Solving Combinatorial Space-Routing Problems Using Mixed-Integer Linear Problem Solvers

Jasper Slimmens

Delft University of Technology

TU Delft

Solving Combinatorial Space-Routing Problems Using Mixed-Integer Linear Problem Solvers

by

Jasper Slimmens

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday the 18th of January 2024.

Student number: 4286324
Project duration: March 2022 – January 2024
Thesis committee:

Dr.ir. Dominic Dirkx	TU Delft, Chair
Dr. Angelo Cervone	TU Delft, Examiner
Ir. Kevin Cowan MBA	TU Delft, Supervisor

TU Delft

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
AAS	American Astronautical Society
AIAA	American Institute of Aeronautics and Astronautics
BILP	Binary-Integer Linear Programming
GACO	Extended Ant Colony Optimisation
GTOC	Global Trajectory Optimisation Competition
MILP	Mixed-Integer Linear Programming
MSU	Moscow State University
PyGMO	Python Parallel Global Multiobjective Optimizer
PySCIPOpt	Python Interface for the SCIP Optimization Suite
SCIP	Solving Constraint Integer Programs
TUDAT	Technical University Delft Astrodynamics Toolbox

List of Symbols

Symbol	Definition	Unit
AU	Astronomical Unit	[km]
$C_{\max}$	Cost budget	[km/s]
C_{opt}	Cost of the best known solution	[km/s]
ΔV	Velocity change	[km/s]
$\vec{r}$	Position vector	
Δr	Absolute difference in radial distance	[km]
Δx	Absolute difference in x position	[km]
Δy	Absolute difference in y position	[km]
Δz	Absolute difference in z position	[km]
μ_S	Gravitational parameter of the Sun	[km^3/s^2]

1

Introduction

With the continued development of space exploration and research missions, the need to be able to predict and plan journeys through space is ever-growing. Since humans first started to make these kinds of calculations, the resources to do so have developed tremendously. Available computational power is many orders of magnitude larger, and methods to calculate and simulate astrodynamic trajectories have made leaps. As both the interest and ability in space exploration keep growing, doors are opening to new types of space missions. Historically, much work has been done into the design and optimisation of these single leg transfers both for ballistic and low-thrust cases [1, 2, 3, 4, 5, 6]. This research is still relevant and integral to solving any space-routing problem, but it has become a smaller part of a larger type of problem.

With the increasing technological ability of modern spacecraft comes a necessity for more complex mission designs. Instead of cases where a journey to a single specific target needs to be simulated, missions could target large sets of objects or points of interest. Now, a much larger and more complex problem must be solved, where targets and trajectories must be selected to design an optimal mission. Relevant and realistic, complex space-routing problems can be found in the challenges posed in the recurring Global Trajectory Optimisation Competition (GTOC). Examples are problem settings where the maximum number of Near Earth Asteroids as possible need to be visited [7, 8, 9], or a large number of debris objects in orbit around Earth needs to be cleaned up [10]. These types of problems can be classed as combinatorial optimisation problems, and solving them efficiently requires implementing state-of-the-art optimisation methods that might not have been used in a space flight context before.

A combinatorial problem refers to a type of problem in mathematics and computer science that involves making selections, arrangements, or combinations from a finite set of objects or elements according to specific rules or constraints. It involves counting, organising, and analysing various possibilities or configurations arising from a given set of objects. One of the best-known examples of a combinatorial problem would be the ‘travelling salesperson’ problem [11, 12, 13]. Combinatorial problems have been studied and solved [14, 15, 16] since long before space flight was born, and powerful tools exist to solve them. This research focuses on solving complex space routing problems that adhere to the combinatorial problem definition. Specifically, this means that the problem consists of a finite set of available targets which are of interest and a set of boundary conditions, constraints and optimisation criteria. These include dynamical models, budgets like time and fuel, and the ultimate mission objectives.

It is essential to develop robust frameworks and methods to handle these larger and more complex problems and solve them to satisfactory optimality. Ideally, these new solving methods build upon or extend the already available tools. In an effort to progress the ability to solve combinatorial space routing problems, this thesis studies the possibility of effectively moulding the problem into a linear form that can be solved with available state-of-the-art solving tools. Specifically, mixed-integer linear problem

(MILP) solving tools like SCIP [17], which is known to be very well suited to solve combinatorial problems [18, 19, 20]. Instead of linearising the problem as a whole, splitting the problem into multiple linear sub-problems is considered. Solving these sub-problems in sequence should provide a valid solution to the entire problem instance. Sub-problems focusing on smaller parts of the problem make linearising easier and help keep the computational requirements low as more minor instances are fed into the MILP solver. This report will study an effective way to split up the main combinatorial routing problem into linear sub-parts, as well as the optimal SCIP setup for solving these individual sub-problems. The sub-problems are defined as follows:

- Fixed Budget: A subset of targets is selected based on a fixed ΔV budget.
- Full Tour: The selected targets are ordered by finding the cheapest trajectory through all targets in the subset.
- Fixed Tour: The flight time for every leg of the trajectory is optimised for a minimal total ΔV.

These parts are integrated into a global optimisation algorithm using the PyGMO [21] optimisation library. In this global optimisation algorithm, the effectiveness of sub-problems and their integration toward finding the final solution is studied. The integration of the sub-steps and possible simplifications that can be made are studied. One possible simplification is to skip solving the Full Tour sub-problem, as reordering the targets seems to have little impact on the final result. Another consideration is between solving every evaluation within the global optimisation algorithm to feasibility or propagating infeasible solutions instead. After solving the three sub-problems, a solution is found which has a certain ΔV cost. This cost may be higher than the budget. The choice is between solving the sub-problems again to find a solution within the budget or marking the result as infeasible after one iteration and moving to the next evaluation instead. Also, the possibility of seeding initial populations with known values instead of randomly generated ones is studied. This will result in a proposal for a global optimisation algorithm to solve combinatorial space routing problems, for which results will be compared to known solutions from GTOC4 [8].

1.1. Research objectives

The purpose of this paper is to study novel methods utilising MILP-solving tools in the process of solving complex space routing problems. For this purpose, a new solving framework, implementing MILP solving techniques, will be created. As part of this study, the following questions are formulated:

- How can MILP-solving techniques be used to solve complex combinatorial space routing problems?

  To allow the use of these specific techniques, the space routing problem needs to be reformulated in a linear form. Instead of linearising the full problem at once, it is split into multiple linear sub-problems. In the study of these sub-problems, the following sub-questions are considered:

  - What are the primary sub-problems, and what is their function?
  - What SCIP settings provide the best performance for solving the isolated sub-problems?
- How do further simplifications to the solving algorithm impact its performance?

  In the process of designing the framework, multiple areas for possible algorithm simplifications were identified. The effectiveness of these simplifications is studied. The following sub-questions are considered:

  - What is the impact of removing the Full-Tour solving step from the algorithm?
  - What is the impact of propagating infeasible individuals instead of iterating every individual to be feasible?
- What is the effect of seeding the initial population with constant values instead of randomly generated values?

  Including known or probable solutions in the initial population for a global optimisation algorithm can greatly affect performance. In this case, the effect of including individuals with a constant

value for all parameters is studied instead of generating all parameters randomly. The following sub-questions are considered:

  - What number or proportion of the initial population would optimally be seeded?

- How do the results from the proposed methodology compare with benchmark solutions from GTOC4?

  To analyse the accuracy and effectiveness of the new solver implementation, GTOC4 results [22] are used as benchmarks. The problem definition of GTOC4 will be used to set up specific test cases for which the optimal solution is known.

1.2. Report structure

This thesis report consists of two main parts. First, in Chapter 2, a paper manuscript is included, which presents the core of the research work. The following chapters then provide additional information on the content provided in the paper. In Chapter 3, the methods and models used regarding space-flight dynamics are discussed. Chapter 4 gives additional explanations on the use and implementations for optimising MILPs and the verification and validation of those implementations. Chapter 5 contains further information on the implementation and analysis of the global optimisation algorithm. Finally, conclusions regarding the research objectives and recommendations for future work are presented in Chapter 6.

2

Paper

This chapter contains the paper manuscript with the main description of the problem and the results. The paper has been written with the format of the AAS/AIAA Astrodynamics Specialist Conference[1]

[1] http://www.univelt.com/FAQ.html#SUBMISSION, online accessed on 23-5-2023

(Preprint) IAA-AAS XX-XXX

SOLVING COMBINATORIAL SPACE-ROUTING PROBLEMS USING MIXED-INTEGER LINEAR PROBLEM SOLVERS

Jasper Slimmens*and Kevin Cowan†

With the development of space research into novel areas, new complex problems arise. The interest in solving space routing problems considering large numbers of targets has recently grown. This paper proposes a novel method to solve the optimal trajectory in such combinatorial space routing problems. This paper focuses on a global optimisation algorithm implemented to solve the problem posed in the 4th Global Trajectory Optimisation Competition (GTOC4). The solution is a trajectory of multiple legs, where each leg links two targets and has a specific flight time. To enable the use of the powerful mixed-integer linear problem solver software, Solving Constraint Integer Programs (SCIP), the routing problem concerned with visiting as many target bodies with a predetermined fuel and time budget is split into linear sub-problems. The Fixed Budget sub-problem selects a subset of the given set of targets. The Full Tour sub-problem orders the targets in the subset, and the Fixed Tour sub-problem optimises the flight time for every leg of the given trajectory to find the solution with the lowest total fuel consumption. Each of these sub-problems is formulated in a linear form and is solved using SCIP. The global optimisation algorithm evolves a population where every individual exists of a set of initial guesses for the time of flight values. Analysis shows that initialising this population with a mix of randomly generated individuals and individuals containing a constant value for all entries leads to the fastest convergence towards the optimal solution. In a population of 20, seeding ten individuals is found to be optimal. It is also found that the algorithm performance can be further increased by evolving individuals with infeasible solutions instead of iterating them until a feasible solution is found and eliminating the Full Tour sub-problem. These simplifications allow for an increase in the cost budget multiplier, which leads to finding better objective values without further increasing computational time. The best-performing setup, which uses a cost budget multiplier of 10, can find the optimal solution to the test problem in 100% of the runs, on average in 9 iterations, with a computation time of 5.82 seconds per evaluation. The results show that the global optimisation algorithm produces results that closely match known results for GTOC4 consistently and accurately.

INTRODUCTION

With the continued development of space exploration and research missions, the need to predict and plan journeys through space is ever-growing. Without this ability, it would be impossible to plot courses to set targets or to estimate the amounts of fuel that would be needed along the way. Since humans first started to make these kinds of calculations, the resources to do so have developed tremendously. Available computational power is many orders of magnitude larger, and methods to calculate and simulate astrodynamic trajectories have made leaps. Still, further improvement is always essential. Currently, these tools are not only crucial to simulate journeys to set targets, but they have become part of a much more extensive and complex problem space. A subset of these complex problems can be described as combinatorial problems, which will be the focus of this paper.

A combinatorial problem refers to a type of problem in mathematics and computer science which involves making selections, arrangements, or combinations from a finite set of objects or elements according to specific rules or constraints. It consists of counting, organising, and analysing various possibilities or configurations

*Graduate Student, Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands
†Education Fellow + Lecturer, Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands

arising from a given set of objects. This paper focuses on solving complex space routing problems that adhere to the combinatorial problem definition. Specifically, this means that the problem consists of a finite set of available targets of interest and a set of boundary conditions, constraints and optimisation criteria. These include dynamical models, budgets like time and fuel, and the ultimate mission objectives. Examples of these problem instances can be found in the challenges posed by the regularly organised Global Trajectory Optimisation Competitions (GTOCs).[1–3] These competitions pose complex global optimisation problems that competing teams from around the world solve using state-of-the-art methods.

These problems are generally hard to solve because of their scale and complexity. Still, they are realistic and relevant in space mission design, especially in up-and-coming areas of space exploration where the number of relevant targets can be vast, like asteroid research or debris clean-up missions. It is crucial to develop robust frameworks and methods to handle these more extensive and complex problems and solve them to satisfactory optimality. Combinatorial problems have been studied and solved since long before space flight was born,[4–6] and powerful tools exist to solve them. Ideally, these new solving methods build upon or extend the already available tools.

The purpose of this paper is to propose a specific implementation of a global optimisation algorithm which can be used to solve complex space routing problems. The particular implementation is set up to solve the problem posed in GTOC4,[7] where the goal is to find a trajectory which allows one to visit the maximum number of given targets within the fuel and time budgets. The problem is split into sub-problems formulated as mixed-integer linear problems (MILPs) to find the optimal solution. This means the sub-problems can be solved in sequence using the Solving Constraint Integer Programming (SCIP)[8] software, which is known to be very well suited to solve combinatorial problems.[9–11] The Fixed Budget sub-problem selects a subset of the given set of targets. The Full Tour sub-problem orders the targets in the subset, and the Fixed Tour sub-problem optimises the flight time for every leg of the given trajectory to find the solution with the lowest total fuel consumption.

The first section of this paper lays out the general definition of the considered space routing problem and an overview of the proposed methods for solving this problem. The following sections then focus on the three sub-problems used in the solving process: target selection, target ordering and trajectory optimisation. Each section will describe and evaluate the methods used to solve the sub-problem. Then, a global optimisation algorithm is presented, where the three sub-problems are aggregated into a single solver. The algorithm's performance and accuracy are analysed, leading to further improvements to the algorithm's efficiency. Finally, the results are compared to known results from the GTOC4.

PROBLEM DEFINITION

In the most general sense, the problem can be described as: *Given a budget, a set of targets, and some boundary conditions, visit as many of the given targets as possible.* Budgets can be fuel and time or some other relevant quantity that depletes during the mission. Targets can be locations in space or objects moving through space in known or assumed orbits. Examples of sets of targets could be near-Earth asteroids in trajectories around the Sun or pieces of space debris in orbits around Earth. The problem is further defined by the starting conditions, which could include but are not limited to launch-time windows, launch location and launch velocities, and final conditions, such as a required final position or state. Other factors that impact the problem are the environmental and dynamical models used to simulate motions through space and the spacecraft's specifications on, for example, thrust. The term "visit" can also be defined in different ways. Common ways to make visits in space are the fly-by, where only the position vectors of the spacecraft and the target need to match up at some time, and the rendezvous, where both the position vectors and the velocity vectors of the spacecraft and the target need to match. Additionally, for a rendezvous, there can be conditions on the minimum or maximum duration, while fly-bys are, by definition, only a singular moment.

In this paper, the general problem will be further specified to make it a suitable test case for analysing the proposed solving algorithm implementation. The problem definition is inspired by the problem presented in GTOC4 but is implemented with slight alterations to keep the complexity low enough to be practical in the sense of computational time needed to optimise the problem. Still, the similarity to the example problem

from GTOC4 allows for a reasonable comparison of the results. For this research, the following assumptions and definitions regarding the problem settings are used, which are the same as those in GTOC4:

- The budgets consists of a maximum velocity change ($C_{\max}$) in km/s and a maximum total mission duration (τ) in s.
- The starting position is the point-mass position of Earth.
- The targets are objects orbiting the Sun in (near-Earth) Kepler orbits.
- A visit is defined as a fly-by of a target. A fly-by constitutes an exact match of positions. Velocities do not need to match.

The following assumptions and definitions are used in this paper but differ from GTOC4:

- The launch time (t_0) is fixed.
- The mission is open-ended. There are no final conditions.
- The spacecraft can change its velocity instantly during a fly-by. Between fly-bys, the spacecraft follows a Kepler orbit around the Sun.
- The first transfer is "free". At launch time, at launch position, the spacecraft can make an unlimited instant velocity change, which does not count to the budget.

