happen in more challenging environment than the Moon.

CHAPTER 6 CONCLUSION

The thesis is now concluded with first a summary of works on the work done and on the main results. Then, limitations of this work will be briefly discussed alongside future research paths and approaches.

6.1 Summary of Works

The initial goal of the work was to create a path planner algorithm for a lunar rover to generate feasible trajectories to navigate the south pole of the Moon, given multiple areas of interest to explore with no particular order. Despite containing many clues or answers to questions related to the formation of our planet, it is quite difficult to navigate that area due to the terrain and the dynamic obstacles.

Overall, the proposed novel path planner satisfies the research objectives outlined in section 1.3. Lunar environment modeling has been taken care of internally at CSA and files containing relevant data on the lunar topography and dynamic conditions are provided. The 2-step genetic algorithm can then make sense of all the unordered waypoints that operators wish to explore on the Moon by returning an order of exploration that satisfies the availability of those areas of interest. Finally, the MD-RRT* can validate the proposed order of exploration by generating a feasible trajectory through the waypoints which avoids all obstacles.

In general, the resulting trajectories of the path planner are shorter in terms of distance and time when compared to manual trajectories generated by CSA operators. Moreover, for small instances, it is shown that the tool can return solutions that match fairly well to optimal solutions. Despite producing those trajectories in record time, there are limitations.

6.2 Limitations

While the path planner certainly compares favorably to the manual method, it is not a perfect. Here is a list of limitations that the proposed tool has:

- The lunar rover is assumed to have constant speed. Rover speed is dependent on many variables, notably the slope terrain, terrain granularity, Sun luminosity, Sun incidence angle with the solar panels, remaining battery charge, etc. Due to the complexity of modelling, the assumption of constant speed for the rover is made.

- The energy consumption of the lunar rover is not taken into account currently. Similarly to the rover speed, energy consumption will depend on many variables whose relationship with energy usage are not clearly understood yet. By taking it into account, trajectories length will be limited and recharge points will need to be integrated. However, by doing so, another consideration needs to be introduced which is the battery recharge rate. As expected, this will also depend on multiple variables.
- For the sampling portion of the MD-RRT*, there is currently a fixed time increment. Though, not a bad baseline to begin the work with, it is possible that a variable time increment works better.
- The current strategy for the 2-step GA is to estimate the distances between the different waypoints with a 2D version of the B-RRT*. It is a better place to start than simply taking the Euclidean distance, since the static obstacles are taken into account, but in areas with lots of dynamic obstacles, this may be complicated in the long run.
- There is no explicit path smoothing that is done for the final trajectory.

6.3 Future Research

The future of this work can happen on multiple fronts. Besides improving on the limitations mentioned above, the biggest concern at the moment is the new requirement of a five second computation time. This will need to be investigated through parallelization of the 2-step genetic algorithm, and perhaps the MD-RRT* as well. For the latter, it may be more challenging, since the planned trajectory between two waypoints will necessarily impact the starting time of the next set of waypoints. Due to this, the workflow is sequential, and it becomes unclear how parallelization can be implemented. On another note, guided sampling may be useful to consider for the MD-RRT*. Instead of randomly sampling the environment, incentives may be given to the sampler to encourage a faster conversion of the B-RRT*.

Furthermore, the tool currently adapts to inconclusive trajectories by pushing back the end waypoint by one hour. The intent is to provide more time for the lunar rover in case of complex obstacles, and the maximum allowable delay is 24 hours before declaring a failure to find a suitable trajectory. Being the default method, this can certainly be improved. Unfortunately, it is still unclear how trajectory generation failure should be reported. The way forward for this aspect of the work is to further compare the manual trajectories with the tool. This would be the only way to discover "limit" cases, i.e., instances where the tool disagrees with the manual trajectories. By understanding those, we can improve on the robustness of the tool and on how to better handle cases where trajectories are not found.

Then, for the 2-step genetic algorithm, the tool uses an approximation of the distance based on the 2D MD-RRT*. This means that the distances between the different waypoints are computed without considering the dynamic obstacles. Despite being more realistic than a simple Euclidean distance, it is not quite representative either of the situation. This boils down to a compromise between computational speed and accuracy of the result. Moreover, the complexity here is that the time stamp of each waypoint is unknown and depends on the order of exploration, which happens to be the output of what the distance estimates will serve to compute. There could be a better approach to estimate the distances between the different waypoints. For example, the tool can take into account dynamic obstacles that do not vary as much in the concerned time frame. However, this will come at the cost of a higher computational cost. The way forward for this issue would be to consult with the operators of CSA to confirm their preference.

