Institut für
Automatisierungstechnik

Master Thesis

Analysis and Evaluation of Global Optimisation Algorithms for Gravity Assist Sequencing of Low Thrust Missions

Nandish Kuntikal Doddi Mahadevaiah

May 31, 2017

Tutor:
Dr.-Ing. Jochen Schüettler
Mr Volker Mailwald

1st Examiner:
Prof. Dr.-Ing. Kai Michels

Declaration of honour

I hereby confirm on my honour that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisor has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

Date: _____ Signature: _____

Declaration of publication

- ☐ I hereby agree, that my thesis will be available for third party review in purpose of academic research.
- ☐ I hereby agree, that my thesis will be available after 30 years (§7 Abs. 2 BremArchivG) in the university archive for third party review in purpose of academic research.
- ☐ I hereby do *not* agree, that my thesis will be available for third party review in purpose of academic research.

Datum: _____ Unterschrift: _____

Acknowledgement

I would like to take this opportunity to express my humble regards and gratitude towards my tutors Dr.-Ing. Jochen Schüttler and Mr. Volker Maiwald for their patience and their expert guidance in this Thesis work.

I would also show my appreciation and greatest respect to my Prof. Dr.-Ing. Kai Michels for his invaluable lectures and inspiring insights in the field of Control Systems.

Special thanks to Deutsches Zentrum für Luft- und Raumfahrt for giving me this opportunity to work with them and for their efforts, resources, infrastructure and their guidance during my time at the institute.

Last but not least I am very much thankful to my family, for their constant support and encouragement.

Abstract

In this thesis work "Analysis and Evaluation of Global Optimization Algorithms for Gravity-Assist sequencing of Low-Thrust missions" research is conducted to verify if the optimization algorithms can find near optimal trajectories for low-thrust gravity-assist trajectory optimization problem. In most of the low-thrust gravity-assist trajectory optimization, the gravity-assist sequence is fed to the optimizer by a mission analyst. The aim of this thesis is to analyse optimization algorithms after including the gravity assist partner as part of the optimization, i.e optimizer itself chooses the gravity-assist sequence. A couple of global optimization algorithms, Random Search and Simulated Annealing are realized in object oriented language(C++) and several experimental results are evaluated to understand the behaviour of these algorithms on a system which is trying to optimize the low-thrust gravity-assist trajectories. Suitability of the optimization algorithm which is to be used in an optimization tool, is verified.

Contents

1. Introduction

Space exploration has always fascinated humans. Ever since the first satellite **Sputnik** was launched, there are numerous attempts to explore outer space and understand the celestial bodies. Improving the efficiency of the missions and trajectory optimizations have taken huge importance in the field of space science. In recent times numerous researches have been conducted on trajectory optimization problem with the help of various optimization methodologies. This thesis work is one such effort to understand the behaviour of optimization algorithms for the problem of low-thrust gravity-assist trajectory optimization.

1.1. Motivation

The targets that can be reached by chemical rockets are limited because of massive propellant requirements. With the help of low-thrust gravity-assist trajectories it is possible for a spacecraft to reach longer distances and explore outer limits of the solar system. Exploration missions are becoming more and more ambitious and rely on gravity assist as free energy providers. For many years NASA, ESA and other space agencies have been using this method to extract freely available energy in space. Missions like **Voyager, Cassini, Messenger** and **New Horizons** all relied on gravity assists for accomplishing their missions. **Dawn** and **Hayabusa** missions showcase the benefits of low-thrust gravity-assist combination, even though the gravity assist was not mission critical for **Dawn** [26] [25]. The main goal of low-thrust gravity-assist missions is not only to reach distant targets but also decrease the mass of the required propellant, therefore increasing the payload of the mission. The potential fuel mass savings and energy benefits makes optimization of low-thrust and gravity-assist combination as one of the exciting research topics.

1.2. Objectives and Proceedings

Usually for low-thrust gravity-assist optimization, the sequence of gravity-assist partners (planets or satellites where the gravity assists occur) is simply fed to the optimization algorithm by an experienced mission analyst and is not part of

optimization of low-thrust mission scenario [25]. The method of allowing optimization algorithm itself to chose the required planet for the gravity assist has been put forward by Mr V.Maiwald [25]. This is a new approach towards the problem of optimizing low-thrust gravity-assist trajectories. By implementing such a condition the number of variables in the optimization problem increases along with increase in the complexity of the optimization problem.

Major task of this thesis work is to check the suitability of global optimization algorithms, if these algorithms would succeed in finding the optimal or near optimal low-thrust gravity-assist trajectories.

The objectives of the thesis:

* Analysis of suitability of at least 2 global optimization algorithms (Random Search to be used as bench mark) based on the ability to improve one leg missions with multi-leg missions.

* Analysis and evaluation of convergence to one solution for one, two and three leg missions.

* Discussion of the conclusions regarding the search space topography, based on the algorithm performance.

Initially overall system of low-thrust gravity-assist trajectory optimization is understood to implement the optimization logic. The provided codes are adapted to implement Random Search algorithm, followed by experiments and analysis of the results. Next follows implementation of Simulated Annealing algorithm for the given optimization problem. To verify the suitability of Simulated Annealing algorithm several experiments are conducted and corresponding results are analysed.

1.3. Report Structuring

This report consists of six further chapters explaining the thesis work in detail. Chapter 2 provides insight into the theoretical background, description of the low-thrust trajectories and optimization algorithms. Chapter 3 explains the state of art, mission parameters and general mission settings. Chapter 4 introduces to Random Search method and analysis of the same for the described system with the obtained results. Chapter 5 explains implementation of Simulated Annealing algorithm. Chapter 6 discusses the experiments and results of Simulated Annealing algorithm. Chapter 7 concludes the thesis with suggestions for future research.

2. Theoretical Background

2.1. Gravity Assist

Gravity assists are methods in which an interplanetary spacecraft travels longer distances with lesser consumption of fuel, with the help of gravitational energy of the celestial bodies it encounters on its path. Gravity assist results in change of both magnitude and direction of the velocity vector of a spacecraft. Gravity assists are also known as fly-by trajectories and are useful in interplanetary missions for reducing the ΔV(change in velocity of the spacecraft) cost of the mission and increasing the payload mass of the mission [9].

2.1.1. Physics behind Gravity Assist

Gravity assist can be seen as a carefully planned hopping of the spacecraft between the planets or planetary satellites. In this technique the spacecraft extracts energy from an orbiting planet. When a spacecraft passes through the gravitational field of an orbiting planet, according to Newton's second and third laws of motion there would be a change in momentum of the spacecraft due to planets gravity and change in planets momentum due to spacecraft, however the momentum of the entire system would be conserved [8]. In heliocentric coordinate system, spacecraft either gains or loses energy from this encounter depending on the position of the spacecraft with respect to the planet.

Fig 2.1 shows the hyperbolic fly by trajectory of the spacecraft relative to planet coordinate system.

Velocity of the spacecraft with respect to the planet when it is infinitely far away (outside the gravitational sphere of influence of the planet) is known as hyperbolic excess velocity and is represented as V_∞ [9].