For the assumptions that differ from GTOC4, the motivation is to reduce the complexity of the problem. These differences have a significant impact on the complexity while allowing the creation of problem instances that are comparable to available partial solutions from GTOC4.

These assumptions and definitions define the optimisation problem considered in this paper. The objective is to find the trajectory that visits the maximum number of targets. The secondary objective is to use the lowest possible ΔV. As there is no comparison between different spacecraft or propulsion systems, the magnitude of required ΔV is directly equivalent to the cost in fuel mass. A solution that visits more targets is always better, but between solutions that visit the same number of targets, the one costing less ΔV, thus fuel, is better. From this problem definition, specific problem instances can be created by defining the values for the initial conditions, budgets, target state histories and time step duration. In the remainder of this section, the most relevant methods and rules of thumb used to come up with these values are introduced. Finally, a general outline of the methodology used in the solving algorithm is provided.

The position of Earth and the targets are derived from the initial conditions given in the GTOC4 problem description. From these initial conditions, the orbits are propagated forward through time using the Technical University Delft Astrodynamics Toolbox (TUDAT).[12] All orbits are propagated as pure Kepler orbits, so given the initial Keplerian elements, the orbits are determined and governed by the equations in Equation 1.[7]

$$\ddot{x} = -\mu_S \frac{x}{r^3}, \quad \ddot{y} = -\mu_S \frac{y}{r^3}, \quad \ddot{z} = -\mu_S \frac{z}{r^3}, \quad \text{where} \quad r = \sqrt{x^2 + y^2 + z^2} = \frac{a(1-e^2)}{1 + e\cos(\theta)} \tag{1}$$

$\ddot{x}, \ddot{y}$, and $\ddot{z}$ represent the Cartesian acceleration components caused by the gravitational force of the central body, the Sun in this case. x, y, and z are the Cartesian positional coordinates, μ_S is the gravitational parameter of the Sun. r is the distance from the Sun, which can be expressed both in Cartesian position coordinates or in Keplerian orbital parameters. a is the semi-major axis, e is eccentricity and θ is the true anomaly. To compare the results from this research with the results of the comparable problem in GTOC4, the same values for relevant constants and conversion parameters are used for all computations in this paper. These values can be found in Table 1.

Table 1: Constants and conversion[7]

Parameter	Value
Sun's gravitational parameter u_s, km^3/s^2	$1.32712440018 \times 10^{11}$
Astronomical Unit (AU), km	$1.49597870691 \times 10^{11}$
Standard acceleration due to gravity, g_0, m/s^2	9.80665
Day, s	86400
Year, days	365.25

The GTOC4 problem description also shows that the spacecraft has an engine with a specific impulse, I_{sp}, of 3000s and a maximum thrust of 0.135N. Also, the propellant mass is 1000kg, and the spacecraft dry mass is 500kg. The value for g_0 is given in Table 1. Using these values and Equation 2, it is determined that these conditions are equivalent to a maximum total velocity change, ΔV, of 32.3 km/s. This value will be used as a reference limit for the test cases in this research, as it will not use low thrust but rather instant velocity changes to reduce complexity.

$$\Delta V = I_{sp} g_0 \ln \frac{m_0}{m_f} \tag{2}$$

By defining the problem and the spacecraft specifications as given above, the spacecraft's trajectory is determined by Kepler arcs from fly-by to fly-by. The velocity change needed at a certain fly-by to reach the next fly-by can be computed by solving the trajectory legs using Lambert solutions.[13] The velocity vectors resulting from the Lambert solutions for the incoming and the outgoing transfer legs can conveniently be used to determine the ΔV needed during a fly-by.

Say a trajectory is selected from A to B, to C, where $A, B$and$C$ are in the set of targets. For a given flight time from A to B, the Lambert solution gives the required velocity at A and the resulting velocity at B for this specific trajectory. In the same way, for a given time of flight, ToF, between B and C, the Lambert solution gives a required velocity at B and a resulting velocity at C. The ΔV needed at B is now defined as the vector difference between the resultant velocity from the $A - B$ leg and the required velocity for the $B - C$ leg. Using this method, the ΔV travelling any three-point arc can be computed, given the flight times for the two legs. It can be used to, in a relatively simple and fast way, calculate the total velocity change needed for a longer multi-target trajectory by patching three-point arcs together. Targets must be selected and ordered to find an optimal solution, and the trajectory through these targets must be optimised for a minimal ΔV. Given the targets in the solution, the solution then consists of a set of consecutive, partly overlapping, three-point arcs. So if the first arc is $A - B - C$, which gives the ΔV at B, the second arc has to start with $B - C - ..$ to provide the ΔV at C. The total ΔV for the trajectory is the sum of the ΔV for all individual arcs.

Ideally, the ToF values are taken as variables to be solved for the lowest possible ΔV. However, the total velocity is not a linear function of the ToF variables. So, solving for the ToF variables directly leads to a non-linear problem, which would be considerably more challenging to solve than a linear problem. To use a powerful linear solver, the problem is split into multiple linear sub-problems, which are solved sequentially to create the optimal solution. The sub-problems are defined as follows:

- Fixed Budget: A subset of targets is selected based on the ΔV budget.
- Full Tour: The selected targets are ordered by finding the cheapest trajectory through all targets in the subset.
- Fixed Tour: The flight time for every leg of the trajectory is optimised for a minimal total ΔV.

The following sections will explain the methodology for solving these individual sub-problems. All sub-problems are formulated as MILPs and solved using PySCIPOpt[14] (a Python wrapper for SCIP). They are

individually evaluated* for setup and performance before patching the sub-problems together and assessing the global optimisation algorithm. In the global optimisation algorithm, PyGMO[15] performs parallel batch evaluations, reducing total computational time. Unless stated differently, run-time values are averages from runs executed in 10 parallel threads, both in sub-problem and global optimisation analyses. Ideally, sets of targets are used for the evaluations for which an optimal, or reasonably close to optimal, solution is known. This allows validation of the results produced by the proposed algorithm. Also, these sets are preferably small to reduce computation time in the performance evaluation. For this purpose, the known solutions for GTOC4[16] are used to generate smaller problem instances consisting of subsets of the complete target set provided. For example, when using the launch position and time of a known solution and searching for a solution among the ten targets first visited by this solution, one would reasonably expect to find a solution which is equal to or similar to the known solution. In evaluating the individual steps, targets will be taken from the targets visited in the solution provided by Moscow State University (MSU).[16] Results produced with the global optimisation algorithm will be compared with multiple partial solutions from GTOC4.

TARGET SELECTION

This first step in the solving process aims to select a subset of the targets given in the problem instance. The requirement is that there is a trajectory which flies by all the targets in the subset while not needing more ΔV and time than the budgets allow. The sub-problem aiming to find the trajectory passing the maximum number of targets within this requirement is called the Fixed Budget sub-problem. To allow solving the sub-problem using a MILP-solver, which is considered well suited for solving optimisation problems of this kind,[17] the sub-problem needs to be formulated as a MILP. To achieve this, all ToF parameters ($\text{tof}_{t,i,j}$), defining the flight time between two targets, i and j, are given a constant value instead of solving them for optimality. All time values and durations are expressed as an integer number of time steps, where the duration of a time step in seconds is a constant. The solution produced by solving this sub-problem is a trajectory visiting a subset of all available targets in the problem instance. This subset is used as input in the following sub-problems.

Problem Formulation

The mathematical formulation for the Fixed Budget sub-problem is given in Equations 3-11.

$$\underset{x}{\text{maximise}} \quad \sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\sum_{k=1}^{n_v} x_{t,i,j,k}, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{3}$$

subject to

$$x_{t,i,j,k} \in \{0,1\}, \quad \text{for all} \quad t,i,j,k \tag{4}$$

$$\sum_{k=1}^{n_v}\sum_{i=1}^{n_v} x_{0,i,1,k} = 1, \quad \text{with} \quad i \neq k, i \neq 1, k \neq 1 \tag{5}$$

$$\sum_{t=0}^{n_t}\sum_{j=1}^{n_v}\sum_{k=1}^{n_v} x_{t,i,j,k} \leq 1, \quad \text{for all} \quad i, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{6}$$

$$\sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{k=1}^{n_v} x_{t,i,j,k} \leq 1, \quad \text{for all} \quad j, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{7}$$

$$\sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{j=1}^{n_v} x_{t,i,j,k} \leq 1, \quad \text{for all} \quad k, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{8}$$

*All computations are executed using Python on an HP ZBook Studio G5 laptop with the following hardware: Intel Core i7-8750U CPU (2.2GHz) and 32 GB 2667 MHz DDR4 SODIMM RAM.

$$\sum_{i=1}^{n_v} x_{t_2,i,j,k} - \sum_{i=1}^{n_v} x_{t,k,i,j} = 0, \quad \text{for all} \quad t,i,j, \quad \text{with} \quad i \neq j, i \neq k, j \neq k, j \neq 1, t_2 = t + \text{tof}_{t,i,j} \tag{9}$$

$$\sum_{t=0}^{n_t} \sum_{i=1}^{n_v} \sum_{j=1}^{n_v} \sum_{k=1}^{n_v} \text{tof}_{t,j,k} \cdot x_{t,i,j,k} \leq n_t, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{10}$$

$$\sum_{t=0}^{n_t} \sum_{i=1}^{n_v} \sum_{j=1}^{n_v} \sum_{k=1}^{n_v} c_{t,i,j,k} \cdot x_{t,i,j,k} \leq C_{\max}, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{11}$$

Equation 4 ensures all x variables have binary values. They represent the inclusion of a particular arc in the solution trajectory. In other words, $x_{t,i,j,k}$ represents the statement: the spacecraft flies by target j at time t, it came in from target i and continues to target k. This is either true, represented by value 1, or false, represented by value 0. Note the x variables are the only variables being solved in this problem, meaning that the problem contains only binary variables. This makes the problem a Binary Integer Linear Programming (BILP), a subset of MILPs. i, j, and k are integers that all exist in the same set of targets. n_v is the total number of targets in the problem instance, including the starting position. (The subscript v is derived from the term vertices, commonly used in graph-based problems. It is used here to distinguish clearly from the number of timesteps n_t.) t values are integers in a finite set of time steps. n_t is the total number of time steps, which defines the maximum mission duration.

x variables only exist for the cases where the three targets are unique. There is no path from a target to itself, and returning paths between only two targets, like A-B-A, are not considered. The complete set of x variables represents all possible trajectory arcs at all available times. The objective, given in Equation 3, is to maximise the number of arcs in the trajectory, as this directly corresponds with the number of targets visited. The $c_{t,i,j,k}$ values directly correspond with the amount of ΔV needed to include arc $x_{t,i,j,k}$ in the solution trajectory. These values can all be computed before starting the actual optimisation, given the constant ToF values. This makes evaluations within the optimisation process cheap and fast regarding computational requirements. To compute the ΔV, the input parameters t, i, j, k must be converted into an actual time value and actual positions in space. These are taken from the state histories computed by orbit propagation.

Constraints are used to ensure the continuity of the trajectory and to enforce the budgets. Equation 5 is a constraint to lock in the start position and start time, requiring that target 1 has an outgoing arc at time 0. Equations 6, 7, and 8 are three sets of constraints that ensure a single target never has more than one incoming and one outgoing arc. The value is allowed to be 0, as it is not required for all targets to be visited. As the ToF values are known, these can be used to ensure the continuity of the trajectory. Equation 9 is a constraint that enforces the outgoing arc departs at the exact time the incoming arc arrives. This also ensures that an outgoing arc always matches an incoming arc. The case where $j = 1$, meaning the trajectory departs from the start point, is excepted from this constraint. The simplest way to fully constrain this problem is to require a closing loop. The cost for any transfer to the start point is set to 0 to make the loop practically open-ended. So, a closed loop is solved, but the cost and duration for the final transfer are disregarded. Finally, the budget constraints for total time and cost are given in Equations 10 and 11.

Note that these constraints do not prevent a solution from containing sub-tours. For example, a solution with six targets might be found, but the trajectory consists of two disconnected loops of three targets. In such a case, additional constraints must be added to the problem before solving it again. The constraints that will be added are given in Equation 12. S is a sub-tour, which contains several targets. For every sub-tour in the solution, a constraint is added to restrict the number of active arcs to be less than the number of targets in the sub-tour. This prevents the sub-tour from being part of the solution in the future. These constraints are only added during solving when sub-tours are encountered, as it is not feasible to set the constraint for every possible sub-tour in the problem beforehand, considering the total number of subsets has a magnitude of

2^{N_v}. After the sub-tour constraints are imposed, the problem is solved again. These steps are repeated until a solution is found, which consists of a single continuous loop. With a more significant number of targets, the chance of finding solutions containing sub-tours increases. This might considerably impact the solving time, as the sub-tours must be eliminated iteratively.

$$\sum_{t=0}^{n_t} \sum_{i \in S} \sum_{j \in S} \sum_{k \in S} x_{t,i,j,k} \leq |S| - 1, \quad \text{for all} \quad S, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{12}$$

Given that the solution is found using constant ToF values that are random or guessed, assuming that the resulting trajectory is not necessarily the optimal trajectory through the given targets is reasonable. The solution will be further optimised in the target ordering and trajectory optimisation steps of the global solving algorithm. As it can reasonably be assumed the final trajectory has a lower fuel consumption than the initially found solution, the $C_{\max}$ budget can be increased for this first step, hoping that the optimisation will bring the solution back within the actual budget. The impact of the budget parameter on this individual step will be evaluated, but the impact will be more significant when used in the complete solver. The considerations in choosing this increased initial budget and the impact on the overall solving performance will be discussed later.

The sub-problem is solved using SCIP,[8] wrapped in Python (PySCIPOpt[14]). The core of this software is a general branch and bound algorithm[18] and a set of well-known and proven effective heuristics.[9] It is effective in solving MILP problems like the problem considered in this paper.[10,11] Extensive setup and tuning of the solver are outside this research's scope, but to ensure sufficient performance, some high-level settings are tuned for solving time and efficiency. For the evaluations, all problem instances will be solved to finality. An instance could be infeasible to solve, but if a solution is found, this is guaranteed to be the best possible solution, according to the objective function, within the given constraints. The considered settings will not change the solution found for a specific instance, but they might impact the time it takes to do so.

The result of solving this problem will be a trajectory, represented by x variables with a value of 1. The problem definition guarantees finding a solution with the maximum number of targets within the budget. Still, there is no guarantee that this is the cheapest solution for that number of targets or even for that specific set of targets. For example, if a total set of 10 targets is given, a solution might be found containing five targets. This means there is no possible solution of six targets within the budget, but there might be a different five-target solution with a lower total cost. It is also crucial to note that the solution here highly depends on the initial ToF values used. It is a locally optimal solution within the whole problem space. Choosing the initial values and finding global optima using this local solution is discussed later in this paper.

Evaluation

The target selection algorithm is tested to tune the SCIP settings and to find sensitivity to several problem parameters. As the aim is to design a computationally efficient algorithm, the most relevant performance parameter is run-time. The default values used to define the problem used as a test case can be found in Table 2.