Lastly, as briefly mentioned, the limitations can be improved upon. Therefore, future research can involve investigating rover speed, energy consumption, variable time sampling and path smoothing. These limitations are considered minor compared to the previous points mentioned. However, given the remaining time prior to the launch of the lunar rover, it is feasible to improve on those aspects.

REFERENCES

[1] S. Creech, J. Guidi, and D. Elburn, “Artemis: An overview of nasa’s activities to return humans to the moon,” in *2022 IEEE Aerospace Conference (AERO)*, 2022, pp. 1–7.

[2] M. Picard, “An overview of the csa recent activities in space robotics,” 2019.

[3] M.-J. Potvin, M. Cantin Halpin, M. Chila, and A. Sura, “Peekbot: a 3d printed thermoplastic rover to survive the lunar night,” 2021.

[4] M. K. Barker, E. Mazarico, G. A. Neumann, D. E. Smith, M. T. Zuber, J. W. Head, and X. Sun, “A new view of the lunar south pole from the lunar orbiter laser altimeter (lola),” *The Planetary Science Journal*, vol. 4, no. 9, p. 183, sep 2023. [Online]. Available: https://dx.doi.org/10.3847/PSJ/acf3e1

[5] E. Mazarico, M. K. Barker, and J. B. Nicholas, “Advanced illumination modeling for data analysis and calibration. application to the moon,” *Advances in Space Research*, vol. 62, no. 11, pp. 3214–3228, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0273117718306501

[6] E. Mazarico, G. Neumann, D. Smith, M. Zuber, and M. Torrence, “Illumination conditions of the lunar polar regions using lola topography,” *Icarus*, vol. 211, no. 2, pp. 1066–1081, 2011.

[7] J.-K. Huang, Y. Tan, D. Lee, V. R. Desaraju, and J. W. Grizzle, “Informable multi-objective and multi-directional rrt* system for robot path planning,” 2022.

[8] M. Li and H. Zhang, “Auv 3d path planning based on a* algorithm,” in *2020 Chinese Automation Congress (CAC)*, 2020, pp. 11–16.

[9] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, “Path planning and trajectory planning algorithms: A general overview,” *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, pp. 3–27, 2015.

[10] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, “Path planning for autonomous mobile robots: A review,” *Sensors*, vol. 21, no. 23, p. 7898, 2021.

[11] J. H. Bai and Y.-J. Oh, “Global path planning of lunar rover under static and dynamic constraints,” *International Journal of Aeronautical and Space Sciences*, pp. 1–9, 2020.

[12] M. Sutoh, M. Otsuki, S. Wakabayashi, T. Hoshino, and T. Hashimoto, "The right path: Comprehensive path planning for lunar exploration rovers," *IEEE Robotics & Automation Magazine*, vol. 22, no. 1, pp. 22–33, 2015.

[13] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE international conference on robotics and automation.* IEEE, 1994, pp. 3310–3317.

[14] S. Kadry, G. Alferov, V. Fedorov, and A. Khokhriakova, "Path optimization for d-star algorithm modification," in *AIP Conference Proceedings*, vol. 2425, no. 1. AIP Publishing, 2022.

[15] X. Yu, Q. Huang, P. Wang, and J. Guo, "Comprehensive global path planning for lunar rovers," in *2020 3rd International Conference on Unmanned Systems (ICUS).* IEEE, 2020, pp. 505–510.

[16] P. Xin, X. Wang, X. Liu, Y. Wang, Z. Zhai, and X. Ma, "Improved bidirectional rrt* algorithm for robot path planning," *Sensors*, vol. 23, no. 2, p. 1041, 2023.

[17] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, no. 2, 2010.

[18] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," 2013.

[19] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[20] M. Tian and J. Yu, "Progressive rapidly-exploring random tree for global path planning of robots," in *2023 9th International Conference on Control, Automation and Robotics (ICCAR)*, 2023, pp. 388–393.

[21] Y. Wang, Q. Yuan, Y. Guo, and W. Han, "Path planning for lunar rover based on bi-rrt algorithm," in *2021 40th Chinese Control Conference (CCC)*, 2021, pp. 4211–4216.

[22] X. Yu, P. Wang, and Z. Zhang, "Learning-based end-to-end path planning for lunar rovers with safety constraints," *Sensors*, vol. 21, no. 3, p. 796, 2021.