$V_{\infty I}^{P}$ is the inbound hyperbolic excess velocity of the spacecraft in planetocentric system. $V_{\infty O}^{P}$ is the outbound hyperbolic excess velocity of the spacecraft in planetocentric system. V_{P}^{H} is the velocity of the planet in heliocentric system. δ is the angle at which the planet rotates the inbound hyperbolic excess velocity vector. α is the angle between the planet velocity vector and outbound hyperbolic excess velocity vector. R_{p} is the perienteric distance of fly

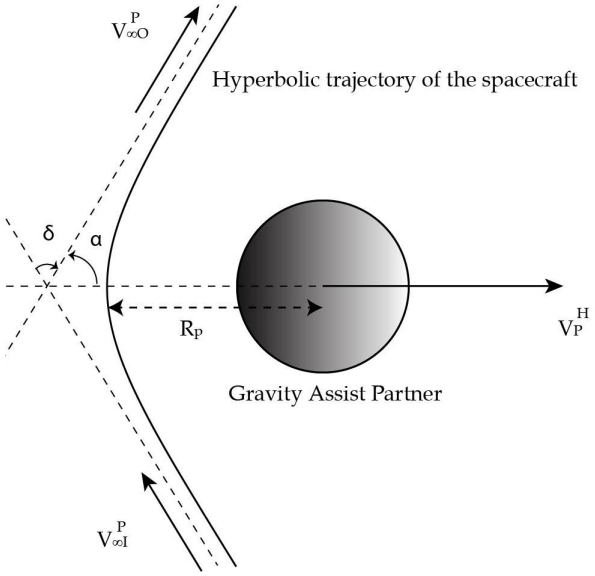


Figure 2.1.: Hyperbolic fly-by trajectory
[9]

by trajectory with respect to planet. [9]

The heliocentric velocities (both inbound and outbound) of the spacecraft can be found by vectorial addition of planets velocity vector with the hyperbolic excess velocity vector V_∞^P (considered in planetocentric system). The vector diagram for the same is shown in Fig 2.2.

$$V_{\mathrm{VI}}^H = V_{\infty\,\mathrm{I}}^P + V_{\mathrm{P}}^H \tag{2.1}$$

$$V_{\mathrm{VO}}^H = V_{\infty\,\mathrm{O}}^P + V_{\mathrm{P}}^H \tag{2.2}$$

V_{VI}^H is the inbound hyperbolic excess velocity of the spacecraft in heliocentric system. V_{VO}^H is the outbound hyperbolic excess velocity of the spacecraft in heliocentric system. ΔV_{FB} is the change in heliocentric velocity of the spacecraft.

The difference between the two values of the planetocentric hyperbolic excess velocities V_∞^P (both inbound and outbound) is equal to the difference between the heliocentric velocities of the spacecraft, shown in equation 2.3.

$$\Delta V_{\mathrm{FB}} = V_{\infty\,\mathrm{O}}^P - V_{\infty\,\mathrm{I}}^P = V_{\mathrm{VO}}^H - V_{\mathrm{VI}}^H [9] \tag{2.3}$$

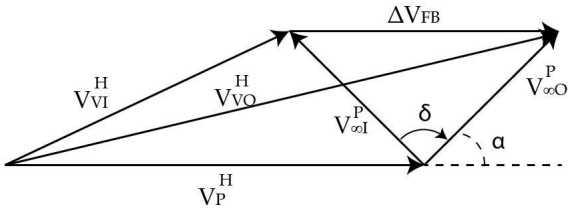


Figure 2.2.: Flyby Velocity Geometry
[9]

It can also be seen from Fig 2.2 that maximum ΔV_{FB} is obtained when the inbound hyperbolic excess velocity vector is rotated by an angle of 180°. [9]

$$\Delta V_{\mathrm{FB}} = 2V_{\infty} sin\left(\frac{\delta}{2}\right) \tag{2.4}$$

It is clear from equations 2.1 and 2.2 that the gain in heliocentric velocity of the spacecraft after its encounter with the planet is entirely dependent on the fact that the planet itself was in motion [8]. A typical example with numerical details of Pioneer 10 spacecraft's gravity assist around Jupiter is explained in [8].

2.2. Rocket Propulsion

Thrust is the force which makes the rocket move through space. Thrust is produced by the propulsion system of the rocket during expulsion of the propellant mass. Basically there are two means of rocket propulsion. It can be classified as chemical and non chemical rocket propulsion systems. Chemical rockets are propelled due to expansion of high pressure gases which are expelled through the nozzle of the rocket to provide thrust. The high pressure gas expansion is obtained by burning the propellant made of fuel and oxidizers. In contrast to chemical rockets non chemical rockets are propelled by accelerating charge particles out of the nozzle using a magnet or electrostatic forces. Non chemical rockets use electric propulsion engines, ion thrusters and are powered by generators, solar panels or nuclear power sources. [11]

Thrust produced by a rocket engine is given by equation 2.5 [7].

$$F = c\frac{dm}{dt} \tag{2.5}$$

where c is the exhaust velocity (velocity of the exhaust gases in case of chemical

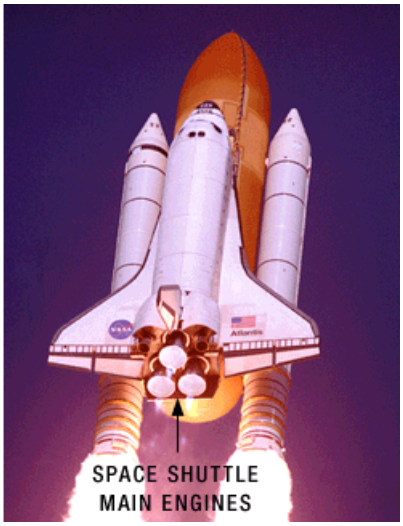


Figure 2.3.: NASA space shuttle main engines
[13]

propulsions and charge particles in case of electric propulsions). $\frac{dm}{dt}$ is the rate at which the materials are expelled out of the rocket system.

The velocity attained by the rocket due to propulsion after neglecting the gravity and drag effects is given by equation 2.6 [7].

$$\Delta V = v_{\mathrm{f}} - v_{\mathrm{o}} = -c \ln \frac{m_{\mathrm{f}}}{m_{\mathrm{o}}} \tag{2.6}$$

where ΔV is the velocity change, v_{f} is the final velocity of the rocket, v_{o} is the initial velocity of the rocket, m_{f} is final mass of the rocket, m_{o} is the initial mass of the rocket and c is the exhaust velocity.

Chemical rockets have higher rates of propellent mass ejection but are restricted by the exhaust velocities. "Today's most efficient operational rocket propulsion system is the main engine of U.S space shuttle, which performs near its theoretical limits with an exhaust velocity of 4.5km/s in vacuum" [15]. Fig 2.3 shows chemical engines used in NASA space shuttle.

For missions requiring to deliver significant amount of rocket's mass to its destination, or missions with large requirements of ΔV, clearly the exhaust velocities have to reach higher values. Low-thrust rocket engine exhaust velocities can go up to 20 times greater than exhaust velocities achieved in chemical rockets, hence requires far less propellent masses. Because of small mass ejection, low-thrust rocket engines push the spacecraft in a gentle manner compared

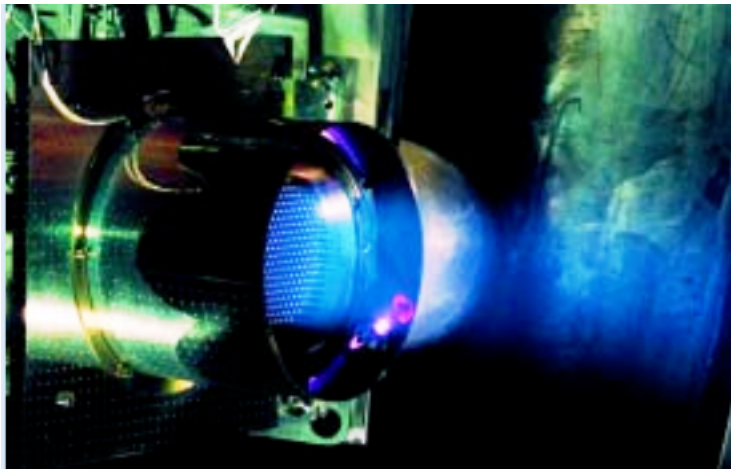


Figure 2.4.: The RIT-10 Gridded Ion Thruster
[11]

to chemical rockets. The thrust acceleration of a low-thrust rocket engine is significantly less compared to that of a chemical rocket engine. This makes low-thrust rocket engines unsuitable for launching a pay load off the earths surface into orbit . Fig 2.4 shows electric grid ion thruster. [11]

In order to lift huge amounts of mass to higher orbits there is a need for very good thrusters or rocket engines, however these rocket engines themselves should not become an hindrance to achieve this goal. By combining chemical rockets with low-thrust it is possible to greatly enhance the future space programs.