Table 2: Default parameter values

Parameter	**Value**
$C_{\max}$, km/s	12
n_v	10
n_t	100
Time-step, days	5
t_0, MJD	58677

The starting position is Earth's position at the defined t_0. The set of 10 targets consists of the first ten targets of a known GTOC4 solution. Precisely, it consists of the following asteroids: 2006QV89, 2006XP4,

2008EP6, 2007KV2, 2005XN27, 2006TB7, 2008AF4, 2006HF6, 2008PK3, 2007VL3. This fully defines the problem instance, first used to find the best combination of SCIP settings. The settings considered are the "Presolve" and "Heuristics" settings. These settings determine the extent to which the default pre-solving methods and default heuristics are used in SCIP's default solving engine. The available options are: "Default", "Aggressive", "Fast", and "Off" for both Presolve and Heuristics. Note that these settings will not impact the solution to the problem, only how this solution is produced. For this reason, computation time is the only relevant performance parameter in this comparison.

Table 3: Run-times for different combinations of SCIP settings in seconds

		Heuristics			
		Default	**Aggressive**	**Fast**	**Off**
Presolve	**Default**	18.16	25.52	19.01	19.06
	Aggressive	17.65	23.72	19.70	17.63
	Fast	18.67	23.02	17.55	18.10
	Off	15.89	34.44	15.63	15.14

The results for running the available combination of the settings can be found in Table 3. These are the mean run-time values over 500 runs evaluated using multi-threading. The only difference between the runs, in this case, is the constant ToF values, which are randomly generated for every run. The results show that using aggressive Heuristics takes a significant amount of time. It can also be seen that more aggressive Presolving in combination with using Heuristics can be beneficial for solving time. Still, setting both the Presolve and Heuristics settings to Off gives the best performance with a mean time of 15.14s per evaluation. The results suggest that the posed problem is relatively easy for the SCIP solver. Pre-solving and Heuristics can help reach good solutions for complex problems faster, but they only slow the solver down in this case. Unless stated differently, the Presolve and Heuristic settings will be turned off when solving the Fixed Budget problem.

Table 4: Run-time sensitivity on $C_{\max}$, n_v and n_t parameters

Parameter	**Run-time [s]**	**Change [%]**	**N vars**	**Change [%]**
$C_{\max}$ **= 6 km/s**	14.72	-2.8	99990	0
$C_{\max}$ **= 18 km/s**	18.59	+22.8	99990	0
n_v **= 5**	1.86	-87.7	12120	-87.9
n_v **= 15**	59.51	+293.1	339360	+239.4
n_t **= 50**	7.14	-52.8	50490	-49.5
n_t **= 150**	25.69	+69.7	149490	+49.5

The same problem instance is then used to study the sensitivity of the Fixed Budget sub-problem to some of the most critical problem parameters: $C_{\max}$, n_v and n_t. n_v and n_t impact the problem's size directly as the number of variables changes. $C_{\max}$ does not have this same direct impact on the problem's size but does have an impact on the size of the feasible solution domain, which in turn impacts solving time. The comparative results for the runs with parameter changes can be found in Table 4. Along with the change in run-time, it also shows the change's impact on the problem's size.

The small changes in run-time, seen in Table 4, show the sensitivity on $C_{\max}$ is minor. The increase has a more significant impact than the decrease. A possible explanation could be that reducing $C_{\max}$ past some lower limit, which would depend on the specific problem instance, does not make the problem any easier to solve. In the same way, there might be an upper limit, past which increasing $C_{\max}$ does not make the problem more challenging to solve, but this value is expected to be much higher. Table 4 shows significant changes in the number of variables and the computational time for n_v, indicating very high sensitivity. It is known that n_v has a cubic relation to the number of variables. With reducing n_v, the reduction in solving time is relatively equal to the reduction in variables. This indicates the relation between these parameters might be linear to a certain point. It can be seen that this linearity is broken when moving in the other direction,

where the increase in run-time is significantly larger than the increase in variables. This same observation is made with the n_t results. n_t has a linear relation to the number of variables. In reduction, the impact on run-time and variables is nearly the same, but when the number of variables grows larger, the computational time increases disproportionally.

These results reveal a real scalability problem. Considering that realistic real-world problem instances might have larger budgets, longer timelines, and, most significantly, target sets containing hundreds or even thousands of targets, the exponential increase in computational time for increasing any of these is a very significant problem. It is recommended that future research consider improvements to the proposed methods or propose alternate methods for target selection to make solving larger problem instances feasible.

TARGET ORDERING

The second step in the solving process aims to find the optimal order to visit the targets given by solving the Fixed Budget sub-problem. That solution is already a feasible trajectory within the allocated budgets, but it is not guaranteed to be the optimal tour through these targets. It is now required that the resulting trajectory flies by all the given targets. The sub-problem targeting to find the trajectory through all targets with the lowest total velocity change is called the "Full Tour" sub-problem. As for the previous sub-problem, constant values are used for all ToF parameters to allow a MILP formulation. In the global optimisation algorithm, solving the Full Tour sub-problem will follow the solving of the Fixed Budget sub-problem. The instance to be solved is the same, meaning that all $c_{t,i,j,k}$ and $\text{tof}_{t,j,k}$ values are the same. This guarantees that a feasible solution to the Fixed Budget sub-problem is also a feasible solution to the Full Tour sub-problem.

Problem Formulation

The formulation of the Full Tour sub-problem is very similar to that of the Fixed Budget sub-problem but has some alterations. The formulation can be found in Equations 13-20. All symbols have the exact definition as in the previous section. Again, the problem only solves for the x variables, which makes the formulated problem a BILP.

$$\underset{x}{\text{minimise}} \quad \sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\sum_{k=1}^{n_v} c_{t,i,j,k}\cdot x_{t,i,j,k}, \quad \text{with} \quad i\neq j, i\neq k, j\neq k \tag{13}$$

subject to

$$x_{t,i,j,k}\in\{0,1\}, \quad \text{for all} \quad t,i,j,k \tag{14}$$

$$\sum_{k=1}^{n_v}\sum_{i=1}^{n_v} x_{0,i,1,k}=1, \quad \text{with} \quad i\neq k, i\neq 1, k\neq 1 \tag{15}$$

$$\sum_{t=0}^{n_t}\sum_{j=1}^{n_v}\sum_{k=1}^{n_v} x_{t,i,j,k}=1, \quad \text{for all} \quad i, \quad \text{with} \quad i\neq j, i\neq k, j\neq k \tag{16}$$

$$\sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{k=1}^{n_v} x_{t,i,j,k}=1, \quad \text{for all} \quad j, \quad \text{with} \quad i\neq j, i\neq k, j\neq k \tag{17}$$

$$\sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{j=1}^{n_v} x_{t,i,j,k}=1, \quad \text{for all} \quad k, \quad \text{with} \quad i\neq j, i\neq k, j\neq k \tag{18}$$

$$\begin{aligned}\sum_{i=1}^{n_v} x_{t_2,i,j,k}-\sum_{i=1}^{n_v} x_{t,k,i,j}=0, \quad &\text{for all} \quad t,i,j,\\ &\text{with} \quad i\neq j, i\neq k, j\neq k, j\neq 1,\\ &t_2=t+\text{tof}_{t,i,j}\end{aligned} \tag{19}$$

$$\sum_{t=0}^{n_t}\sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\sum_{k=1}^{n_v} \text{tof}_{t,j,k} \cdot x_{t,i,j,k} \leq n_t, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{20}$$

There is no constraint on ΔV cost. Instead, as given in Equation 13, the objective is to minimise the total ΔV of all included trajectory arcs. The sets of constraints in Equations 16, 17, and 18 are similar to the ones used previously, but now the number of incoming and outgoing legs for every target is enforced to be exactly one, as it is required that every target is visited. The other constraints are precisely the same as in the Fixed Budget sub-problem. Finding solutions with disjoint sub-tours is again handled by adding additional constraints as in Equation 12.

The solution to the formulated problem is a trajectory defined by the x variables with a value of 1. If an optimal solution is found, this is guaranteed to be the cheapest trajectory to visit all the targets in the set, given the constant ToF values. This can be the exact solution found in the Fixed Budget sub-problem or a different, cheaper solution. Again, this is only a locally optimal solution, which strongly depends on the ToF values given. Cheaper trajectories likely exist when different ToF values are considered.

Evaluation

To optimise the solving performance for this sub-problem, the SCIP settings are re-evaluated. The test case is very similar to the one used in the previous section. The performance of this sub-problem will be tested in combination with the Fixed Budget sub-problem. The Fixed Budget and Full Tour sub-problems will be solved in sequence for every run. The solution generated with the Fixed Budget is used as input for the Full Tour sub-problem. This means the number of targets in the problem to be solved depends on the result of the first sub-problem and is not a constant. To make meaningful comparisons, comparing runs with the same problem size is essential.

Table 5: Default parameter values

Parameter	**Value**
$C_{\max}$, km/s	50
n_v	10
n_t	100
Time-step, days	5
t_0, MJD	58677

The specific problem instance used for analysis is chosen so that solving the Fixed Budget sub-problem will likely result in a five-target solution. The problem instance's parameters can be found in Table 5. The starting position and asteroids in the initial target set are the same as in the previous section.

Table 6: Run-times for different combinations of SCIP settings in seconds

		Heuristics			
		Default	**Aggressive**	**Fast**	**Off**
Presolve	**Default**	5.48	5.00	5.79	4.83
	Aggressive	6.01	6.11	6.70	5.78
	Fast	4.44	5.43	4.80	4.44
	Off	0.76	5.23	0.61	0.17

The same options are available for both the Presolve and Heuristics settings and the results for average run-times on iterations can be found in Table 6. These are the mean run-time values over 500 runs evaluated using multi-threading. These results show a massive impact of the Presolving setting on the run-time. Setting

it to "Off" makes solving an order of magnitude faster in all cases except for Aggressive Heuristics. The relative impact of the Heuristic setting is much smaller, but still, the quickest solving times are found when turning the Heuristics off completely.

Table 7: Solution improvement proportions

		Heuristics			
		Default	**Aggressive**	**Fast**	**Off**
Presolve	**Default**	16.25%	9.5%	19.25%	17.25%
	Aggressive	16.5%	14.5%	17.25%	17.25%
	Fast	16.5%	11.25%	18.25%	15.25%
	Off	2.75%	2.75%	5.5%	4.5%

For this ordering step to be practical in the global optimisation algorithm, the time it takes to get a solution should be in line with the probability of finding a different, better solution than the Fixed Budget solution. The proportion of improved solutions is computed for the different SCIP settings to assess this effectiveness. The results are found in Table 7. This shows that although the Presolving setting considerably increases the solving time, it also significantly increases the chance of finding a better solution. In this isolated scenario, it is impossible to make the trade-off between computational time and the chance of finding better solutions, as only their impact on the global optimisation algorithm is relevant.

This second step takes much less time in the global solving process than the previous target selection step. This is primarily because the problem instance considered in this step is much smaller. The Fixed Budget problem considers all targets in the given problem, while the Full Tour problem only considers the targets that are part of the previous solution. But also, when comparing problems of similar size, the second step seems more straightforward to solve. A Fixed Budget problem instance with five targets and 100 time steps on average takes 1.86s to solve (see Table 4), while the Full Tour problem with the same values only takes 0.174 seconds on average. This is likely a result of the different constraints for both problems. The Fixed Budget problem has multiple sets of inequality constraints (Equations 6, 7 and 8), allowing tours not to visit every target. These are replaced with equality constraints (Equations 16, 17 and 18) in the Full Tour problem, where it is required to visit all targets. The equality constraints leave a much smaller feasible domain to be searched.

TRAJECTORY OPTIMISATION

The last step in the solving process aims to optimise the ToF values for the trajectory produced by solving the Full Tour sub-problem. This trajectory is guaranteed to be the cheapest order to visit these targets, given the constant ToF values that were initially assumed. Now, the targets and their order are considered fixed, and the objective is to find the ToF value for every leg of the trajectory, which results in the lowest possible total cost for the entire trajectory. This sub-problem is called the "Fixed Tour" sub-problem. As the ToF values will now be variable, the problem formulation and solving process substantially differ from those in the previous steps.

Problem Formulation

Even though the implementation for this sub-problem is significantly different, the idea behind the formulation is the same as for the Fixed Budget and Full Tour problems. There is a set of binary x variables that represent possible trajectory arcs that are selected to either be part of the solution trajectory or not. These are the only variables to solve, making this sub-problem a BILP. This formulation differs from the previous sub-problems in multiple ways. The Fixed Tour optimisation problem formulation is given in Equations 21-24.

The input to this problem is a complete trajectory (T), of which the exact order will be followed. T consists of a set of consecutive arcs (a). One arc represents three targets previously shown as i, j, k. Every arc a then consists of two legs, one inbound (a_1, e.g. from i to j) and one outbound (a_2, e.g. from j to k) and each

$$\underset{x}{\text{minimise}} \quad \sum_{a \in T} \sum_{t \in tp_a} \sum_{\text{tof}_1 \in \text{tof}_{a_1}} \sum_{\text{tof}_2 \in \text{tof}_{a_2}} c_{t,a,\text{tof}_1,\text{tof}_2} \cdot x_{t,a,\text{tof}_1,\text{tof}_2} \tag{21}$$

subject to

$$x_{t,a,\text{tof}_1,\text{tof}_2} \in \{0,1\}, \quad \text{for all} \quad t, a, \text{tof}_1, \text{tof}_2 \tag{22}$$

$$\sum_{t \in tp_a} \sum_{\text{tof}_1 \in \text{tof}_{a_1}} \sum_{\text{tof}_2 \in \text{tof}_{a_2}} x_{t,a,\text{tof}_1,\text{tof}_2} = 1, \quad \text{for all} \quad a \in T \tag{23}$$

$$\begin{aligned} \sum_{\text{tof1}_A \in \text{tof}_{a_{1_A}}} x_{t_A,a_A,\text{tof}_{1_A},\text{tof}_{2_A}} - \sum_{\text{tof}_{2_B} \in \text{tof}_{a_{2_B}}} x_{t_B,a_B,\text{tof}_{1_B},\text{tof}_{2_B}} = 0, \\ \text{for all} \quad a_A \in T, a_B \in T, t_A \in tp_{a_A}, \text{tof}_{2_A} \in \text{tof}_{a_{2A}}, \\ \text{with} \quad a_{2A} = a_{1B}, \text{tof}_{1_B} = \text{tof}_{2_A}, t_B = t_A + \text{tof}_{2_A} \end{aligned} \tag{24}$$

leg has a ToF (tof$_1$, tof$_2$). To get the cheapest possible solution, optimal values for tof$_1$ and tof$_2$ need to be found. However, these optimal values cannot be searched for directly by setting all ToFs as variables, as this would break the problem's linearity. To work around this issue, the ToF values are varied over a small discrete number of available values. tof$_{a_1}$ and tof$_{a_2}$ represent the sets of available ToF values for the inbound and outbound legs of arc a. The binary decision variables x now represent the statement: the spacecraft travels the three-point arc a, passing the middle-point at time t, with a flight time of tof$_1$ for the inbound leg, and a flight time of tof$_2$ for the outbound leg. Again, for every x, the ΔV required to include that specific arc in the solution is computed. This cost is represented by the $c_{t,a,\text{tof}_1,\text{tof}_2}$ values. The set of possible timestamps at which a target can be visited is called tp_a, where the target considered is the middle point of three-point arc a.

The objective, Equation 21, is to minimise the sum of the ΔV of all active arcs in the solution. Equation 23 ensures that every arc of the Full Tour solution is included exactly once, and Equation 24 ensures the continuity of the resulting trajectory. A and B represent consecutive arcs of the trajectory. The trajectory arcs and their order are already determined. Still, the continuity constraint enforces that the arrival and departure times match at every target and match the chosen flight times between the targets.