[23] R. Hu and Y. Zhang, "Fast path planning for long-range planetary roving based on a hierarchical framework and deep reinforcement learning," *Aerospace*, vol. 9, no. 2, p. 101, 2022.

[24] L. Mantoani, C. J. Pérez-del Pulgar-Mancebo, G. J. Luque-Polo *et al.*, "Path planning with far-away obstacles detection under uncertainty." 2023.

[25] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018, pROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING IN DATA SCIENCES, ICDS2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091830125X

[26] A. Ram, G. Boone, R. Arkin, and M. Pearce, "Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation," *Adaptive behavior*, vol. 2, no. 3, pp. 277–305, 1994.

[27] D. Cook, "Evolved and timed ants: Optimizing the parameters of a time-based ant system approach to the traveling salesman problem using a genetic algorithm." *Computer Science Department, New Mexico State University, USA*, 2000.

[28] A. Elshamli, H. Abdullah, and S. Areibi, "Genetic algorithm for dynamic path planning," in *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513)*, vol. 2, 2004, pp. 677–680 Vol.2.

[29] A. Tuncer and M. Yildirim, "Dynamic path planning of mobile robots with improved genetic algorithm," *Computers & Electrical Engineering*, vol. 38, no. 6, pp. 1564–1572, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790612001255

[30] P. Shi and Y. Cui, "Dynamic path planning for mobile robot based on genetic algorithm in unknown environment," in *2010 Chinese control and decision conference.* IEEE, 2010, pp. 4325–4329.

[31] B. Patle, A. Pandey, D. Parhi, A. Jagadeesh *et al.*, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.

[32] M. Elhoseny, A. Tharwat, and A. E. Hassanien, "Bezier curve based path planning in a dynamic field using modified genetic algorithm," *Journal of Computational Science*, vol. 25, pp. 339–350, 2018.

[33] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.

[34] L. Lu and D. Gong, “Robot path planning in unknown environments using particle swarm optimization,” in *2008 Fourth International Conference on Natural Computation*, vol. 4. IEEE, 2008, pp. 422–426.

[35] J. Cao, “Robot global path planning based on an improved ant colony algorithm,” *Journal of Computer and Communications*, vol. 4, no. 2, pp. 11–19, 2016.

[36] M. A. Kreslavsky and J. W. Head III, “Kilometer-scale roughness of mars: Results from mola data analysis,” *Journal of Geophysical Research: Planets*, vol. 105, no. E11, pp. 26 695–26 711, 2000.

[37] P. Tompkins, A. Stentz, and D. Wettergreen, “Global path planning for mars rover exploration,” in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No. 04TH8720)*, vol. 2. IEEE, 2004, pp. 801–815.

[38] M. Ono, T. J. Fuchs, A. Steffy, M. Maimone, and J. Yen, “Risk-aware planetary rover operation: Autonomous terrain classification and path planning,” in *2015 IEEE aerospace conference*. IEEE, 2015, pp. 1–10.

[39] G. Ishigami, K. Nagatani, and K. Yoshida, “Path planning for planetary exploration rovers and its evaluation based on wheel slip dynamics,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 2361–2366.

[40] S. Swinton, J.-H. Ewers, E. McGookin, D. Anderson, and D. Thomson, “A novel methodology for autonomous planetary exploration using multi-robot teams,” *arXiv preprint arXiv:2405.12790*, 2024.

[41] H. Ali, D. Gong, M. Wang, and X. Dai, “Path planning of mobile robot with improved ant colony algorithm and mdp to produce smooth trajectory in grid-based environment,” *Frontiers in neurorobotics*, vol. 14, p. 44, 2020.

[42] Z. Cheng, H. Zhang, and Q. Zhao, “The method based on dijkstra of multi-directional ship’s path planning,” in *2020 Chinese Control And Decision Conference (CCDC)*. IEEE, 2020, pp. 5142–5146.

[43] D. Henrich, C. Wurll, and H. Wörn, “Multi-directional search with goal switching for robot path planning,” in *Tasks and Methods in Applied Artificial Intelligence: 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA-98-AIE Benicàssim, Castellón, Spain, June 1–4, 1998 Proceedings, Volume II 11*. Springer, 1998, pp. 75–84.