2.2.1. Impulsive Mission

The name impulsive mission is because the rocket engines thrust the spacecraft for very small amounts of time (which range from 200 to 500 seconds) compared to mission flight time [9]. Once the spacecraft is thrusted, it is placed on a trajectory to its destination, the spacecraft coasts from one point to another. In case of path corrections, additional impulses are applied. An example trajectory of an impulsive mission for spacecraft moving from low earth orbit to geosynchronous orbit is shown in Fig 2.5.

The minimum energy transfer path between two circular orbits is an elliptical path with perigee (point on elliptical path closest to Earth) on the inner orbit and apogee (point on elliptical path farthest to Earth) on the outer orbit and is known as Hohmann trajectory. The elliptical path between the two circular

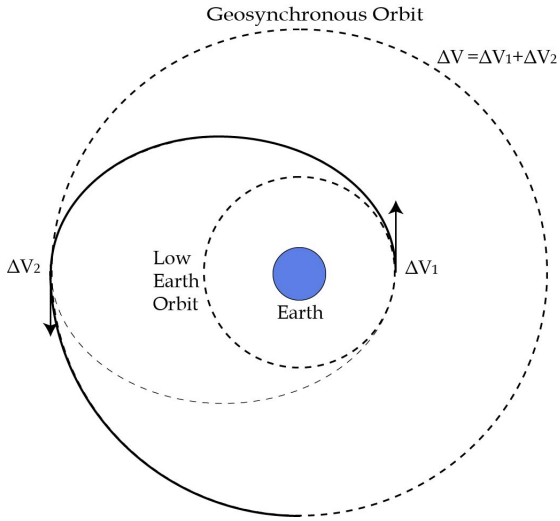


Figure 2.5.: Impulsive Mission Trajectory (Hohmann Transfer) [9]

orbits is known as transfer orbit. The transfer itself involves thrusting of the spacecraft at two different points, once at perigee (ΔV_1 to move the spacecraft from inner circular orbit to transfer orbit) and once at apogee (ΔV_2 to move the spacecraft from transfer orbit to outer circular orbit) as shown in Fig 2.5. The transfer is named after the German scientist Walter Hohmann and is known as **Hohmann Transfer**. [9]

2.2.2. Low-Thrust Mission

Contrasting to impulsive missions, low-thrust missions are thrusted for longer periods of time during the mission. The thrust is applied continuously, sometimes up to few months, making the thrusting time non negligible when compared to mission flight time [11]. Unlike Hohmann transfer as shown in Fig 2.5, in low-thrust mission the spacecraft spirals out from the initial point to final destination, making revolutions around the central body. A low-thrust trajectory for a similar mission from low earth orbit to geosynchronous orbit appears as shown in Fig 2.6 and the $\Delta V_1, \Delta V_2, \Delta V_3, \Delta V_{...}$.....$\Delta V_n$ denotes thrusting of the spacecraft and is shown at regular intervals for easy understanding, however in a real mission scenario the intervals are infinitesimally small. [9]

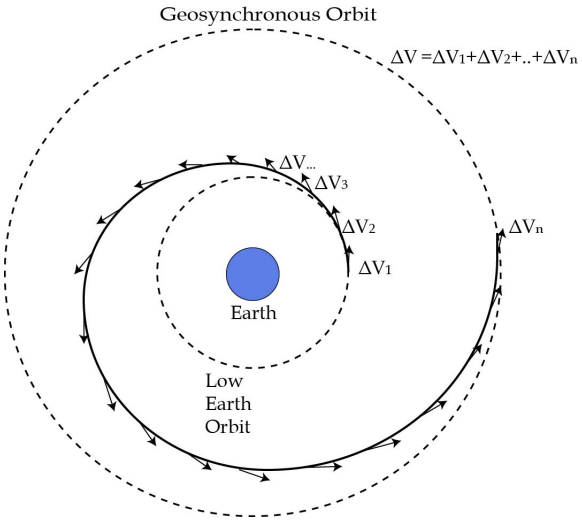


Figure 2.6.: Low thrust mission trajectory
[9]

2.3. Tisserand Criterion

In the previous section, low-thrust missions and impulsive missions are explained. Due to the efficiency of low-thrust trajectories and free energy provided by gravity assists, a combination of both would lead to improvement of overall mission performance. DLR(German Aerospace Center) is currently working on the methods to combine low-thrust trajectories with gravity assist and to include the gravity-assist sequence with in the optimization algorithm. The first attempt to achieve this task is to consider Tisserand Criterion which is used in Impulsive missions for sequencing the gravity-assist partners. Tisserand Criterion is an energy based function for orbital parameters. [25]

$$\frac{R_{\mathrm{pl}}}{a} + 2\sqrt{\frac{a(1-e^2)}{R_{\mathrm{pl}}}}\cos i = constant \tag{2.7}$$

R_{pl} is the solar distance of gravity-assist partner planet. a is semi-major axis of the comet's heliocentric orbit. e is eccentricity of the comet's heliocentric orbit. i is the inclination of the heliocentric orbit.

Equation 2.3 remains constant before and after encounter of a planetary body. Tisserand criterion was developed by Tisserand to study the comets subjected to orbital change due to encounter with Jupiter. Fig 2.7 is an ex-

ample of Tisserand Graph and shows possible heliocentric orbits at possible planetcentric relative energy between spacecraft and planet (given by hyperbolic excess velocity V_∞). Tisserand Graphs are graphical representation of Tisserand Criterion. With the help of Tisserand Graphs it is possible to map, from energy point of view all the orbits a spacecraft can obtain after the encounter with a planet. Gravity-assist sequences for an impulsive mission can be planned a priori with the help of Tisserand Graphs. [25]

2.3.1. Tisserand Graphs and Low-Thrust Missions

In case of low-thrust missions, the thrusting of the spacecraft is continuous and for longer periods of time. Thrusting period is non negligible compared to the over all flight time. Tisserand criterion is a energy based property and when the effect of thrust acceleration is considered in terms of energy basis, then it turns out that the effects are not negligible. This makes an argument that the Tisserand criterion cannot be used for low-thrust gravity-assist sequencing without a correction term. Derivation of the correction term is explained in [25].

Although the correction term exists, Tisserand graphs can not be used for low-thrust gravity assist sequencing as opposed to impulsive missions, due to lack of a priori information on the trajectory. Therefore optimization strategies should be used to find the gravity-assist sequence for low-thrust missions.

2.4. Optimization Algorithms

In previous sections the energy gained from gravity assist is explained, also the efficiency of low-thrust missions. Combining both to obtain energy efficient trajectories leads to a trajectory optimization problem. To solve this optimization problem, some sort of optimization logic is required. Before going through optimization algorithms, general description of optimization problem is explained below.