With the discrete set of allowed ToF values for every trajectory leg, timestamps can be determined at which every target could be visited. The values in this set and the size of the set depend on the possible ToF values and the number of legs travelled before reaching the target. For example, one could consider three possible flight times for every leg of the trajectory: the constant value given in the previous step, a ToF of 1 timestep shorter, and a ToF of 1 timestep longer. In this case, as the starting point is fixed at $t = 0$, the trajectory's first target can be visited at three different times, corresponding with the three possible flight times for the leg from the start point to the first target. So, the leg from the first to the second target has three possible start times and three possible flight times. As the start times and the flight times are three consecutive numbers of timesteps, the second target could be visited at five different times. This way, all possible times for visiting all targets along the trajectory can be expanded, producing the tp_a sets.

As the expansion of all possible trajectory arcs can quickly lead to an excessive number of variables, the ranges in which the ToF values are considered need to be reasonably small. The result is a trajectory with optimised values within their small ranges. Still, there is a high probability that the actual optimal value could be outside the initial considered range. To reach the actual optimal flight times, the solving is done iteratively. After an iteration, the ToF values in the solution are checked, and if a value is at the bound of the range, the range is shifted. If the value is found at the lower bound, it is taken as the new upper bound, expanding the range to lower values for the next iteration. When a solution is found for which none of the bounds are required to be shifted, the iterations stop, and the solution is considered the optimal solution.

Given the formulation of this sub-problem and the fact that the ToF value that is part of the previous solution will always remain available in the range for the next iteration, the solution for every next iteration

is guaranteed to be equal to or better than the previous iteration. Where better means the total ΔV of the trajectory is lower. This means that after this last step, it is very likely that an optimised solution is found that has a lower cost than the initial solution found for the Fixed Budget sub-problem. This is why the cost limit in that first step can be increased relative to the actual mission limit in a global optimisation context. In that case, the real limit would need to be checked after solving this final sub-problem. An optimal solution is found if the cost is below the actual limit.

The solution found by solving the Fixed Tour sub-problem is guaranteed to be the cheapest trajectory through the given targets in the given order. It is not dependent on the initial ToF values that are used. This is still a locally optimal solution in the global optimisation context, as the targets and their order were produced by solving the previous sub-problems that are highly dependent on the initial ToF values.

Evaluation

The problem is solved in an iterative manner, where with every iteration, the ToF ranges for the legs of the trajectory are shifted if necessary to finally find the solution with the lowest possible cost for this specific trajectory. The performance of this sub-problem is evaluated in combination with the other sub-problems. This means the Fixed Budget sub-problem is solved to provide a subset of targets, and the Full Tour sub-problem is solved to get the optimal order, which is used as input to solve the Fixed Tour sub-problem finally. The Fixed Budget problem is initiated every run with randomly generated ToF values. The problem instance considered is the same as the one used to evaluate the Full Tour sub-problem. Relevant problem parameters can be found in Table 5. To compare the different SCIP settings only runs with five target solutions are considered.

Table 8: Total run-times for different combinations of SCIP settings in ms

		Heuristics			
		Default	**Aggressive**	**Fast**	**Off**
Presolve	**Default**	115.3	436.9	124.4	148.7
	Aggressive	126.5	385.9	143.7	127.2
	Fast	101.7	470.3	113.6	136.8
	Off	60.4	401.4	50.9	32.4

The same options are available for both the Presolve and Heuristics settings. Table 8 shows the average results for different setting combinations over 500 runs, evaluated using multi-threading. Some of the same general trends as in the previous sections are seen. Setting Aggressive Heuristics significantly increases the solving time, while Aggressive Presolving can be quicker than Default and Fast. Still, turning both settings to Off gives the fastest solving time by a large margin. As for the Fixed Budget problem, the SCIP settings are not considered to impact the quality of the solution, only the time it takes to reach it. For this reason, the setting giving the quickest solving time is considered the best.

Notably, the time needed to solve this sub-problem is much shorter than the other sub-problems, especially compared to the Fixed Budget sub-problem. For all settings, the average solving time is only a fraction of a second, while the found impact on the solution is tremendous. On average, optimising the flight times reduces the trajectory cost resulting from the previous sub-problem by more than 50%, with cases where the reduction is up to 99%. This shows this sub-problem is a critical part of the solving algorithm, and the high efficiency observed in solving it is very positive.

The Fixed Tour solver's performance depends on the time per single evaluation and the number of iterations needed to reach the optimal solution. The number of iterations is not expected to rely on the SCIP settings but rather on the initial ToF values and the width of the ToF ranges used. This is confirmed by comparing the number of iterations and the solving time per iteration between different settings. For this test problem instance, the average amount of iterations varies between a minimum of 7.45 and a maximum of 8.44 for different setting combinations, with standard deviations of 3.1 and 3.8, respectively. This difference is considered insignificant, and the variation is thus independent of SCIP settings.

Table 9: Average number of iterations for varying n_v and ToF range widths

		ToF range			
		2	**4**	**6**	**8**
n_v	**4**	10.38	6.68	4.83	5.00
	5	9.98	6.17	4.90	4.17
	6	8.60	5.26	4.65	3.44

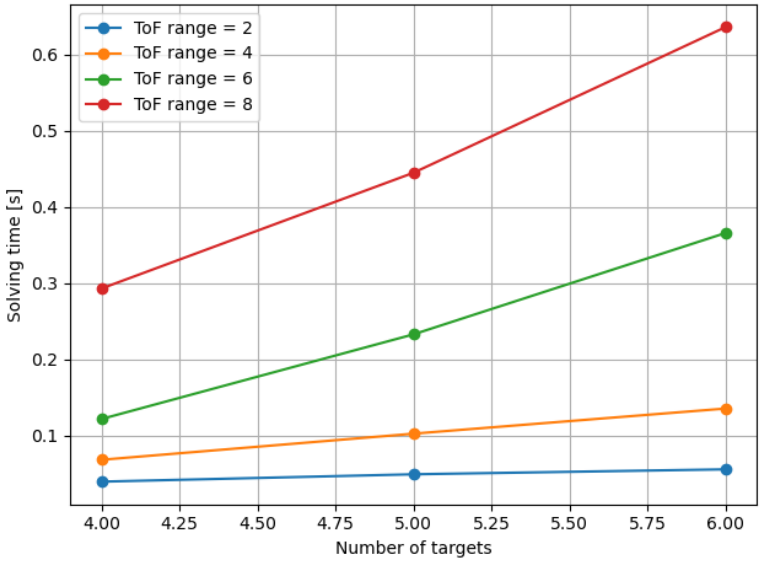

Figure 1: A plot showing the time needed to solve problems with varying amounts of targets. A comparison is made between different ToF range widths.

A ToF range of 2 means that the values from ToF-1 until ToF+1, where ToF is the constant ToF value given at the start of the Fixed Budget problem, are considered in the initial iteration of solving the Fixed Tour problem. So, three possible ToFs are considered for every trajectory leg: [ToF-1, ToF, ToF+1]. A ToF range of 4 extends this to: [ToF-2, ToF-1, ToF, ToF+1, ToF+2], etc.

Reasonably, a wider ToF range reduces the number of iterations necessary to find the optimal solution. Still, it also increases the complexity of the problem to be solved every iteration. Table 9 shows the average number of iterations needed to reach the optimal solution for different ToF range widths. Although the reduction is apparent, it is not 1:1 with the range increase. Doubling the range leads to roughly 1/3 reduction of iterations. Figure 1 shows the total solving times corresponding to different ToF range widths. Total solving time means the time needed to solve all the necessary iterations to reach the optimal solution. The plot shows that the effect of the increased complexity outweighs the reduced number of iterations. The solving time increases significantly with the width of the ToF range. So, the smallest possible ToF range of 2 will be used going forward.

GLOBAL OPTIMISATION

The Fixed Budget, Full Tour and Fixed Tour sub-problems form the core of a global optimisation algorithm. Given a set of initial constant ToF values, solving these sub-problems sequentially gives a locally optimal solution to a specific problem instance. The dependence on the initial ToF values makes the solution locally optimal. The global optimum will only be found if specific initial ToF values are used. No simple methods or heuristics are known to compute or guess these initial values, so they must be searched using a global

optimisation algorithm. PyGMO's "Extended Ant Colony Optimization algorithm", GACO, which allows for parallel batch computations, is used as an optimisation engine. The objective is to find a solution which visits the highest number of targets, and for solutions with the same number of targets, a solution with a lower ΔV cost is considered better. As the algorithm works with a single objective, the objective to be maximised is defined in Equation 25. The cost will always be smaller than 100 km/s, so this definition guarantees that a solution with a higher number of targets always has a better objective value while preferring a lower cost for the same number of targets.

$$\text{obj} = n_{\text{targets}} * 100 - \Delta V \tag{25}$$

Table 10: Default parameter values

Parameter	Value
$C_{\max}$, km/s	15
n_v	6
n_t	100
Time-step, days	5
t_0, MJD	58677
Population size	20
Iterations	50
$C_{\max}$ multiplier	4

Table 11: Known optimal tour for six targets

Target	ToF [days]	Cost [km/s]
Start: Earth	65	-
2006QV89	70	0.261
2006XP4	65	0.667
2008EP6	105	0.431
2007KV2	115	0.605
2005XN27	30	1.726
2006TB7	-	-
Total	450	3.690

To analyse and validate the global optimisation algorithm, a problem instance is created which is as similar as possible to a known solution of the GTOC4.[16] This instance is made with the targets visited in the solution by MSU, which won the competition. If an instance with six targets is mentioned, the first six targets visited by the MSU solution are part of the problem instance. Other default problem parameters are found in Table 10. This relatively small problem instance allows the production of results in reasonably short computation times. t_0 is when the first transfer departs from Earth's position. Its value is derived from the MSU solution to match the time of their departure from Earth.

The population size and the number of iterations are parameters for the GACO algorithm. The $C_{\max}$ multiplier is a factor with which $C_{\max}$ is multiplied before solving the Fixed Budget sub-problem for the first time. As the reasonable expectation is for the cost of the solution to be significantly improved when solving the Fixed Tour sub-problem later, this can still lead to a feasible solution for the global problem. An iteration is made if the Fixed Tour sub-problem returns a solution with a cost higher than $C_{\max}$. First, the ToF values for the trajectory legs part of the Fixed Tour solution are updated to the optimal values found. Then, all three sub-problems are solved again, but $C_{\max}$ for the Fixed Budget problem is set to the cost of the previous solution of the Fixed Tour problem. This new iteration will either return a trajectory with the same number of targets and a lower total cost or a solution with a lower cost and a lower number of targets. These iterations are repeated until a solution is returned with a total ΔV cost below the global $C_{\max}$ budget. Ideally, the number of iterations is as low as possible to reduce solving time while setting the $C_{\max}$ multiplier as high as possible to increase the feasible space that is initially being searched.

So, the global optimisation algorithm evolves a population of individuals represented by a set of initial ToF values. In every iteration, the population is evolved and evaluated. The evaluation of an individual consists of solving the three sub-problems sequentially until a feasible solution is found. With Equation 25, the fitness value corresponding with this individual's solution is calculated. The higher the fitness, the better the individual. In every iteration, the fitness of the whole population is evaluated, and the best-known values are updated before starting the next iteration.

Using this global optimisation algorithm, an optimal solution is found with the problem instance based on the values in Table 10 and the first six targets from the MSU solution. Table 11 shows the trajectory, with the order in which targets are visited, the flight times for the trajectory legs, and the ΔV cost of the manoeuvres. Figure 2 visually compares the results. This shows that the algorithm's result closely corresponds with the

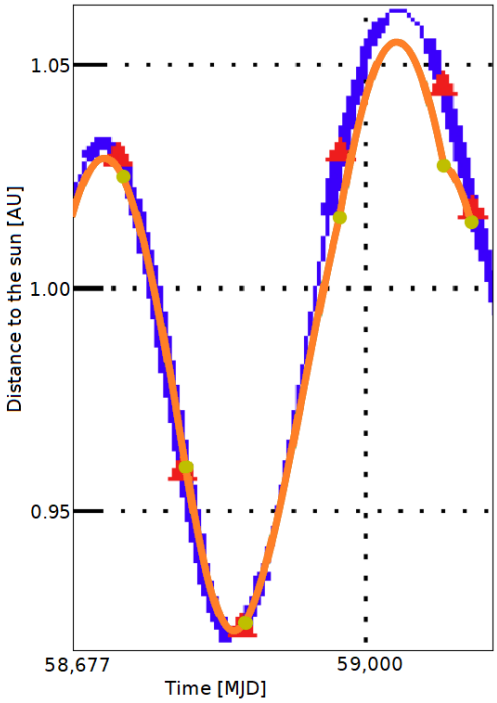

Figure 2: Comparison between the known and generated solutions. The markers represent the fly-by of a target. The blue line and red markers show the GTOC4 result by MSU.[16] The orange line with yellow markers shows the result produced with the global optimisation algorithm.

known solution from GTOC4. Even with the differences in propulsion models and the algorithm using a fixed timestep of 5 days, the order in which targets are visited is exactly the same, and the timing and location of the visits are close to identical. This resemblance confirms the validity of the solving method. It shows the default parameters to be well-suited as base values for performance analysis, as the known optimal solution can be found with this setup.

Table 12 summarises the results from multiple comparative solving runs. Different setups and parameters for the global solving algorithm are compared by solving the problem instance with six targets five times. All values in the table are averaged over the five runs for that setup. In every column, the value considered best is marked green. Extensive tuning of the global optimisation algorithm is outside the scope of this research. Only the newly developed algorithm's integral and critical parameters and settings are analysed. The base setup is the setup shown in Table 10, used to produce the results found in Table 11 and Figure 2. First, the impact of the $C_{\max}$ multiplier and the SCIP settings in the Full Tour sub-problem are analysed.

The results show the expected impact of changing the $C_{\max}$ multiplier. Reducing it decreases the number of iterations within evaluations, as a smaller improvement of the solution between the Fixed Budget and Fixed Tour sub-problems is needed to produce a globally feasible solution. This results in shorter solving times, a reduced objective value for the average found solution and a reduced probability of finding the optimal solution. Increasing the $C_{\max}$ multiplier shows opposite results: Higher iterations per evaluation, higher solving time, better average solutions and a higher probability of finding the optimal solution.

An interesting observation is that the solving time is primarily dominated by the Fixed Budget solving step in all cases, except where the computational effort in the Full Tour step is explicitly increased by turning the use of Presolving and Heuristics on. "Aggressive ordering" corresponds with setting both Presolve and Heuristic settings to Fast when solving the Full Tour sub-problem. Using this setting significantly increases solving time, while the other parameters are only slightly better than for the base setup. "No ordering"

Table 12: Performance statistics for different solver setups. All values are the average of 5 runs performed at the same setting. 1: Best objective, 2: Iterations until best objective, 3: Average solving iterations per evaluation, 4: Average targets in evaluation, 5: Average solving time per evaluation [s], 6: Percentage solving time Fixed Budget, 7: Percentage solving time Full Tour, 8: Percentage solving time Fixed Tour.