[44] K. Qian, Y. Liu, L. Tian, and J. Bao, "Robot path planning optimization method based on heuristic multi-directional rapidly-exploring tree," *Computers & Electrical Engineering*, vol. 85, p. 106688, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790620305437

[45] T. W. Manikas, K. Ashenayi, and R. L. Wainwright, "Genetic algorithms for autonomous robot navigation," *IEEE Instrumentation & Measurement Magazine*, vol. 10, no. 6, pp. 26–31, 2007.

[46] M. Jünger, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," *Handbooks in operations research and management science*, vol. 7, pp. 225–330, 1995.

[47] K. Ilavarasi and K. S. Joseph, "Variants of travelling salesman problem: A survey," in *International conference on information communication and embedded systems (ICICES2014)*. IEEE, 2014, pp. 1–7.

[48] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations.* Springer Science & Business Media, 2006, vol. 12.

[49] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy," *Computer Science Review*, vol. 40, p. 100369, 2021.

[50] M. López-Ibáñez, C. Blum, J. W. Ohlmann, and B. W. Thomas, "The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization," *Applied Soft Computing*, vol. 13, no. 9, pp. 3806–3815, 2013.

[51] H. Abidi, K. Hassine, and F. Mguis, "Genetic algorithm for solving a dynamic vehicle routing problem with time windows," in *2018 International Conference on High Performance Computing & Simulation (HPCS).* IEEE, 2018, pp. 782–788.

[52] P. Toth and D. Vigo, "An overview of vehicle routing problems," *The vehicle routing problem*, pp. 1–26, 2002.

[53] M. Jünger, G. Reinelt, and G. Rinaldi, "Chapter 4 the traveling salesman problem," in *Network Models*, ser. Handbooks in Operations Research and Management Science. Elsevier, 1995, vol. 7, pp. 225–330. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801215

[54] R. Matai, S. P. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling salesman problem, theory and applications*, vol. 1, no. 1, pp. 1–25, 2010.

[55] C. Chauhan, R. Gupta, and K. Pathak, "Survey of methods of solving tsp along with its implementation using dynamic programming approach," *International journal of computer applications*, vol. 52, no. 4, 2012.

[56] D. L. Applegate, *The traveling salesman problem: a computational study.* Princeton university press, 2006, vol. 17.

[57] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 61–63, 1962.

[58] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Industrial and Applied mathematics*, vol. 10, no. 1, pp. 196–210, 1962.

[59] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[60] R. Fontaine, J. Dibangoye, and C. Solnon, "Exact and anytime approach for solving the time dependent traveling salesman problem with time windows," *European Journal of Operational Research*, vol. 311, no. 3, pp. 833–844, 2023.

[61] S. Miki, D. Yamamoto, and H. Ebara, "Applying deep learning and reinforcement learning to traveling salesman problem," in *2018 international conference on computing, electronics & communications engineering (ICCECE).* IEEE, 2018, pp. 65–70.

[62] P. Kowalik, G. Sobecki, P. Bawoł, and P. Muzolf, "A flow-based formulation of the travelling salesman problem with penalties on nodes," *Sustainability*, vol. 15, no. 5, p. 4330, 2023.

[63] M. Goycoolea, "Cutting planes for large mixed integer programming models," Ph.D. dissertation, Citeseer, 2006.

[64] F. Li, B. Golden, and E. Wasil, "Solving the time dependent traveling salesman problem," in *The Next Wave in Computing, Optimization, and Decision Technologies.* Springer, 2005, pp. 163–182.

[65] A. R. Kuroswiski, H. B. Pires, A. Passaro, L. N. Frutuoso, and E. L. F. Senne, "Hybrid genetic algorithm and mixed integer linear programming for flying sidekick tsp," *arXiv preprint arXiv:2304.13832*, 2023.

[66] C. Chekuri and K. Quanrud, "Fast approximations for metric-tsp via linear programming," *arXiv preprint arXiv:1802.01242*, 2018.

[67] M. Diaby and M. H. Karwan, *Advances in combinatorial optimization: linear programming formulations of the traveling salesman and other hard combinatorial optimization problems.* World Scientific, 2016.

[68] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems," *Mathematical programming*, vol. 97, pp. 91–153, 2003.

[69] ——, *Finding cuts in the TSP (A preliminary report).* Citeseer, 1995, vol. 95.

[70] ——, "Tsp cuts which do not conform to the template paradigm," *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, pp. 261–303, 2001.