Optimization is a method of maximizing or minimizing a function or a set of functions to find the best possible solution for a given problem. The goal of optimization is to find the best possible element x, which is a vector of m

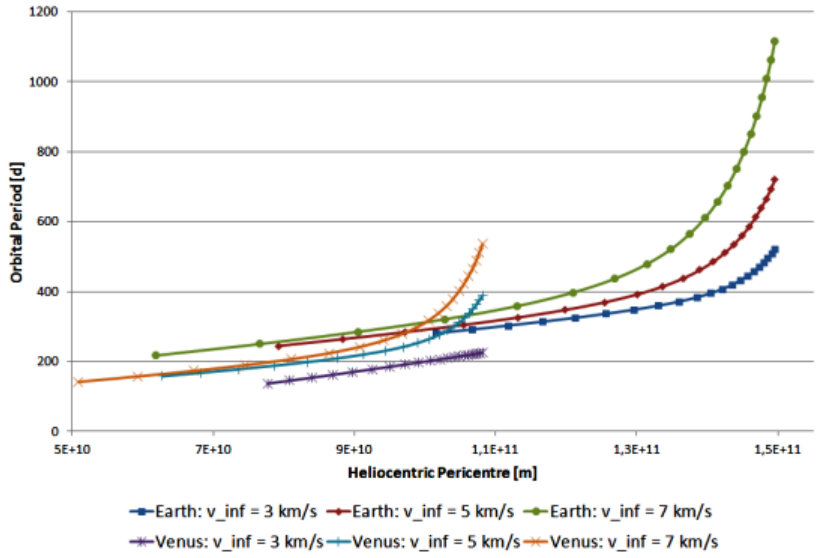


Figure 2.7.: "Example of Tisserand Graphs for Earth (right, note the maximum possible heliocentric pericentre being approx. 1AU for the spacecraft) and Venus (left) for various hyperbolic excess velocities (planetcentric). Orbital period (proportional to the semi-major axis, just like the specific orbit energy) as function of heliocentric peicentre of spacecraft is given."
[25]

decision variables,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \tag{2.8}$$

from a set X according to set of criteria $F = \{f1, f2, f3, ...fn\}$. X is called the problem space and can represent any type of elements like numbers, lists, etc. The functions $f1, f2, ...fn$ are known as the objective functions. [23]

Objective Function : The objective functions could be mathematical functions or complex algorithms which are subjected to optimization (objective function is either minimized or maximized depending on the problem). There could be a single or multiple objective functions for a given optimization problem. [23]

Decision Variables : A set of unknown elements of x (which belongs to problem space X) which influences the value of objective functions are called decision variables. These variables cannot assume any arbitrary values, they are constrained by set of functions, for example $g(x) \geq 0$. The condition $g(x) \geq 0$ is known as **constraint** and the vector x is known as **solution candidate**. [23]

Problem Space : Problem space X of an optimization problem is a set containing all elements x which could be its solution [23].

Search Space : Search space G of an optimization problem is a set of elements g which can be processed by the search operation [23].

Local Maximum A local maximum $x_{\text{lmax}} \in X$ having objective function value $f(x_{\text{lmax}}) \geq f(x)$ for all x neighboring x_{lmax} [23].

Local Minimum : A local minimum $x_{\text{lmin}} \in X$ having objective function value $f(x_{\text{lmin}}) \leq f(x)$ for all x neighboring x_{lmin} [23].

Local Optimum : A local optimum $x_l^* \in X$ is either a local minimum or a local maximum [23].

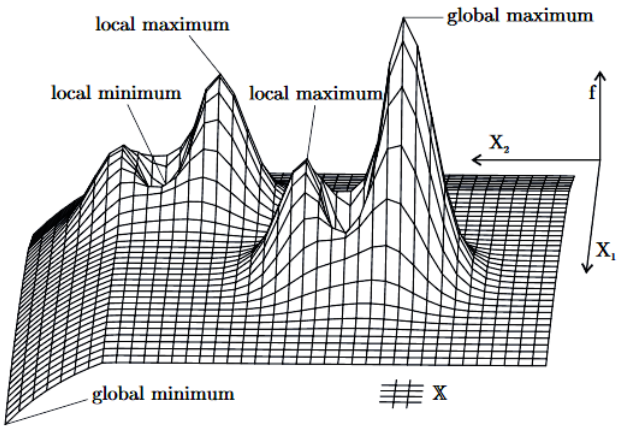


Figure 2.8.: Search space of optimization problem
[23]

Global Maximum A global maximum $x_{\text{gmax}} \in X$ having objective function value $f(x_{\text{gmax}}) \geq f(x)$ for all x present in problem space X [23].

Global Minimum : A global minimum $x_{\text{gmin}} \in X$ having objective function value $f(x_{\text{gmin}}) \leq f(x)$ for all x present in the problem space X [23].

Global Optimum : A global optimum $x_{\text{g}}^* \in X$ is either a global minimum or a global maximum. $x_{\text{g}}^* \in X$ need not be unique, there could be several global optimums or even infinite optimums [23].

Optimal Set : Optimal set $X^* \subset X$ is a set containing all the optimal solution candidates [23].

The global and local optimums for a two dimensional function is shown in Fig 2.8.

The global optimization techniques are the algorithms which are used to find the global optimum for a given optimization problem. Broadly they can be classified as deterministic and probabilistic algorithms.

Deterministic Algorithms : These algorithms are used when a clear relation between the characteristics of possible solutions and their utility for a given problem exists. Search space is efficiently explored by methods such as divide and conquer [23]. Deterministic algorithms guarantee asymptotic convergence

towards the global optimum. The large-scale complex engineering problems would involve discrete and continuous decision variables and turns out to be a NP-hard problem (class of problems which are unlikely to be solved within the amount of time and computational effort bounded by polynomial function [16]) for deterministic methodology. There are several optimization problems where exact methods do not exist or where deterministic methods are too complex to implement. For such complex scenarios probabilistic algorithms are made use of [28].

Probabilistic Algorithms : These algorithms use some kind of randomness or probability with in the definition of the method. Probabilistic algorithms are used when the relation between solution candidate and its fitness function is too complex or the dimensionality of the search space is too high [23]. There is evidence that a NP-hard problem for deterministic methodology can be solved in polynomial time using probabilistic search strategies. Probabilistic algorithms provide probabilistic convergence towards the global optimum. More importantly the quality of global optimum is traded off with the computational time. Probabilistic algorithms provide quick and useful information of an optimization problem. Further, the probabilistic algorithms are classified as instance based methods and model based methods. Instance based methods use the current solution candidate to generate the new solution candidate where as model based methods use sampling distribution to generate new candidates randomly. [28]

The following section explains few of the probabilistic search algorithms in brief.

2.4.1. Hill Climbing

The hill climbing logic is one of the simplest optimization techniques available for problems with single objective function. Fitness of a randomly selected solution candidate is evaluated and the new solution candidate is created using current solution candidate. The new solution candidate replaces the current one if it has better fitness value, else current solution is retained. This procedure is looped and is repeated till the best solution is found. Hill climbing uses current best solution x^* to produce new solution candidate [23].

One of the major issues with Hill climbing logic is, the algorithm cannot escape the local optimum and leads to premature convergence. To overcome the premature convergence issues, it is suggested to randomly restart the search process several times, this method is also known as stochastic hill climbing [23].