Setting	1	2	3	4	5	6	7	8
Base	637.0	20.2	1.45	4.49	4.25	94.9	4.5	0.6
$C_{\max}$ 2x	597.7	27.2	1.09	3.99	3.97	97.5	2.0	0.6
$C_{\max}$ 10x	676.7	25.0	1.83	4.89	4.76	89.2	10.0	0.8
Aggr. ordering	676.7	24.2	1.39	4.60	7.52	51.5	48.1	0.3
No ordering	656.8	37.8	1.41	4.59	4.42	99.3	0.0	0.7
$N_{\text{seeds}} = 5$	656.7	14.2	1.47	4.64	5.06	93.9	5.5	0.6
$N_{\text{seeds}} = 10$	696.3	19.8	1.40	4.96	5.44	92.2	7.2	0.6
$N_{\text{seeds}} = 20$	657.0	22.0	1.48	4.71	5.00	93.7	5.7	0.6
No iterations:								
$C_{\max}$ 2x	696.3	13.6	1.00	4.69	6.23	95.9	3.7	0.4
$C_{\max}$ 4x	696.3	25.6	1.00	5.25	6.31	92.6	6.8	0.5
$C_{\max}$ 10x	696.3	19.8	1.00	5.61	6.35	86.7	12.5	0.9
$C_{\max}$ 10x, no ordering	696.3	9.0	1.00	5.75	5.82	99.0	0.0	1.0
$C_{\max}$ 15x, no ordering	696.3	15.6	1.00	5.84	5.07	98.9	0.0	1.1

skips solving this sub-problem completely. The result from the Fixed Budget problem is directly input into the Fixed Tour problem. In Table 12, the "No ordering" case shows performance parameters similar to the Aggressive ordering case but with a much shorter solving time.

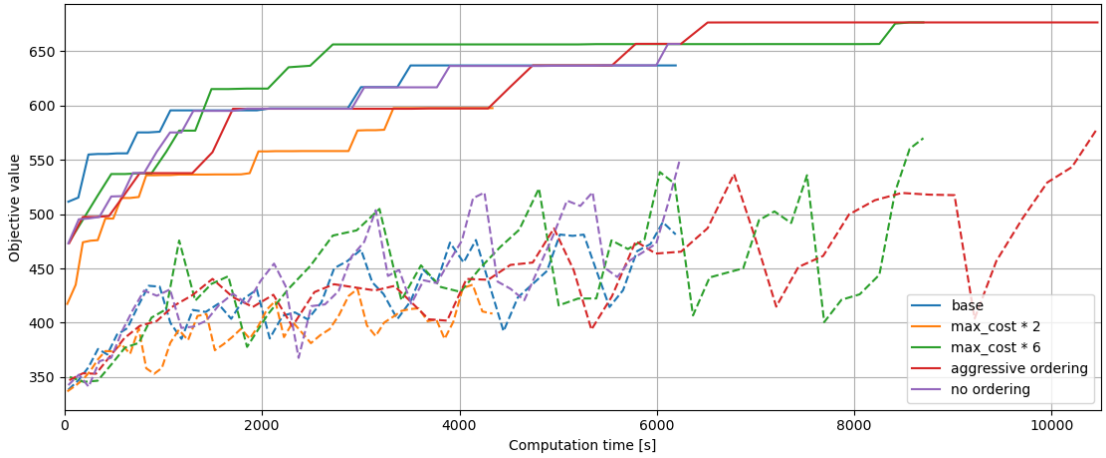

Figure 3: Optimisation performance for different $C_{\max}$ multipliers and ordering settings per total computational time. Solid lines plot the best-known objective in the population. Dashed lines plot the average objective of the population.

Figure 3 visually represents the solving results. It shows the progression of the best-known objective value (continuous lines) and the population's average objective value (dotted lines) over time during solving. Most of the cases show significant variances in the average values. The behaviour of the GACO algorithm can explain this. The algorithm aims to find a better solution by converging towards the current best-known solution. This process is reset to avoid getting stuck near local optima if a better solution is not found in a predefined number of iterations. This resetting would explain the sharp reduction in the average objective value of the population and the following increase as the population again converges. As the settings that are

compared influence the solving time per iteration, the visual comparison is made over total computational time instead of the number of iterations. So, while all runs have the same number of iterations, it can be seen that a setting with a short solving time, like with $C_{\max}$ multiplier $= 2$, terminates much earlier than other settings. The computational time in the plot is computed as the sum of the run-time for all evaluations. In reality, ten evaluations run in parallel, but this does not impact the relative performance between the cases.

The plot shows that increasing the $C_{\max}$ multiplier benefits the algorithm's performance. It helps find better solutions more quickly than any other setting. It also shows that the significant increase in solving time for the "Aggressive ordering" makes that setting relatively slow to find better solutions. It does find good solutions eventually, but the computational time it takes is a considerable disadvantage. Especially considering this is a simple testing problem, when scaling the instance size, this run time difference is expected to increase. Using "No ordering" shows very similar results to the base setup. It does not offer a clear benefit over the base setting, but this also means there is no clear benefit of using the Full Tour sub-problem in its base setting compared to not using it at all.

Another possible improvement to the global optimisation algorithm is considered in seeding the initial population. The algorithm searches for the best initial ToF values to produce the best possible solution. By default, the initial population consists of randomly generated individuals. If there is a different way to initialise the individuals, resulting in better initial objective values, this is very beneficial for the algorithm's performance. There is no simple way to guess the best initial ToF values accurately. Instead, a straightforward policy is considered: a seeded individual is initialised with a constant ToF value for all (t, i, j). A set of these individuals could make a good base for quickly evolving a population to an optimal solution. The constant values that will be used depend on the number of individuals seeded and the number of timesteps in the problem instance. The value for the first seeded individual is taken as $n_t/(2 * n_{\text{seeds}})$, and every next individual will have the value of the previous plus the first value. So an instance with $n_t = 100$ and $n_{\text{seeds}} = 5$ will use values $10, 20, 30, 40$ and 50.

Table 12 shows the results for three different cases. The case with ten seeded individuals in a total population shows the most promise. It shows the highest probability of finding the optimal solution; it was found in all of the five runs. Also, the iterations per evaluation are low, while the average solution has a high number of targets. This would suggest that the average quality of the individuals in the population is the highest. In Table 12, all cases with seeded individuals show longer solving times than the base setup. This can likely be attributed to the average solution's higher number of targets.

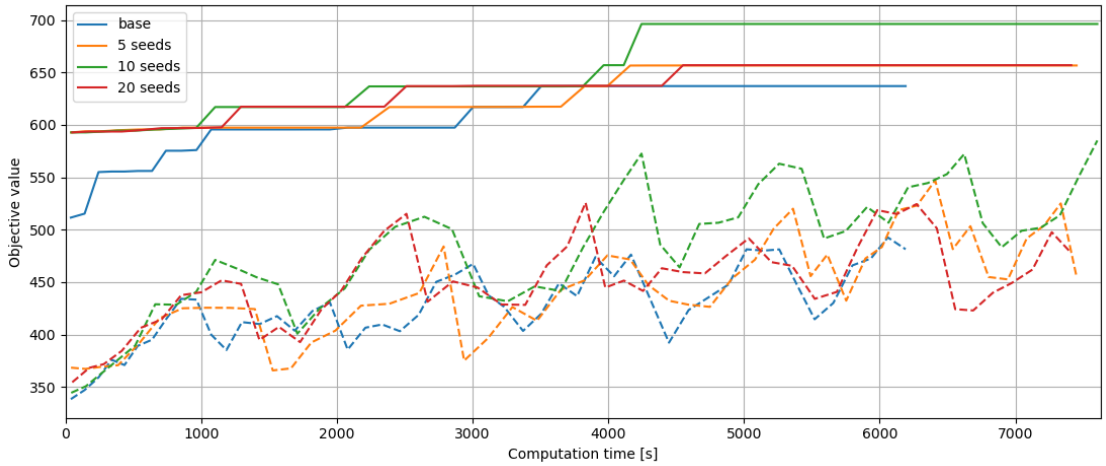

Figure 4: Optimisation performance for different numbers of seeded individuals in the initial population. Solid lines plot the best-known objective in the population. Dashed lines plot the average objective of the population.

A visual representation is given in Figure 4. This plot shows that seeding considerably affects the initial best objective value in all cases. This supports the theory that using seeds with a constant ToF value helps to produce high-value individuals in the initial population reliably. The population averages are not immediately much higher but converge to higher values more quickly, especially for the case with ten seeded individuals. With the best performance seen for the case with 10/20 seeded individuals, it is concluded that a population that consists of 50% seeded individuals and 50% random individuals is optimal. This is the setup that will be used going forward.

A final possible improvement is considered. It has been shown that increasing the $C_{\max}$ multiplier positively impacts the algorithm's performance. However, it also results in an increase in iterations during the evaluation of individuals. Increasing the multiplier increases the chance of finding infeasible solutions, which are always iterated. However, the data shows that these iterations are not at all effective. In all analyses until now, 40,000 individuals have been evaluated. Of the 15,303 individuals that needed multiple iterations to be solved, only five resulted in the global optimal solution. Meanwhile, the global optimum was found for 1,306 of the 24,697 individuals solved in a single iteration. This shows that the chance of finding the optimal solution with an individual that is infeasible after the first iteration is close to zero. So, any effort put into the following iterations is wasted. As an alternative to making these iterations, a case is analysed where individuals that produce an infeasible solution on the first iteration are given a penalty objective value of $-2 * C_{\max}$.

This should allow to raise the $C_{\max}$ multiplier and increase performance while reducing the compromising increase in solving time. As seen in Table 12, increasing the multiplier increases the average number of targets in the solutions. This, in turn, leads to increased solving time, most notably in the Full Tour sub-problem, which can be seen to only require 2% of the computational time with a $C_{\max}$ multiplier of 2, while it takes up 10% of the solving time when the $C_{\max}$ multiplier is 10. So, with an increasing multiplier, removing the Full Tour sub-problem from the algorithm might be increasingly beneficial. Table 12 shows that the results for all cases not using internal iterations are auspicious. All five runs per setup result in the global optimal solution in less than 50 iterations. As expected, increasing the $C_{\max}$ multiplier leads to an increase in average targets in the solutions and an increase in the proportional solving time needed to solve the Full Tour sub-problem. Disabling that sub-problem shows excellent benefit in this case. In Table 12, it is seen that the average solution quality rises even further while solving times are significantly reduced.

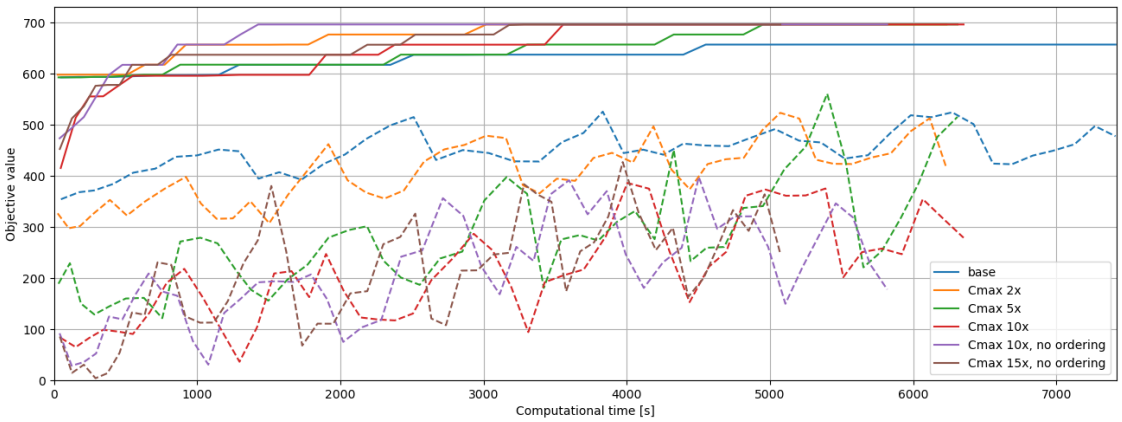

Figure 5: Optimisation performance for different setups not using internal iterations. Solid lines plot the best-known objective in the population. Dashed lines plot the average objective of the population.

These results are also visualised in Figure 5. The impact of penalising the infeasible individuals instead of iterating them till feasible is seen in the much lower population averages. It also shows that the best-known objectives for the high $C_{\max}$ multiplier setups are low, but these very quickly improve and surpass the best solutions of the lower $C_{\max}$ multiplier setups. It is interesting to see that increasing the $C_{\max}$ multiplier from

10 to 15, when not using ordering, significantly reduces computational time while keeping good performance considering the best objective value. Still, the 10x setting reaches the best objective value much sooner regarding time and iterations. From the data, it is concluded that not using internal iterations and not using the Full Tour sub-problem, in combination with using a high $C_{\max}$ multiplier, maximises the algorithm's performance.

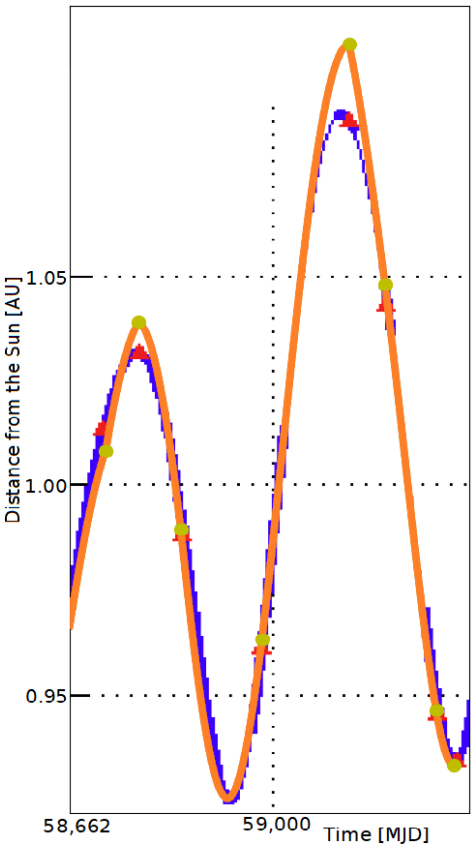

Figure 6: Comparison between the first 8 targets of the GTOC4 solution from The Aerospace Corporation,[16] and a solution created with the proposed algorithm.

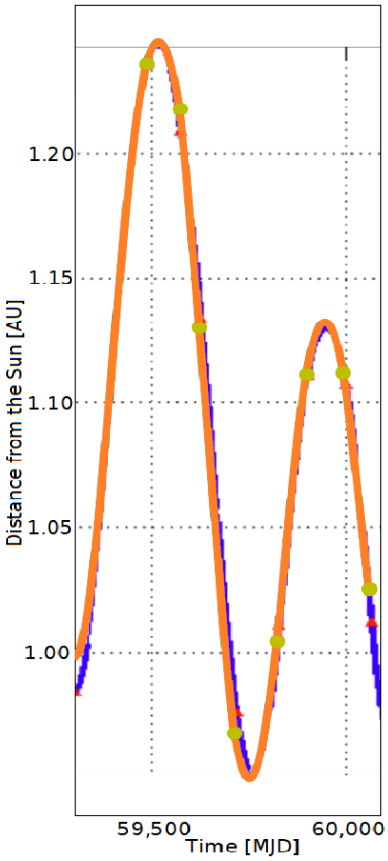

Figure 7: Comparison between the second 8 targets of the GTOC4 solution from The Aerospace Corporation,[16] and a solution created with the proposed algorithm.