[71] D. Applegate, R. Bixby, W. Cook, and V. Chvátal, "On the solution of traveling salesman problems," 1998.

[72] D. Applegate, W. Cook, and A. Rohe, "Chained lin-kernighan for large traveling salesman problems," *Informs journal on computing*, vol. 15, no. 1, pp. 82–92, 2003.

[73] G. Kizilateş and F. Nuriyeva, "On the nearest neighbor algorithms for the traveling salesman problem," in *Advances in Computational Science, Engineering and Information Technology: Proceedings of the Third International Conference on Computational Science, Engineering and Information Technology (CCSEIT-2013), KTO Karatay University, June 7-9, 2013, Konya, Turkey-Volume 1.* Springer, 2013, pp. 111–118.

[74] M. Hahsler and K. Hornik, "Tsp—infrastructure for the traveling salesperson problem," *Journal of Statistical Software*, vol. 23, pp. 1–21, 2008.

[75] K. Genova and D. P. Williamson, "An experimental evaluation of the best-of-many christofides' algorithm for the traveling salesman problem," *Algorithmica*, vol. 78, no. 4, pp. 1109–1130, 2017.

[76] B. Chandra, H. Karloff, and C. Tovey, "New results on the old k-opt algorithm for the traveling salesman problem," *SIAM Journal on Computing*, vol. 28, no. 6, pp. 1998–2029, 1999.

[77] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: a case study," *Local search in combinatorial optimization*, pp. 215–310, 1997.

[78] X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," *Applied Soft Computing*, vol. 11, no. 4, pp. 3680–3689, 2011.

[79] E. H. Aarts, J. H. Korst, and P. J. van Laarhoven, "A quantitative analysis of the simulated annealing algorithm: A case study for the traveling salesman problem," *Journal of Statistical Physics*, vol. 50, pp. 187–206, 1988.

[80] R. Gan, Q. Guo, H. Chang, and Y. Yi, "Improved ant colony optimization algorithm for the traveling salesman problems," *Journal of Systems Engineering and Electronics*, vol. 21, no. 2, pp. 329–333, 2010.

[81] M. Malek, M. Guruswamy, M. Pandya, and H. Owens, "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 21, pp. 59–84, 1989.

[82] S. Basu, "Tabu search implementation on traveling salesman problem and its variations: a literature survey," 2012.

[83] G. C. Onwubolu, B. Babu, and M. Clerc, "Discrete particle swarm optimization, illustrated by the traveling salesman problem," *New optimization techniques in engineering*, pp. 219–239, 2004.

[84] E. F. Goldbarg, M. C. Goldbarg, and G. R. de Souza, "Particle swarm optimization algorithm for the traveling salesman problem," *Traveling Salesman Problem*, vol. 1, pp. 75–96, 2008.

[85] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, pp. 8091–8126, 2021.

[86] X.-A. Dou, Q. Yang, X.-D. Gao, Z.-Y. Lu, and J. Zhang, "A comparative study on crossover operators of genetic algorithm for traveling salesman problem," in *2023 15th International Conference on Advanced Computational Intelligence (ICACI)*, 2023, pp. 1–8.

[87] K. Deb, R. B. Agrawal *et al.*, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.

[88] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and evolutionary computation*, vol. 1, no. 1, pp. 32–49, 2011.

[89] A. Khanra, T. Pal, M. K. Maiti, and M. Maiti, "Multi-objective four dimensional imprecise tsp solved with a hybrid multi-objective ant colony optimization-genetic algorithm with diversity," *Journal of Intelligent & Fuzzy Systems*, vol. 36, no. 1, pp. 47–65, 2019.

[90] A. Hussain, Y. S. Muhammad, M. Nauman Sajid, I. Hussain, A. Mohamd Shoukry, S. Gani *et al.*, "Genetic algorithm for traveling salesman problem with modified cycle crossover operator," *Computational intelligence and neuroscience*, vol. 2017, 2017.

[91] K. Qian, Y. Liu, L. Tian, and J. Bao, "Robot path planning optimization method based on heuristic multi-directional rapidly-exploring tree," *Computers & Electrical Engineering*, vol. 85, p. 106688, 2020.

[92] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, "The traveling salesman problem," in *The Traveling Salesman Problem.* Princeton university press, 2011.

[93] K. L. Hoffman, M. Padberg, G. Rinaldi *et al.*, "Traveling salesman problem," *Encyclopedia of operations research and management science*, vol. 1, pp. 1573–1578, 2013.