Algorithm 1 Hill Climbing Algorithm

Choose an initial configuration
while *Stopping condition is not reached* **do**
Step 1:
Generate new configuration, a perturbation of old configuration
Step 2
if *quality(new configuration) better than quality(old configuration)* **then**
old configuration:= new configuration

2.4.2. Simulated Annealing

Simulated Annealing search algorithm follows the annealing process seen in metallurgy. Similar to hill climbing logic, a random solution candidate is evaluated for its objective function value and new solution candidate is created from ´the current solution candidate. If new solution candidate has better fitness (objective function) value, the solution is accepted as current solution candidate for the next iteration. Unlike Hill Climbing logic the new solution candidates having worse fitness are not discarded straightaway, instead these solution candidates replace the current solution candidate based on a probability function. The probability function depends on current temperature of the system and objective function value of the solution candidate. [23]

The detailed explanation of Simulated Annealing is provided in section 3.9.

2.4.3. Threshold Accepting

Threshold Accepting is a search algorithm provided by Deuck and Scheuer in 1990 [6]. The algorithm appears to be modified version of the Simulated Annealing algorithm, but much simpler without involving the parameters such as temperature, cooling schedule etc. The search begins with a random solution candidate and a new solution candidate is generated from the current solution candidate. If the objective function of the new solution candidate is better than the current solution candidate, then new candidate replaces the current one in the next iteration. If the new solution candidate has objective function value worse than the current solution candidate, then new candidate replaces the current one only if the difference in objective functions is less than a defined threshold value [6].

Algorithm 2 Threshold Accepting for Maximization

choose an initial configuration
choose an initial THRESHOLD T >0
Opt:
create new configuration, a stochastic small perturbation of old configuration
compute $\Delta E := quality(newconfiguration) - quality(oldconfiguration)$
if $\Delta E > -T$ **then**
    old configuration:= new configuration
if *a long time no increase in quality or too many iterations* **then**
    lower THRESHOLD T
if *some time no change in quality anymore* **then**
    Stop
goto *Opt.*

2.4.4. Multi-Start

Several local optimization techniques can efficiently find the local optimum, to diversify the search such a local optimization is randomly restarted at any other point on the search space. Multi-Start method is one in which local search is initiated at a set of multiple random starting points uniformly distributed over the search space. The search looks for several local optimums x_l^* with the view that one of the local optimum turns out to be the global optimum. [18]

Algorithm 3 Multi-Start Algorithm

initialize $i = l$
while *Stopping condition is not reached* **do**
Step 1:
    Generate solution x_i
Step 2
    Apply local search method to improve the solution x_i
    if x_i *improves the best* **then**
        update the best
    $i = i + 1$.

2.4.5. Random Optimization

Random Optimization is the simplest of all algorithms. Originally proposed by Rastrigin, has undergone improvisation by Schumer and Steiglitz, and by

Matyas [14]. In this technique successive solution candidates are chosen at random and is evaluated for its objective function. The search process is repeated till the stoping criterion is satisfied, it may be the required fitness or required number of iterations. Each solution candidate is independent of the solution candidate from previous iteration, i.e search method does make use of the information of already evaluated solution candidates. It converges to the global optimum, however the time required by the algorithm could be astronomical [14].

Here only a handful number of algorithms are discussed, apart from these, there are several other algorithms such as Tabu Search, Ant-colony optimization, Particle Swarm Optimization and whole section of evolutionary algorithms etc.

3. State of the Art

3.1. Introduction

The C++ code for the calculation of the trajectories is provided by V.Maiwald. Specific enhancement of the code is performed to additionally accommodate Simulated Annealing algorithm. Software tool used to compile and execute C++ codes is Microsoft Visual Studio.

Depending on the nature of trajectory modelling, the number of variables in the system of trajectory optimization is decided. With Shape-Based Approximation method (explained in section 3.3) Launch Date, Number of Revolutions, Time of Flight are the values to be set to obtain the values of thrust and position history [4]. A mission being optimized considering gravity-assist sequence would include gravity-assist partner as an additional variable. The variables cannot be set arbitrarily but with the dependency on physical restrictions and scientific observations. There exists interdependency between the variables, for example large number of revolutions requires larger ΔV as longer distances needs to be covered within the specified time. Each variable change greatly influences the performance of the overall mission. For example, flight time suitable for gravity assist at one planet need not be suitable for gravity assist at another planet. There are additional variables such as Turning Angle δ, hyperbolic excess velocity vector V_∞ etc for trajectories involving gravity assists. There exists an optimal combination of these variables along with the variables involved in trajectory approximation that defines optimal mission trajectory. [25]

3.2. Mission Trajectories

The described system of mission trajectories can be seen as a sum of trajectories. A mission is defined as a trajectory between the start body and the end body. This could consist of one or more trajectories depending on the gravity assists. In this thesis work three types of trajectories are considered.

- One leg trajectory (No gravity assist)

- Two leg trajectory (One gravity assist)
- Three leg trajectory (Two gravity assist)

The trajectory from start body to a gravity-assist partner, or from one gravity-assist partner to another, or from gravity-assist partner to end body is known as a trajectory leg. The approximate trajectory is calculated based on the Shape Based Approximation method. The parameters that define each trajectory leg and the entire mission is explained in the following sections.

3.3. Shape Based Approximation

In Chapter 2 the advantages of low-thrust propulsion and energy gained using gravity assists is discussed. Combing these two would be the next step to utilize the advantages of both techniques. Low-thrust propulsion mission is also demonstrated in practice, NASA's Deep Space 1 spacecraft is the first interplanetary solar electric propulsion mission [3]. An optimization tool to easily identify the low-thrust gravity-assist trajectories would be useful for mission designers. The low-thrust mission design requires a method for approximating spacecraft's trajectories. This method could be used to prune large search space or for providing good initial guess to the trajectory optimizers. This is the starting point for trajectory optimization. A shape based method to determine low-thrust trajectories when combined with optimization algorithm to optimize the free parameters (such as Launch Date, Time of Flight etc) will possibly lead to optimal or near optimal trajectories. [4]

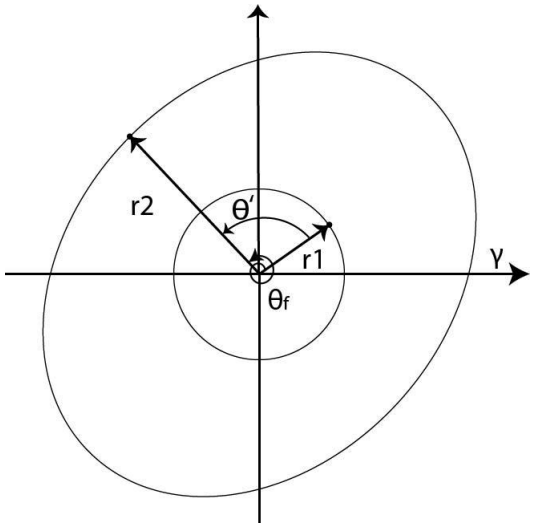
This method approximates the low-thrust trajectory with a function parameterized in terms of polar coordinate θ, i.e $r(\theta)$. Fig 3.1 shows a low-thrust trajectory defined in terms of $r(\theta)$.

To obtain a function, parametric data from a known optimal low-thrust rendezvous trajectory is fed into a curve fitting algorithm that tests hundreds of function and returns the function that best fits the data. The polynomial function obtained is shown in equation 3.1. [4]

$$r = \frac{1}{a + b\theta + c\theta^2 + d\theta^3 + e\theta^4 + f\theta^5 + g\theta^6} \tag{3.1}$$

The detailed version of the solution for the coefficients is provided by Wall and Conway in [4].

Using this method the only unknowns are Launch Date, Flight Time, and Number of Revolutions. To locate the optimal parameter values of a trajectory, global optimization algorithms may be used. The equations are integrated



[4]

Figure 3.1.: Low-thrust orbit transfer

using hundred equal segments and the trapezoidal rule [4]. Fig 2.6 shows thrust vectors along the trajectory, in total there are hundred such vectors which are associated with each of the hundred segments that form individual trajectory leg.