To show the performance and accuracy of the improved global optimisation algorithm, four different test cases are considered. These are four different problem instances designed to correspond with known GTOC4 results directly. Figures 6 - 9 show the comparative results for all cases. The orange lines show the solutions produced with the algorithm, and the blue lines show the known solutions from Reference 16. All plots show the distance from the Sun over time, and the markers represent target fly-bys. This form of visual comparison is used because this is the only form in which the GTOC4 results are available. Two cases are taken from The Aerospace Corporation's solution and two from ESA's Advanced Concepts Team's solution. For these solutions, the visited targets and their order are known. To create the problem instances, the initial departure from Earth and the visit of the 8th target are chosen as two different starting points. The following eight targets visited in the known solution then make up the set of targets available for the problem. This makes four instances. In all instances, 160 time steps of 5 days, a $C_{\max}$ of 10, a $C_{\max}$-multiplier of 10 and a population size of 20 are used.

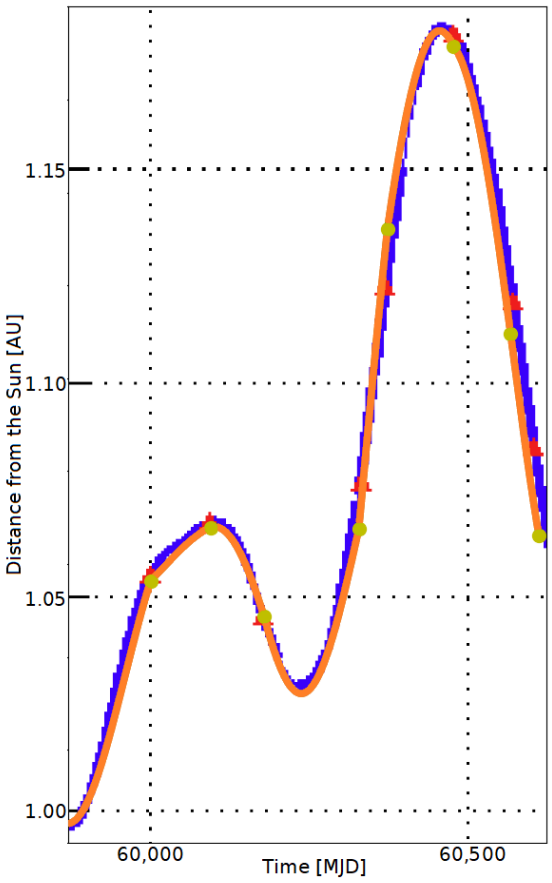

Figure 8: Comparison between the first 8 targets of the GTOC4 solution from ESA's Advanced Concepts Team,[16] and a solution created with the proposed algorithm.

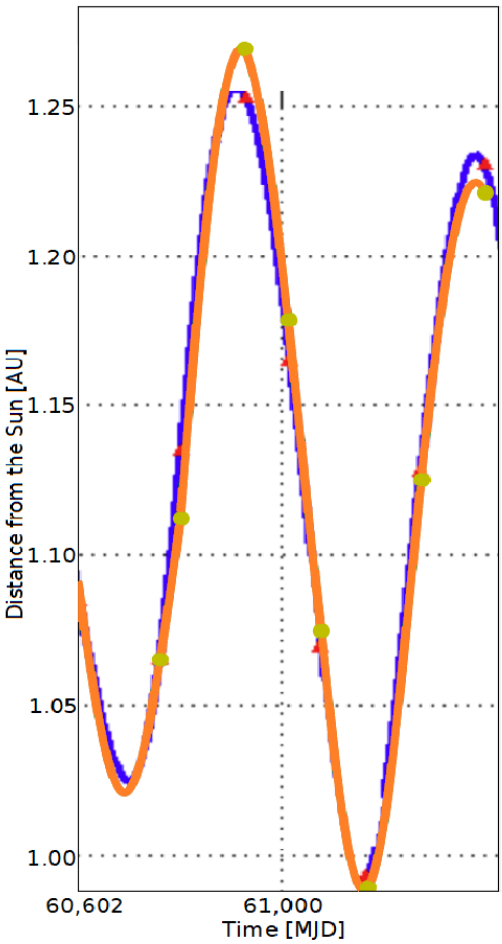

Figure 9: Comparison between the second 8 targets of the GTOC4 solution from ESA's Advanced Concepts Team,[16] and a solution created with the proposed algorithm.

All instances are solved five times, and the best solution found for each case is shown in Figures 6 - 9. In these plots, a close resemblance between the known solution and the solution produced with the proposed algorithm is seen. In all four cases, a solution is found that can visit all targets in the 8-target set, and in all cases, the targets are visited in the same order as in the known result. Still, the results are not perfect matches. Both Figure 6 and Figure 9 show differences at the plot peaks. Likely factors impacting these changes are numerical errors in propagated state histories for the targets. Depending on the initial state, the propagation might lead to a larger or smaller error in the propagated state. Also, the timing of the visits is relevant. Using a time step of five days is likely to lead to mismatches in visit times with the known solutions. This difference in time causes a difference in position, which is most notable in the trajectory's extreme points, the peaks in the plots. Figure 7 and Figure 8 show the most significant divergence between the plotted lines near the end of the trajectory. Of influence here is the difference in final conditions. The known solutions are segments of a longer trajectory. The visits of the last targets are, in those cases, chosen optimally, considering the following targets they will be visiting. The solution created with the algorithm is open-ended. This makes it likely that the optimal time and place to visit these last targets is different.

So, the differences can largely be attributed to numerical errors and implementation differences. They do not have a significant impact on the solution, which critically consists of the chosen targets and their order. These test cases show that the global optimisation algorithm can consistently and accurately re-produce the best-known solutions for problem instances with different starting points and target sets.

CONCLUSION

This paper proposes a global optimisation algorithm for complex combinatorial space routing problems. First, the problem is split into sub-problems, which are solved sequentially using SCIP. The sub-problems are all formulated as binary integer linear programmings. Solving the first sub-problem, called the Fixed Budget problem, produces a subset of targets from the problem instance, then solving the Full Tour problem orders these targets, and finally, solving the Fixed Tour problem optimises the time-of-flight for each leg of the trajectory to minimise fuel consumption. The global optimisation algorithm then searches for the initial values for which solving the three sub-problems sequentially produces the best possible solution.

The results show that the algorithm dramatically benefits from simplifications that were made. Turning off both the iterations within the evaluations and the Full Tour sub-problem reduces computational times, allowing an increase in the $C_{\max}$ multiplier, which increases the chance of finding solutions with a higher objective value. Using a $C_{\max}$ multiplier of 10 has shown the best performance for the considered test case. Using this, the optimal solution was found in every one of the five runs, in an average of 9.0 optimisation iterations, with an average computation time of 5.82 seconds per evaluation. The algorithm can also reproduce the best-known result for all four considered test cases taken from GTOC4. This shows that the proposed optimisation algorithm and the underlying sub-problems are valid and produce accurate results. However, considering scalability to larger, more complex problem instances, improvements are needed before this algorithm can practically be applied. Solving the Fixed Budget sub-problem requires the grand majority of computational resources. To be practical, the solving time for this specific step should be improved considerably. Future research into improving this specific solving step by tuning the setup and hyper-parameters or considering different problem methodologies or formulations could be of interest.

REFERENCES

[1] L. Casalino, G. Colasurdo, and M. R. Sentinella, "Problem Description for the 3rd Global Trajectory Optimisation Competition," 2007.

[2] A. E. Petropoulos, "Problem Description for the 6th Global Trajectory Optimisation Competition," 2012.

[3] D. Izzo, "Problem description for the 9th Global Trajectory Optimisation Competition," 2017.

[4] A. Lodi, "MIP Computation," *50 years of integer programming 1958-2008*, 2009.

[5] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj, "Gomory cuts revisited," *Operations Research Letters*, 1996.

[6] R. Gomory, "An algorithm for the mixed integer problem," *Tech. Report RM-2597*, 1960.

[7] R. Bertrand, R. Epenoy, and B. Meyssignac, "Problem Description for the 4th Global Trajectory Optimisation Competition," 2009.

[8] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. v. Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. v. d. Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 8.0," technical report, Optimization Online, 12 2021.

[9] T. Achterberg, "SCIP: solving constraint integer programs," *Mathematical Programming Computation*, 2009.

[10] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to Run Heuristics in Tree Search," *IJCAI*, 2017.

[11] D. Izzo and M. Märtens, "The Kessler Run: On the Design of the GTOC9 Challenge," *Acta Futura 11*, 2018.

[12] D. Dirkx, M. Fayolle, G. Garrett, M. Avillez, K. Cowan, S. Cowan, J. Encarnacao, C. F. Lombrana, J. Gaffarel, J. Hener, X. Hu, M. v. Nistelrooij, F. Oggionni1, and M. Plumaris, "The open-source astrodynamics Tudatpy software – overview for planetary mission design and science analysis," *Europlanet Science Congress 2022*, 2022.

[13] D. Izzo, "Revisiting Lambert's Problem," *Celestial Mechanics and Dynamical Astronomy*, 2015.

[14] S. J. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano, "PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite," *Mathematical Software - ICMS 2016*, Vol. 9725, 2016, pp. 301 – 307.

[15] F. Biscani and D. Izzo, "A parallel global multiobjective framework for optimization: pagmo," *Journal of Open Source Software*, Vol. 5, No. 53, 2020, p. 2338, 10.21105/joss.02338.

[16] R. Bertrand, R. Epenoy, and B. Meyssignac, "Final Results of the 4th Global Trajectory Optimisation Competition," 2009.

[17] K. Mehlhorn and P. Sanders, *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[18] J. Clausen, "Branch and Bound Algorithms - Principles and Examples," 1999.

3

Dynamical models

In this chapter, additional context and explanation on the dynamical and propulsion models used in the context of the solver proposed in the paper in Chapter 2 is provided. Before the global optimisation algorithm can be used to start solving the space-routing problem, a full cost matrix is needed. This matrix contains the ΔV required for every possible fly-by at every time step. To compute these ΔV values, the state histories for all targets in the problem are needed. In Section 3.1, the orbit propagation used to create these state histories is discussed. In determining the cost of travelling from one point in space to another, it is important to consider the propulsion system, explained in Section 3.2. Finally, the method of calculating the required ΔV, using the propulsion system, is described in Section 3.3.

3.1. Orbit Propagation

In the GTOC4 [8] problem, which is used as a test case, targets are considered for which initial Kepler elements are provided. The state histories are then created by propagating the orbits from this initial Kepler state, using the TUDAT software [24]. All propagation environments are set up using TUDAT's standard kernels and the J2000 orientation frame. For values that are not explicitly mentioned, defaults are used.

Table 3.1: Constants

Parameter	Value
μ_S, km^3/s^2	$1.32712440018 \times 10^{11}$
AU, km	$1.49597870691 \times 10^{8}$

First, the Keplerian elements are converted to a Cartesian state, using the Sun as the central body. This is done with TUDAT's `keplerian_to_cartesian` function and the constants found in Table 3.1. Angles given in degrees are converted to radians using Numpy's `deg2rad`, and the given mean anomaly is converted to a true anomaly using TUDAT's `mean_to_true_anomaly`. This Cartesian state will be used as the initial state for the propagation. The only force considered in the propagation is the point-mass gravitational force from the Sun, using the μ_S value from Table 3.1. The fourth-order Runge-Kutta method is used to integrate, and the translational propagator from TUDAT is used. The total time for which the state is propagated depends on the problem instance. The start time for the propagation is determined by the epoch of the provided initial state, and the termination time is determined by the problem's start time and the number and duration of time steps.

The propagation does not necessarily use the same time-step duration as the optimisation problem. The reason is that the accuracy for propagated states quickly reduces when larger time steps are used. The propagation only needs to be performed once at the start of the solver run and takes little computational time relative to the rest of the computation. This allows the use of a shorter step duration, providing more accurate state histories. To make a trade-off, the position errors for different time step durations are compared. As a baseline, a time step of 1 hour is used. The position errors relative to this baseline,

Table 3.2: Accuracy of propagation with different time-step durations

Time-step	Absolute error [km]	Relative error [%]	Run-time [s]
2 hours	0.0390	1.784×10^{-8}	0.842
4 hours	0.9959	4.704×10^{-7}	0.411
12 hours	194.4	9.531×10^{-5}	0.135
24 hours	5830	2.893×10^{-3}	0.066
48 hours	2.222×10^{6}	1.210	0.033

using longer time steps, are provided in Table 3.2. The errors are averaged over 45 propagated objects taken from the GTOC4 data set. The 44 asteroids visited by MSU's solution [22] and Earth are used as objects. Both the absolute error and relative error are taken for the final position of the object after propagating for a total duration of 12 years. The absolute error is computed using Equation (3.1), and the relative error is calculated using Equation (3.2). In both, $\vec{r}$ is the position vector where the subscript denotes the time step. For the problem instances solved in the paper, a maximum of 10 objects need to be propagated for a total duration of roughly 15 years. With that context, the 4-hour time step is considered a reasonable balance between run time and accuracy. The average absolute error of <1 km is sufficient to make meaningful comparisons between results using these state histories and the results from GTOC4. Therefore, a time step of 4 hours is used for all propagations in Chapter 2.

$$\text{Absolute error} = |\vec{r}_t - \vec{r}_{1h}| \tag{3.1}$$

$$\text{Relative error} = \frac{|\vec{r}_t - \vec{r}_{1h}|}{|\vec{r}_{1h}|} \tag{3.2}$$

The propagation is done using the methods published in the TUDAT software. None of the methods or computations is custom implemented. As the published software used is considered to be verified and validated, the results for these propagations can be regarded as valid.

3.2. Propulsion Model

Velocity changes the spacecraft makes along the trajectory are modelled as impulsive shots. At the time a manoeuvre is made, the velocity of the spacecraft changes instantly. There is no restriction on the direction or the magnitude of the velocity change. Practically, at every fly-by, the velocity of the spacecraft is instantly changed from its current velocity to the velocity that is needed to visit the next target. How the required velocity is determined is explained in Section 3.3.

In the optimisation algorithm, the magnitude of the velocity changes is considered the cost of a manoeuvre. In practice, to get this change in velocity, a propulsion system is needed, which uses fuel to change the spacecraft's velocity. The exact amount of fuel required would depend on the specifications of the propulsion system and the mass of the spacecraft. As there is no need to compare different spacecraft or different propulsion systems to optimise this specific problem, the ΔV can directly be used as a cost parameter, and it can be considered equivalent to consumed fuel mass.

3.3. Lambert Problem

To solve for the optimal trajectory from target to target, considering impulsive shots are used at the time of flying by a target, Lambert solutions [4] are used. Specifically, the `LambertTargeterIzzo` implementation from TUDAT [28] is used. To solve a Lambert problem, the initial position, final position, flight time and gravitational parameter of the central body are needed as inputs. The flight time can have a value of any integer number of time steps. The initial and final positions are taken from the propagated state histories, where the time at the final position is the initial time plus the flight time. The μ_S value from Table 3.1 is used for the central body. The targeter finds an exact Kepler orbit matching the given inputs, giving the velocity vectors belonging to that orbit in the initial and final positions. These velocities

are then used to determine the needed ΔV at every target fly-by.

For example, to solve the trajectory from A to B to C, given the time of flight for both legs, the Lambert targeter is used for both the leg from A to B and the leg from B to C. Both will give a velocity vector at point B; one is the resulting velocity from the leg from A to B, and the other is the initial velocity for the leg from B to C. The ΔV needed in point B is the magnitude of the difference between these two velocity vectors. This method can be used to determine the required ΔV at every target along the trajectory.

As the TUDAT code base is considered verified and validated, the results produced with the `LambertTargeterIzzo` are also regarded as valid. The velocity vectors from the Lambert targeter will be used to propagate the spacecraft's trajectory. To prevent the trajectory from diverging due to growing numerical errors, the spacecraft trajectory is propagated leg by leg. At the start of a leg, the spacecraft is assumed to be at the exact position of the target it visits. The initial velocity at that position is taken from the Lambert solution. This initial state is then used to propagate the orbit to the next visit, again using a 4-hour time step.

Table 3.3: Accuracy of propagation of Lambert solutions

Fly-by	Δx [km]	Δy [km]	Δz [km]	Δr [km]
1	1.376×10^{-3}	4.011×10^{-2}	7.511×10^{-5}	-1.214×10^{-2}
2	-9.678×10^{-4}	2.773×10^{-1}	-3.511×10^{-3}	2.067×10^{-1}
3	-4.446×10^{-2}	1.006×10^{-1}	-2.449×10^{-4}	1.089×10^{-1}
4	4.952×10^{-2}	-2.338×10^{-2}	2.947×10^{-3}	-1.233×10^{-2}
5	-4.315×10^{-1}	3.352×10^{-2}	8.204×10^{-5}	-4.035×10^{-1}
6	2.949×10^{-1}	6.247×10^{-4}	4.631×10^{-3}	2.948×10^{-1}

Table 3.3 shows the position errors found at every fly-by in the 6-target solution also used in the paper's Global Optimisation section. The delta value is the difference between the spacecraft position at the end of the propagated leg and the target position taken from the earlier propagated state history. The table shows the absolute position errors are <1 km in every case, and the errors do not grow throughout the trajectory. Note that these are the position differences at the end of each leg. At the start of a leg, the position error is nullified by assuming the spacecraft is at the exact position of the visited target. The errors are considered acceptable for all purposes in this research.

4

Mixed-Integer Linear Problem Optimisation

In this chapter, additional context and explanation on the implementation and optimisation of MILPs in the context of the optimisation algorithm proposed in the paper in Chapter 2 is provided. The algorithm implements and optimises different problem variations using the PySCIPOpt software [25], which will be discussed in the following sections. In Section 4.1, the implementation of the similar Fixed Budget and Full Tour sub-problems is discussed. Also, an explanation of the verification and validation steps is provided. Then, in Section 4.2, the implementation for the Fixed Tour sub-problem is explained. The mathematical formulation for these sub-problems can be found in the following sections of the paper: Paper/Target selection (page 9), Paper/Target ordering (page 13) and Paper/Trajectory optimisation (page 15). The focus of this chapter will be the specific software implementation that is used.

4.1. Fixed Budget and Full Tour

The Fixed Budget and Full Tour sub-problems are very similar concerning their implementation and setup. Although the optimisation objectives are different, they use almost identical constraints. The Fixed Budget sub-problem maximises the number of targets for a fixed cost, while the Full Tour sub-problem minimises the cost for a fixed set of targets. Because of these similarities, the implementation and, verification and validation of these sub-problems will be discussed simultaneously in this section.

4.1.1. Implementation

The sub-problem instances are solved with the PySCIPOpt code library [25]. The `Model` class is used to set up the problems, as formulated in the Target Selection and Target Ordering sections of Chapter 2. First, the x variables are created using `Model.addVar`. Every x variable represents a possible fly-by. It corresponds with a time t, previous target i, fly-by target j and next target k. A variable is created for every possible fly-by, meaning every combination of unique i, j, and k at every available time. All x variables are binary variables, where a value of 1 means the specific fly-by is part of the solution, and a value of 0 means it is not. Next, for every x variable, the ΔV cost values are computed using the method described in Section 3.3. These are then used to add the relevant constraints to the `Model` instance using `Model.addCons`, and the optimisation objective is set using `Model.setObjective`. In the Fixed Budget case, the objective is the number of visited targets, which is maximised, while in the Full Tour case, the objective is the total ΔV, which is minimised. After this setup, the `Model.optimize` method is used to find the optimal solution. No custom settings or implementations are used except for the `Presolve` and `Heuristics` settings, which are set using `Model.setPresolve` and `Model.setHeuristics`. The used settings differ per sub-problem and are explained in Chapter 2.

It could be the case that no feasible solution can be found. In this case, an `infeasible` status is returned. If a solution is found, a check is done to make sure the solution does not contain disjoint sub-tours. The pre-set constraints do not exclude this situation, where a solution consists of two or more smaller closed loops that are not connected. It is not feasible to pre-set all constraints to prevent these sub-tours, as the set of constraints needed to eliminate all sub-tours has a magnitude of 2^N, where N is the number of targets in the problem instance. Instead, the constraint like Equation (4.1) is added only if a solution containing sub-tours is found. This constraint prevents the specific sub-tours that are found from being part of the solution in the future. S is a sub-tour, which contains a subset of the targets in the problem. After adding a constraint for every sub-tour in the solution, the problem is optimised again, and this process is repeated until a solution is found that consists of a single tour.

$$\sum_{t=0}^{n_t} \sum_{i \in S} \sum_{j \in S} \sum_{k \in S} x_{t,i,j,k} \leq |S| - 1, \quad \text{for all} \quad S, \quad \text{with} \quad i \neq j, i \neq k, j \neq k \tag{4.1}$$

4.1.2. Verification and Validation

To verify the accuracy and correctness of the optimisation setups, the outputs of multiple optimisation runs with random initialisation for both the Fixed Budget, and Full Tour problems are manually checked to be accurate and feasible. The solution in both cases is in the form of a sequence of targets, with a flight time and a ΔV cost for every leg of the trajectory. For every leg of the solution, the cost is re-computed using the `LambertTargeterIzzo`. In all considered cases, the re-computed values exactly match the optimisation result. This shows that the cost computation is correctly integrated into the optimisation algorithm. Also, none of the inspected results are found to contain disjoint sub-tours, which proves the discussed method of preventing sub-tours is effective and well-implemented.

Validation is more complicated, as there are no problems with this exact format for which the optimal solution has been published to be used for comparison. Still, a trivial way to validate the functionality of the optimisation is to observe the progression of solutions over iterations. In all the observed cases, the convergence curves show the expected pattern, where the best-known solution first improves quickly, and the improvement rate slows down as the solution converges towards the optimal solution. The optimal solution for the test cases are derived from the known GTOC4 solutions. The final returned solutions are valid and feasible in all inspected cases.

4.2. Fixed Tour

In the Fixed Tour sub-problem, the targets in the trajectory and their order are fixed, while the time of flight for each leg is varied in search of the solution with the lowest total ΔV. These differences require changes in the implementation, which will be discussed in this section.

4.2.1. Implementation

The problem is implemented and optimised using PySCIPOpt's `Model` class. First, the binary x variables are added using `Model.addVar`. Note that these x variables differ from the x variables mentioned in the previous section. They do still represent specific fly-bys that are either part of the solution (value 1) or not (value 0), but the possible fly-bys are defined differently, as the order of targets is now fixed. This difference is further explained in Paper/Trajectory optimisation (page 15). The ΔV cost for every x variable is computed using the same method as before. Also, a set of possible visit times for every target is stored. This set depends on the possible visit times for the previous target and the set of possible flight times for the inbound trajectory leg. The set of possible time of flight values is a range of durations. The length of the range is a sub-problem parameter. Figure 4.1 visualises the possible visit times. The arrows represent the different possible flight times between the sequential targets in the trajectory. In this case, three time of flight values are available for every trajectory leg. It can be seen that along the trajectory, with every next target, the amount of possible visit times grows. Finally, the constraints are added using `Model.addCons`, the settings to be used are set using `Model.setPresolve` and `Model.setHeuristics` and the objective is set using `Model.setObjective`. The objective is to minimise the total ΔV cost.

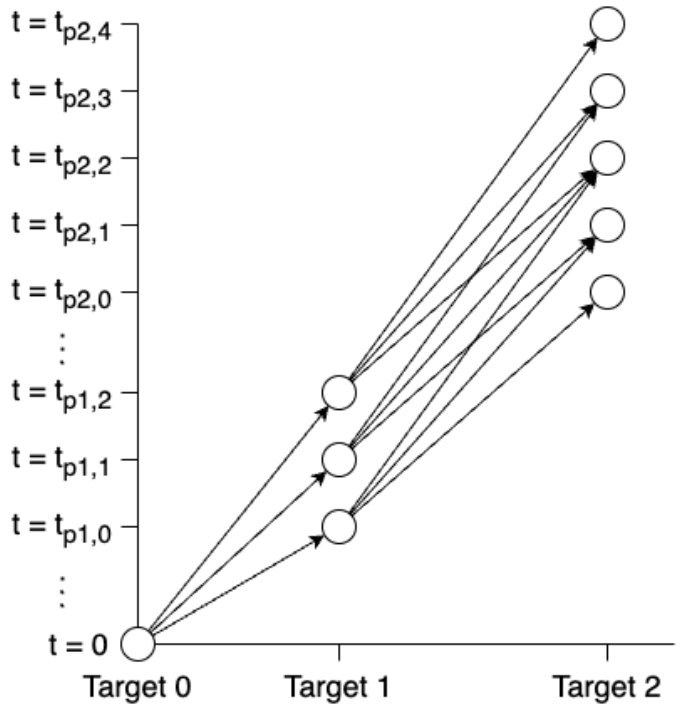

Figure 4.1: Visualisation of possible visit times, t_p

This optimisation will always return a feasible solution, as the solution from the Full Tour problem is always available within the constraints of the Fixed Tour problem. It is more likely that a better solution is found where some or all of the time of flight values are different from the previous solution. The solution is restricted by the time of flight ranges. Suppose one or more of the time of flight values in the optimal solution is at the very edge of the range of available values. In that case, it is assumed that a solution with a lower total cost might be available outside of the current ranges. So, in this case, for the trajectory legs where the value is found at the edge of the range, the range is shifted to find a better solution. If the value is at the bottom of the range, the range is moved, so the previous value is not at the top of the range. The constraints for the optimisation problem are then reset with the new ranges, and the problem is solved again. In this further optimisation, the previous solution is always available, as the prior time of flight values are still available in the new ranges. Iterations, where ranges are shifted, will continue as long as any time of flight values are found at the edge of the range. A solution will be accepted as optimal when, for all of the time of flight values, it is not at the edge of the range or the same value as in the previous solution.

4.2.2. Verification and Validation

The verification and validation done for this optimisation step are the same as discussed in Section 4.1.2.

5

Global Optimisation

This chapter provides additional context and explanation on the implementation of the global optimisation algorithm as proposed in the paper in Chapter 2. Solving the sub-problems using PySCIPOpt, discussed in the previous chapter, provides a solution that depends heavily on the time of flight values initially supplied. It is challenging to determine or predict the initial time of flight values that will provide the optimal solution. So, to find a globally optimal solution to the problem, an algorithm is implemented to search for the initial values that lead to the best solution. The first section of this chapter focuses on the specific implementation of the algorithm, and the second section provides additional context on the analysis done in Paper/Global optimisation (page 18).

5.1. Global Optimisation Algorithm

The global optimisation algorithm is implemented using the PyGMO software library [21]. A pseudo-code version of the algorithm can be found in Algorithm 1. First, all orbits are propagated to obtain state histories for all relevant bodies, using the methods explained in Section 3.1. Then, a PyGMO population is initiated. An individual in the population consists of an integer time of flight value for every (t, i, j), where t is any epoch, i is any of the targets that are part of the problem, and j is any target except i.

Different methods of initialising the population are discussed in the Global Optimisation section of Chapter 2. Note that all the variables are integers, and all are bounded by $[1, n_t]$. After initialisation, the optimisation iterations start. The algorithm used to optimise is the Extended Ant Colony Optimization algorithm (GACO) [29] implemented by PyGMO. This algorithm is chosen as it can optimise the custom fitness function, which in this case consists of solving the MILP sub-problems, and it allows for batch evaluation of the population fitness. This means that the `for`-loop over individuals in the population can run several loops in parallel. As explained in Chapter 2, ten parallel evaluations are done in most of the cases seen in the paper. The default GACO setup is used for all relevant parameters. The parameters that can be varied are the size of the population and the total number of iterations.

The $C_{\max,\text{temp}}$ value is used within the loop on lines 7-13 only, while $C_{\max}$ is the global problem parameter. $C_{\max,\text{temp}}$ will always be larger than the $C_{\max}$ value because of the multiplier that is applied in line 6. This larger $C_{\max,\text{temp}}$ value increases the feasible space in which a solution can be found in the Fixed Budget sub-problem, and the steps on lines 10 and 11 will reduce the C_{opt} value of the solution. If it is then below the global $C_{\max}$, the algorithm continues. If it is not, in the initial version of the algorithm, a new iteration is made where the $C_{\max,\text{temp}}$ is updated with the C_{opt} value of the last iteration.

Some changes are made to this algorithm throughout the research and analysis in the paper. In a trade-off between computation time and solution quality, the solving of the Full Tour sub-problem on line 10 is skipped. In this case, the Fixed Tour sub-problem is solved with the targets and order from the Fixed Budget solution. Also, the `while`-loop on line 7 is eliminated. The code within the loop is still

Algorithm 1 Global optimisation algorithm using GACO

```
Propagate Earth and target orbits
Initialise population
for iteration in range(N_iterations) do
    Evolve population (PyGMO GACO algorithm)
    for individual in population do
        C_opt = C_max × multiplier
        while C_opt > C_max do
            C_max,temp = C_opt − 0.01
            Solve Fixed Budget problem with a ΔV budget of C_max,temp
            Solve Full Tour problem with the sub-set of targets from the Fixed Budget solution
            Solve Fixed Tour problem with the targets and order from the Full Tour solution
            Update C_opt with the value from the Fixed Tour solution
        end while
        if Solution = infeasible then
            N_targets = 0
            C_opt = C_max × 2
        end if
        Fitness = −(N_targets in solution) × 100 + C_opt
    end for
    Update and store Best Individual and Best Fitness
end for
```

executed, but only once. If the C_{opt} value after that one execution is larger than $C_{\max}$, the solution is considered `infeasible`.

5.2. Performance Analysis

To analyse and compare the results produced with the global optimisation algorithm, relevant parameters are stored with every fitness evaluation. With this data, the tables and plots in Paper/Global optimisation (page 18) are created. The optimisation curves in the plots, showing the progression of both the average and best-known objective values in the population over time, improve significantly in the early stages as the algorithm explores the solution space. Then, as the algorithm begins to exploit the best solutions found, the improvements in fitness become more incremental. This behaviour is typical of global optimisation algorithms approaching an optimum.

Finally, a comparison is made between the results produced by the global optimisation algorithm and the known solutions from GTOC4 [22]. These comparisons can be seen in Paper/Figure 6 - Paper/Figure 9. The comparison using a time vs radius plot is not the most obvious, but this is the only way in which the GTOC4 results are available, and it does give a clear visual comparison of the results. The four problem instances used for comparison are created by taking a starting time and position derived from the GTOC4 result and setting up the problem with the next eight targets visited by the solution from that starting point. Table 5.1 shows the initial conditions and targets for each of the comparative problem instances. The similarity seen in the plots gives confidence that this algorithm finds the optimal solution in all of these test cases. Also, it provides confidence that the global optimisation algorithm and all its underlying parts produce accurate and feasible results within the boundaries given by the GTOC4 problem.