3.4. Mission Parameters

Mission parameters are the decision variables for the trajectory optimization problem. The parameters within the mission not only influences the mission but also the individual trajectories. The parameters can be seen as two set of parameters. One set associated with the whole mission and the other associated with the individual trajectories.

Tab 3.1 is also the structure of one leg mission trajectory, the two leg and three leg missions will have additional trajectory data.

Global variables are those which influence the entire mission. The global variables are also the decision variables for this optimization problem.

- Global variables : The variables associated with the entire mission.
  - * **Start Body** : Starting body of the mission.
  - * **End Body** : Destination of the mission.

One Leg Mission

Start Body	Mission Variables
End Body	
Launch Date	
Time of Flight	
Number of Revolutions	
Gravity-Assist Partner	
Mission Fitness	
Mission ΔV	
Start Body	Trajectory Leg Variables
End Body	
Launch Date	
Number of Revolutions	
Coefficient a,b,c,d,e,f,g	
V_inf_arrival	
V_inf_departure	
R_peri_flyby	
Turning Angle(δ)	
Thrust History	
Position History	
Trajectory Fitness	
Trajectory ΔV	

Table 3.1.: Parameters of the Mission and the Trajectory with in the Mission

Two Leg Mission

Mission Variables
Trajectory leg one
Trajectory leg two

Table 3.2.: Structure of two leg mission

Three Leg Mission

Mission Variables
Trajectory leg one
Trajectory leg two
Trajectory leg three

Table 3.3.: Structure of three leg mission

  * **Launch Date** : The Julian modified date [1] on which the mission is set to launch.
  * **Time of Flight** : Number of days set for the total mission.
  * **Number of Revolutions** : Number of revolutions of the spacecraft around the sun.
  * **Gravity-assist partner id** : Identification number indicating the planet at which the gravity assist occurs.

- Trajectory leg variables : The variables associated with each leg or individual trajectory between planet A to planet B. These planets need not be start body and end body of the mission.
  * **Start Body** : Starting body of the trajectory leg.
  * **End Body** : End body of the trajectory leg.
  * **Launch Date** : Launch date of the trajectory leg.
  * **Time of Flight** : Time of flight of individual trajectory.
  * **Number of Revolutions** : Number of revolutions of the spacecraft around the sun.
  * **Coefficient a,b,c,d,e,f,g** : Trajectory coefficients derived from Shape Based method.
  * **V_inf_arrival** : Inbound hyperbolic excess velocity.
  * **V_inf_departure** : Outbound hyperbolic excess velocity.
  * **R_peri_flyby** : Distance from the planet where the spacecraft executes a gravity assist.
  * **Turning Angle (δ)** : Angle at which the inbound hyperbolic excess velocity vector ($V_{\infty\,\mathrm{PI}}$) rotated after a gravity assist.

Apart from the above mentioned parameters, there are parameters which determine the quality of the solution obtained. These are:

- ΔV : The total velocity change required for the spacecraft to complete the mission.
- **Mission Fitness** : Cost function which evaluates the quality of the solution.

[1] Modified Julian Day is a dating method used by astronomers, geophysicists and others to have unambiguous dating system. [1]

- ΔV : The total velocity change required for the spacecraft to complete the trajectory.
- **Trajectory Fitness** : Cost function which evaluates the quality of the trajectory.

3.5. Objective Function or Cost Function

As already explained in Chapter 2, goal of optimization algorithm is to minimize or maximize the objective function. The objective function here considers minimizing the ΔV requirement of the trajectories. The objective function definition is given by equation 3.2. [26]

$$J = k\frac{1}{\sum_n \Delta v'} \tag{3.2}$$

where k is the weighting factor that allows scaling of the fitness depending on the ΔV performance. Index n counts through number of segments for a given solution candidate. Objective function considers the fitness in global perspective, i.e the ΔV summation informs about the entire mission and not just the individual trajectories. Combination of optimal individual trajectories need not guarantee an optimal mission trajectory.

3.5.1. Violation of Constraints

The objective function is modified to penalize the solution candidate fitness for violating the constraints rather than discarding the solutions. For the violation of minimum allowable radius the objective function is

$$J^* = J\frac{R_{\text{min,is}}}{2.R_{\text{min,allowed}}} \tag{3.3}$$

where $R_{\text{min,is}}$ is the actual minimum radius and $R_{\text{min,allowed}}$ is the minimum allowed radius, J is the fitness value determined according to equation 3.2 and J^* is the penalized fitness. [26]

Similarly for violation of maximum allowable thrust, the fitness function is changed as shown below.

$$J^* = J\frac{T_{\text{max,allowed}}}{2.T_{\text{max,is}}} \tag{3.4}$$

where $T_{\text{max,is}}$ is the actual thrust value and $T_{\text{max,allowed}}$ is the maximum allowed thrust value. [26]

3.6. Mission Settings

The mission settings are the values which are set by the mission analyst to evaluate a particular mission. Few parameters, such as start body and end body are fixed, i.e the optimizer does not change these values throughout the optimization process. Few parameters such as flight time, gravity-assist partner id have the boundary value defined and the optimizer can chose any value that is bounded within these boundary conditions.

Tab 3.4 shows the general mission settings used in this thesis work for all the calculations.

3.7. Problem in Hand

Correlating to the general optimization problem described in section 2.4 it is clear that the mission parameters explained in section 3.4 are the decision variables and also form the solution candidates for the low-thrust gravity-assist trajectory optimization. Cost function explained in section 3.5 is the objective function of the optimization problem and the mission settings along with the thrust value and *Rmin* restrictions correspond to the constraints. The problem in hand is a constrained optimization problem.

3.8. Choice of Optimization Algorithm

Optimization algorithms are chosen based on complexity of implementation and with the plausibility of obtaining quick results and useful information on the search space topography. This is important because there is no prior knowledge of the topography of the search space for the defined problem. There are no previous trajectory optimization methodologies including the gravity-assist sequencing along with the optimization to be considered for the choice of optimization algorithm. Optimization algorithm should work with both continuous and discrete variables. Optimization technique should not depend on the search space topography. Algorithm should move in direction of global optimum or should provide near optimal results. Taking all the above factors into consideration two different optimization strategies are selected.

3.8.1. Random Optimization

Random optimization is one of the simplest optimization algorithms in terms of implementation. It also provides quick and rough overview of the search space

Mission Settings			
Parameter	**Value**	**Units**	**Explanation**
start_body	Earth	-	Start body of the mission
end_body	Jupiter	-	End body of the mission
r_min	0.25	AU	Minimum allowable distance from system barycenter for trajectory
launch_date	56,000	Days	Modified Julian Date
launch_window	100	Days	Number of days within which the launch can happen.
flight_time_max	3,000	Days	Maximum allowed flight time for the mission.
flight_time_min	1,000	Days	Minimum allowed flight time for the mission.
accuracy	0.0001	Days	Relative difference between the set flight time and calculated flight time.
max_thrust	0.00015	(m/s^2)	Maximum allowed thrust acceleration.
nrev_max	6	-	Maximum number of revolutions
step_no	100	-	Number of steps for the integral of the flight time
fitness_weight	60,000	-	Weighting of ΔV over Flight Time, the larger, the more important is ΔV.
v_inf_start	1,000	m/s	Value of hyperbolic excess velocity to start with.
v_inf_end	50	m/s	Value of hyperbolic excess velocity to end with.
N_GA_max	2	-	Maximum number of gravity-assists.

Table 3.4.: General mission settings used in this thesis work

topography. Random optimization technique searches randomly over different areas of the search space ensuring diversity.

3.8.2. Simulated Annealing

The choice of Simulated Annealing is influenced by the analysis of the problem using Random Search (explained in detail in Chapter 4). Random Search algorithm hints towards a complex multi-modal search space. Such a complex search space can be successfully explored by Simulated Annealing algorithm. Simulated Annealing does not require the knowledge of mathematical model of the problem and is suitable for the problem as long as the perturb and evaluate functions can be implemented. Model of the system can be treated as black box model, i.e for a certain input, certain output is obtained without actually investigating into the details of the system. But one of the major issues with Simulated Annealing would be the large time consumption. [16] [20]

After making a choice on optimization algorithm next step is to go through the basics of the algorithm.

3.9. Basics of Simulated Annealing

Simulated Annealing is a general probabilistic search algorithm introduced in 1983 by Cerny and Kirkpatrik in order to solve combinatorial optimization problems [16]. Simulated Annealing means artificially implementing the process of annealing. In metallurgy, annealing is a process of heat treating a material to change its properties. A solid material is heated to a very high temperature at which the material liquifies and molecules randomly arrange themselves, followed by slow cooling gradually lowering the temperature of the heat bath. In this way the particles of the material arrange themselves in low energy state of corresponding lattice leading to a stable and strong solid material. The initial temperature and the process of cooling would determine the quality of the solid material. The Simulated Annealing algorithm is very similar to this physical process. [10]

While solving an optimization problem using Simulated Annealing algorithm, the quality of the solid material would represent the solution of the problem, random movement of particles would represent the perturbation of the input parameters, temperature would be the parameter influencing if the perturbed parameters and the new solutions corresponding to them have to be accepted. The acceptance of the new solution is based on a probability, solution with better fitness (cost function) is accepted but the solution with lesser fitness is probabilistically accepted (3.9.2). [10]

3.9.1. Neighborhood

Implementation of Simulated Annealing algorithm requires definition of configurations. This can be formalized as (R,C) where R is finite set of configurations (configuration space or input parameter space) and C is a cost function associated with corresponding configuration. The perturbation mechanism defines the neighborhood configuration for R_i consisting all configurations that can be reached from R_i. [16]

3.9.2. Acceptance Probability

Simulated Annealing algorithm does not show tendency to end up in a local minima or local maxima because the new solution acceptance or rejection is characterized by a probability function [10].

$$P(\Delta E) = \begin{cases} e^{\frac{-\Delta E}{k_B T}}, & \text{if } \Delta E<0 \\ 1, & \text{otherwise} \end{cases} \tag{3.5}$$

where:

$P(\Delta E)$ = Probability of acceptance of new solution.
ΔE = Difference of the cost function between the current and the new state.
k_B = Boltzman constant.
T = Current Temperature.

Acceptance Logic Once the new state solution is created, i.e the solution by perturbing the previous solution or by moving to the neighboring state configuration, the quality of the solution (cost function or solution fitness) will be compared with the quality of the current state solution. If the quality is better than the current state solution, i.e $\Delta E \geq 0$, then the neighboring state solution is definitely accepted, probability is 1 for such a scenario as shown in equation 3.5.

If the neighbor state solution is of lesser quality, i.e $\Delta E < 0$ then the acceptance probability depends on ΔE and T. If the value of $\Delta E < 0$ then the probability is inversely proportional to the magnitude of ΔE. There also exists a control parameter (temperature), acceptance probability is directly proportional to temperature and reduces gradually with the temperature. At high temperatures the acceptance probability would be high, giving enough chance for the algorithm to accept lower quality solutions and escape from the local minima or maxima and reach the global optimum. [20]

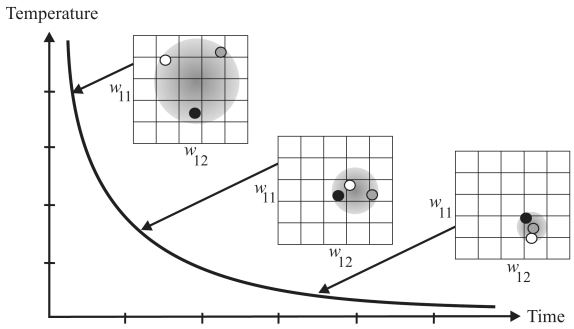


Figure 3.2.: Method of Simulated Annealing [10]

- Simulated Annealing search has two search phases, global search phase and local search phase. Initially at higher temperatures, algorithm can explore the search space erratically, this mimics the global search phase. As the temperature is gradually reduced, the acceptance probability is gradually lowered and only good solutions are accepted making the algorithm tend towards Hill Climbing logic (local search phase).
- If the new solution is extremely bad, then the probability to move to such a state is equally less.

The logic of this algorithm can be visualised as shown in Fig 3.2.

3.9.3. Cooling Schedule

Cooling schedule describes how the control parameter (temperature) is reduced during the optimization. Reduction of temperature has major influence on the performance of the algorithm. The temperature will be set to initial temperature which will be greater than zero and reaches zero at the end of optimization, i.e for fixed number of iterations or when required fitness is obtained. [27]

There exists wide range of temperature scheduling methods, some of them are reducing the temperature linearly, exponentially, or in logarithmical manner etc. Few of the temperature reduction methods are shown in the below equations [27].

Equation 6.1 shows the exponential reduction of temperature [27].

$$T(t) = T_0 \alpha^t \tag{3.6}$$

where :

$T(t)$ = Actual temperature value.
T_0 = Initial temperature.
α = Constant factor to repeatedly lower the temperature ($0 < \alpha < 1$).
t = Time, which is also the step count.

Equation 6.2 shows the linear reduction of the temperature [27].

$$T(t) = T_0 - \eta t \tag{3.7}$$

where :

T(t) = Actual temperature value
T_0 = Initial temperature.
η = Constant factor to repeatedly lower the temperature ($0 < \eta < 1$).
t = Time, which is also the step count.

Equation 3.8 shows the logarithmic reduction of the temperature introduced by Geman and Geman [22].

$$T(t) \geq \frac{c}{log(1+t)} \tag{3.8}$$

where :

T(t) = Actual temperature value
c = Generally considered largest energy barrier in the problem.
t = Time, which is also the step count.

However this method of temperature reduction is considered extremely slow for practical purposes [27].

There are several other temperature reduction techniques which are used in Simulated Annealing algorithm.

3.9.4. Epoch Length

Epoch length is the parameter which determines the number of iterations or the amount of time a system spends in a given temperature level during optimization. The value of epoch length can be set proportional to size of problem instance or can be set proportional to neighborhood solutions of a given solution. It can also be set proportional to the total number of iterations. [12]

Algorithm 4 Simulated Annealing Algorithm (for maximization)

Initialize the system configuration
Randomize the initial input vector X(0), current state
Initialize temperature T *with a large value* T_0
while *stopping condition is not reached* **do**
  while *Epoch length* $\neq$ *0* **do**
    Create new state, randomly perturbing current state, $x = x + \Delta\ x$
    Evaluate change in cost function, ΔE= *E(x+*$\Delta\ x$*) - E(x)*
    if $\Delta E \geq 0$ **then**
      current state = new state
    else
      current state = new state, with the probability $P = e^{\frac{-\Delta E}{k_B T}}$
    Reduce Epoch length, $L = L - 1$
  Reset Epoch length
  Set $T = T_0$ - ΔT

3.9.5. Simulated Annealing Algorithm

3.10. Overview

The over all system is represented by a simple block diagram shown in Fig 3.3.