Table 5.1: Initial conditions and targets for the comparative problem instances

	Figure 6	**Figure 7**	**Figure 8**	**Figure 9**
Start time [MJD]	58662	59306	59868	60602
Starting body	Earth	2007DC	Earth	2004QJ13
Targets	2006QV89	2005NJ63	2003BN4	2004YC
	1999TM12	2007FO3	2004XN44	2005YP180
	2003YT70	1999VS6	2006UB17	2006HU30
	164207	2006BC8	2002AU4	2007TK15
	137126	2007LF	2005BN1	2006RJ1
	2004TP1	2003OQ13	4581	1998XX2
	1998BT13	2003WY153	2005NW44	2006AU3
	2007DC	2006KV89	2004QJ13	2001RQ17

6

Conclusions & Recommendations

In this section, the findings of the study will be presented. The described problem, the proposed solving methodology, and its results are considered in the context of the research objectives posed in Section 1.1. Also, recommendations for future work are provided.

6.1. Conclusions

The purpose of this paper was to study novel methods utilising MILP-solving tools to solve complex space routing problems. For this purpose, a new solving framework, implementing MILP solving techniques, was created.

This report presented and studied a framework for solving complex, combinatorial space routing problems. Splitting the problem up into linear sub-problems allowed the application of SCIP, a powerful MILP-solving software. Solving the sub-problems in sequence gave valid solutions to the whole problem instance. However, these solutions were only locally optimal and highly dependent on provided initial time-of-flight values. A global optimisation layer using PyGMO's GACO algorithm was added to reach better global optimal solutions.

At the start of this thesis, some research objectives related to different areas of the design and analysis of the global optimisation algorithm were defined. Below is a summary of the results that were found in the context of these objectives:

- How can MILP-solving techniques be used to solve complex combinatorial space routing problems?

  To allow the use of MILP-solving techniques, the considered problem needed to be in linear form. Instead of linearising the entire instance at once, the choice was made to split the instance into linear sub-problems. These splits were made by adding constraints and fixing variables. Solving the sub-problems sequentially, using SCIP, where every sub-problem takes input from the sub-problem solved before, leads to a final solution.

  - What are the primary sub-problems, and how do they function?

    A split into three different sub-problems was found to work well. These are:

    * Fixed Budget:
      The Fixed Budget sub-problem aims to select a subset of all available targets in the problem instance. To do so, time-of-flight values for every possible transfer between two targets are taken as input. These values are used to compute the needed velocity change for all possible transfers, which is used to find a trajectory through the maximum number of targets within the given budget. The returned solution is guaranteed a valid trajectory within the cost budget. Still, it is not guaranteed to be optimal regarding the targets' visiting order or times-of-flight between them.

    * Full Tour:
      The Full Tour sub-problem aims to find the optimal order of visiting the targets in the subset provided by the Fixed Budget sub-problem. The same time-of-flight values are used to determine the necessary velocity change for all possible transfers. However, the optimisation objective now is to find the trajectory that visits all the given targets with the lowest possible total velocity change. The returned solution is either the same trajectory as seen in the previous step or one with a different order and a lower total cost. Still, the times-of-flight between targets are not optimised.
    * Fixed Tour:
      The purpose of the Fixed Tour sub-problem is to optimise the times-of-flight for every leg of the trajectory that resulted from the Full Tour sub-problem. For every leg of the trajectory, the time-of-flight is considered in a small discrete range of values. A ”trajectory tree” can be expanded with these ranges, containing all possible trajectory legs, target visiting times, and transfer times-of-flight. Optimising for the lowest total velocity change gives the optimal time-of-flight per leg. For every time-of-flight value found to be on a bound of the discrete range, the range is shifted, and the problem is solved again. Iterations are made until no bounds need to be moved or no better solution can be found. The times-of-flight given in the solution to this sub-problem are guaranteed to be the values for which visiting the given targets in the given order takes the lowest possible velocity change.

  – What solver settings provide the best performance for solving the isolated sub-problems?

    The performance impact of the top-level SCIP settings ”Presolve” and ”Heuristics”, which determine the use and aggressiveness of default pre-solving techniques and default heuristics during the optimisation of the given problem, was studied. For all sub-problems, it was found that turning both settings completely off led to a significant decrease in the computational time needed to find a solution to the test cases.

    For the Fixed Budget and Fixed Tour sub-problems, the settings were found to have no impact on the quality of the solutions. For these cases, it was concluded that turning the settings off leads to the best performance, which is the same result in a reduced amount of time.

    In contrast, for the Full Tour sub-problem, it was found that using more aggressive settings for both Presolve and Heuristics increased the chance of finding a target order with a lower total cost. The impact on the integrated global optimisation algorithm was studied to make a trade-off between the effects of reduced computational time. This led to the conclusion that the increased chance of finding a better solution did not outweigh the reduction in computational time.

    So, for all sub-problems, turning both Presolve and Heuristics settings off showed the best performance regarding a combination of solution quality and computation time.

- How do further simplifications to the solving algorithm impact the performance?

  Performance analysis of the global optimisation algorithm identified multiple possible inefficiencies in the setup. Firstly, with the optimal SCIP settings for the Full Tour sub-problem, it was shown that the chance of finding a better solution in this step is only 4.5%. Therefore, the consideration was made to remove this sub-problem from the algorithm and directly input the targets in the order found in the Fixed Budget sub-problem into the Fixed Tour sub-problem. The analysis has shown that removing the sub-problem from the algorithm significantly reduces the computation time, while the solution quality was not noticeably reduced.

In the original algorithm iterations were used, solving the three sub-problems in a loop until a feasible solution was found. The analysis showed that these iterations were ineffective in finding the optimal solution. In all observed cases, the best solution was found for an individual where a feasible solution was found within the first iteration. Individuals that needed multiple iterations to find a feasible solution never returned a solution that would be globally optimal. For that reason, the iterations within the algorithm were disabled.

- What is the impact of removing the Full Tour sub-problem from the algorithm?

  The performance while skipping the Full Tour sub-problem was analysed in two scenarios. First, cutting the sub-problem did not show a performance improvement in the case where all individuals are iterated until feasible. Compared to the base setup, the chance of finding the best-known solution was slightly higher, but it would take, on average, 87% more generations to reach the best solution. This made it significantly slower to find good solutions. However, the performance of removing the sub-problem in combination with turning off the iterations shows a significant improvement. Then, it found the best solution in 55% fewer generations while computational time per generation is 8% shorter. When the $C_{\max}$ multiplier was increased, this led to even better results, with a further computation time reduction of 13% and a slight increase in the number of targets in the average solution. So, removing the Full Tour sub-problem from the global optimisation algorithm is beneficial in combination with disabling the iterations.

- What is the impact of propagating infeasible individuals instead of iterating every individual to be feasible?

  It was found that individuals that needed iterations to return a feasible solution could never produce the known optimal solution. So, to improve the solving algorithm, they were given a very low objective value instead of iterating individuals that initially returned an infeasible solution. This would reduce the computational time spent on these non-optimal individuals, and the low associated objective would help to evolve away from these individuals towards more optimal ones. A combination of these effects can be seen in the performance analysis.

  For all cases observed with iterations disabled, the chance of finding the known optimal solution was much higher than for previous cases. Every single run led to the optimal 6-target trajectory. With a low $C_{\max}$ multiplier, it is seen that the average solution contained a larger number of targets but also that the computational time was higher than for the case with iterations. This was attributed to solutions with more targets taking longer to compute. When the $C_{\max}$ multiplier was increased, the average solution contained more targets, and computational requirements increased further. Most of the increase came from solving the Full Tour sub-problem. As mentioned in the previous answer, the optimal setup was found to be a combination of disabling the iteration and removing the Full Tour sub-problem.

- What is the effect of seeding the initial population with constant values instead of randomly generated values?

  Analysis showed that seeding the population with constant values positively affects performance. The most significant improvement was found in the solutions' quality in the first generation of global optimisation. Using several seeded individuals helped to reliably find a better solution in the initial generation, which helped to converge more quickly towards the known optimal solution. All runs using seeded individuals show a better chance of finding the known optimal solution. It should be noted that the average computational time per generation was slightly longer due to the average solution having a higher number of targets.

  - What number or proportion of the initial population would optimally be seeded?

    Three different proportions were compared, and the best performance was found when 10 out of 20 initial individuals were seeded. All three cases start with the same, higher than average, best solution in the first iteration. After that, the 10/20 case converged towards the

known optimal solution faster than the 5/20 and 20/20 cases. In the 10/20 case, all observed runs lead to the known optimal solution. This led to the conclusion that seeding at least half of the initial population is beneficial, but keeping part of the population randomly generated could help the convergence.

- How do the results from the proposed methodology compare with benchmark solutions from GTOC4?

  It was shown that the proposed global optimisation algorithm could produce solutions that almost exactly match the known GTOC4 solutions, considering missing segments with up to 8 targets. This was shown for multiple different target sets and starting points. This ability to reproduce results has provided a high-level validation of the proposed algorithm and the methods used within. The results were exact matches in terms of the targets visited and their order. Slight differences could still seen in the precise trajectories. This was explained by possible numerical differences in the propagation of the asteroid orbits and the different propulsion models used in the GTOC4 problem.

These results for the proposed novel methodology look promising. It was demonstrated that solving a complex space routing problem using the SCIP solver is possible by breaking it down into linear sub-problems. The efficiency of the algorithm was further improved by making available simplifications. Also, the algorithm can reproduce partial GTOC4 solutions well. However, further improvements to the methods are required to enable the algorithm to solve more extensive and complex problem instances. Some of the possible improvement directions are discussed in the next section.

6.2. Recommendations

In the final version of the algorithm, solving the Fixed Budget sub-problem limits the performance. Paper/Table 12 shows that this step takes up 99% of the required computation time, and Paper/Table 4 shows a significant sensitivity of the computational effort to the number of targets and the number of time steps in the problem instance. To scale the algorithm's capabilities to allow solving larger problem instances, the performance in this specific sub-problem has to be improved. Possible improvements can be made in different directions.

The optimisation problem formulation and implementation could be reviewed and improved to find a new formulation that leads to the same result more efficiently. Also, using a more specific SCIP setup could help. It might be possible to develop specific heuristics that help solve this specific problem more quickly. This is also relevant for the other sub-problems. In a search for MILP-compatible sub-problems, all sub-problem formulations turned out to be BILPs, a specific subset of MILPs where all variables are binary. So, for all sub-problems, it might be possible to use a more effective or specialised solver designed to solve BILPs. This could be SCIP using a specific setup with custom heuristics or a completely different software.

More drastic changes are also possible. Targets do not necessarily need to be selected by solving an optimisation problem. There might be possibilities where the global optimisation algorithm searches for a subset of targets as initial values instead of searching for initial ToF values to feed into the Fixed Budget sub-problem. Machine learning applications might be suitable in the search for this subset. Clustering and pruning techniques could be of great value in finding targets that can be combined into a good solution.

Another area that could have a significant impact on the performance of the global optimisation algorithm is the seeding of the initial population. In this study, a straightforward policy was used to seed the population partially, leading to a significant increase in performance. Future research could investigate the use of more elaborate techniques or heuristics to improve the initial population further. This could potentially lead to faster convergence and better-quality solutions.

List of References

[1] Dario Izzo. “Lambert’s Problem for Exponential Sinusoids”. In: *Journal of guidance, control, and dynamics* (2006).

[2] Oliver Schütze et al. “Designing optimal low-thrust gravity-assist trajectories using space pruning and a multi-objective approach”. In: *Engineering Optimization* (2009).

[3] Bernardetta Addis et al. “A global optimization method for the design of space trajectories”. In: *Computational Optimization and Applications* (2011).

[4] Dario Izzo. “Revisiting Lambert’s Problem”. In: *Celestial Mechanics and Dynamical Astronomy* (2015).

[5] Daniel Hennes and Dario Izzo. “Interplanetary Trajectory Planning with Monte Carlo Tree Search”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[6] Haibin Shang and Yuxin Liu. “Assessing Accessibility of Main-Belt Asteroids Based on Gaussian Process Regression”. In: *Journal of Guidance, Control, and Dynamics* (2017).

[7] Lorenzo Casalino, Guido Colasurdo, and Matteo Rosa Sentinella. “Problem Description for the 3rd Global Trajectory Optimisation Competition”. In: (2007).

[8] Régis Bertrand, Richard Epenoy, and Benoît Meyssignac. “Problem Description for the 4th Global Trajectory Optimisation Competition”. In: (2009).

[9] Ilia S. Grigoriev and Maxim P. Zapletin. “GTOC5: Problem statement and notes on solution verification”. In: *Acta Futura 8* (2014).

[10] Dario Izzo. “Problem description for the 9th Global Trajectory Optimisation Competition”. In: (2017).

[11] G. Dantzig, R. Fulkerson, and S. Johnson. “Solution of a Large-Scale Traveling-Salesman Problem”. In: *Journal of the Operations Research Society of America* (1954).

[12] E. Balas. “The Prize Collecting Traveling Salesman Problem and its Applications”. In: *The Traveling Salesman Problem and its Variations* (2002), pp. 663–696.

[13] Karla L. Hoffman, Manfred Padberg, and Giovanni Rinaldi. “Traveling salesman problem”. In: *Encyclopedia of Operations Research and Management Science* (2013).

[14] A. Lodi. “MIP Computation”. In: *50 years of integer programming 1958-2008* (2009).

[15] E. Balas et al. “Gomory cuts revisited”. In: *Operations Research Letters* (1996).

[16] R.E. Gomory. “An algorithm for the mixed integer problem”. In: *Tech. Report RM-2597* (1960).

[17] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[18] Tobias Achterberg. “SCIP: solving constraint integer programs”. In: *Mathematical Programming Computation* (2009).

[19] Elias B. Khalil et al. “Learning to Run Heuristics in Tree Search”. In: *IJCAI* (2017).

[20] Dario Izzo and Marcus Märtens. “The Kessler Run: On the Design of the GTOC9 Challenge”. In: *Acta Futura 11* (2018).

[21] Francesco Biscani and Dario Izzo. “A parallel global multiobjective framework for optimization: pagmo”. In: *Journal of Open Source Software* 5.53 (2020), p. 2338. DOI: 10.21105/joss.02338. URL: https://doi.org/10.21105/joss.02338.

[22] Régis Bertrand, Richard Epenoy, and Benoît Meyssignac. “Final Results of the 4th Global Trajectory Optimisation Competition”. In: (2009).

[23] Anastassios E. Petropoulos. “Problem Description for the 6th Global Trajectory Optimisation Competition”. In: (2012).

[24] Dominic Dirkx et al. “The open-source astrodynamics Tudatpy software – overview for planetary mission design and science analysis”. In: *Europlanet Science Congress 2022* (2022).

[25] Stephen J. Maher et al. “PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite”. In: *Mathematical Software - ICMS 2016*. Vol. 9725. 2016, pp. 301–307.

[26] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[27] Jens Clausen. “Branch and Bound Algorithms - Principles and Examples”. In: (1999).

[28] TU Delft. *TU Delft Astrodynamics Toolbox - 5.2. Mission Segments*. 2018. URL: https://tudat.tudelft.nl/tutorials/tudatFeatures/astroTools/missionSegments.html (visited on 10/20/2021).

[29] Martin Schlueter, Jose A. Egea, and Julio R. Banga. “Extended ant colony optimization for non-convex mixed integer nonlinear programming”. In: *Computers & Operations Research* 36.7 (2009), pp. 2217–2229. DOI: 10.1016/j.cor.2008.08.015. URL: https://doi.org/10.1016/j.cor.2008.08.015.