The mission input parameters are set in accordance with the restrictions defined in the mission settings, for example the maximum Number of Revolutions, maximum Time of Flight etc. Mission input parameters are fed to Shape-Based Trajectory Approximation method which calculates a trajectory for the set input parameters. Each trajectory and the mission is associated with ΔV value and fitness value (which are fed back to the Optimization block). The optimization algorithm uses the fitness information to adapt the mission input parameters. The number of executions of this loop is decided by the free parameters of the Optimization block.

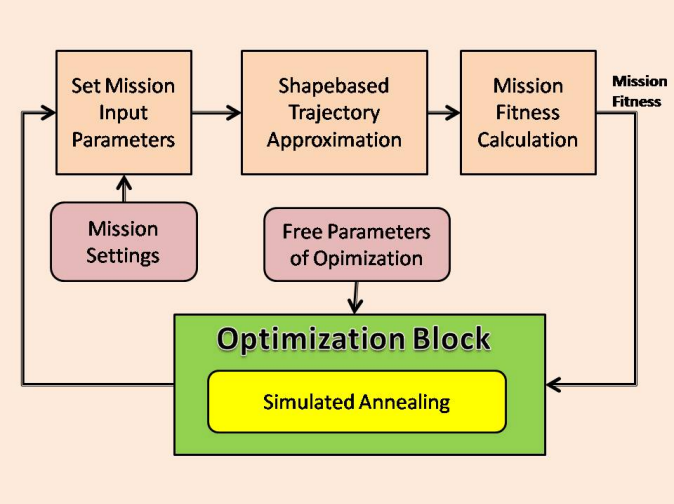


Figure 3.3.: Block Diagram of the Overall System

4. Random Search Analysis

4.1. Introduction

Low-thrust gravity-assist trajectory optimization is a complex optimization problem, the simplest approach to find the optimal trajectories is to use Random Search algorithm. The aim here is to try and find a good solution or a set of good solutions randomly, i.e randomly generated inputs are given to the system, obtained solutions are then sorted to find the best solution. The inputs are generated independently, i.e the new solution candidate do not depend on the previous solution candidate. The basic idea behind this process is to explore the search space randomly and to obtain a fair amount of knowledge on the search space, this information can be used by other search algorithms. Expectation in this search method is to hit one of the local maximum or the global maximum, however the chances of hitting the global optimum could be very small.

4.2. Bench marking

Random Search is one of the simplest optimization algorithm. Evaluating the quality of the results obtained in Random Search method actually sets the minimum quality level or minimum fitness level of the solutions for further searches. In order to implement any other search technique and perform the analysis, the search technique has to deliver better set of results compared to that of Random Search approach. Therefore results from Random Search approach is treated as a bench mark for other search methods.

4.3. Solution Approach

The free parameter of the search algorithm is the number of solutions considered at each execution, or in simple terms population size. The chances of finding a global optimum is higher with larger population exploring the search space at each execution. The technique appears to be similar to that of brute force technique, requiring more force for better set of results.

Random search	
Execution	Population (number of samples)
1	15,000
2	30,000
3	50,000
4	75,000
5	100,000

Table 4.1.: Parameter variation of random search technique

For the calculations both mission settings and the algorithm settings are to be considered. The mission settings are set as shown in the Tab 3.4 and search algorithm parameter (population or number of solutions) is set as shown in Tab 4.1.

Steps for the calculation :

- Set the mission settings with the values given in the Tab 3.4.
- Set the population size of Random Search to 15,000.
- Obtain the solutions and sort for the best, for all three scenarios, one leg, two leg, three leg.
- Average the results and plot the graphs.
- Vary the population size as shown in Tab 4.1 and repeat the experiments.

Note : 20 executions are considered for each population size (parameter setting) to obtain consistency in the results.
Note : The C++ program is coded such that it provides trajectory solutions for all the three cases: no gravity-assist (one leg solution), one gravity-assist (two leg solution), and two gravity-assist (three leg solution) for the same mission settings.
Note : All calculations are performed for the mission from Earth to Jupiter.

The goal of the optimization tool is to optimize the low-thrust gravity-assist trajectories with in a time frame of 24 hours. A Random Search calculation for population size of 100,000 on a average needed approximately 63453 seconds.
Note : This value is only a fair approximation and not accurate measurement, because the code was simultaneously executed on several

machines. Several factors influence the run time, speed of the processor, efficiency of the code etc. Due to this limitation, Random Search analysis is confined to a maximum population size of 100,000.

4.4. Results

Various parameters such as ΔV, Solution Fitness, Time of Flight, Launch Date, Gravity-assist partner are analysed and few best results by Random Search method are tabulated. The plots reveal how the values of various parameters alter with increase in the population size (free parameter of Random Search).

4.4.1. ΔV results

One leg missions

Fig 4.1 shows the plot of ΔV for one leg solutions against the population size. Even though the graph appears to be sloping downwards with increase in the population size, there is no much difference between the average ΔV values of the solutions examined at the population sizes 15,000 and 100,000. The maximum difference in the average ΔV values is found to be $(1.6090 * 10^4 - 1.6056 * 10^4 = 34m/s)$ which is negligible when compared to average value of $16km/s$. The standard deviations at population size 100,000 is the least $(4m/s)$ clarifying that most of the solutions have similar ΔV.

Two leg missions

Fig 4.2 shows the plot of average ΔV for two leg solutions against the population. The average ΔV appears to decrease with increase in the population. The interesting observation is that the reduction in average ΔV between the smaller population sizes (15,000, 30,000) is more compared to that at the larger population sizes (75,000,100,000). However it is also seen that there is a significant amount of average ΔV reduction with the population of 100,000 compared to population size of 15,000 $(1.8800 * 10^4 - 1.7000 * 10^4 = 1.88km/s)$, 10% improvement on the average ΔV. The graph also tends towards saturation as the improvement of average ΔV between the population sizes of 75,000 and 100,000 is considerably less $(1.7020 * 10^4 - 1.7000 * 10^4 = 20m/s)$. The standard deviation also reduces in a similar fashion to that of average ΔV.

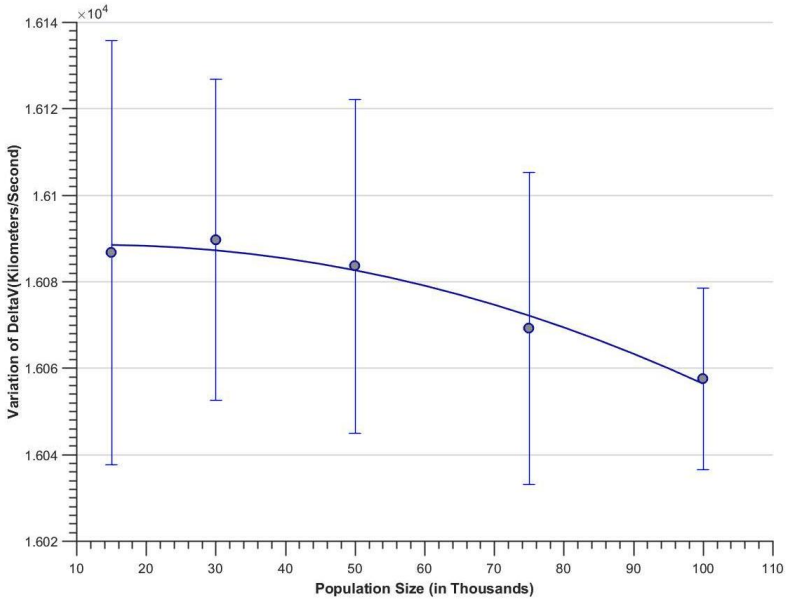


Figure 4.1.: Plot of average ΔV for one leg missions against variation of population size

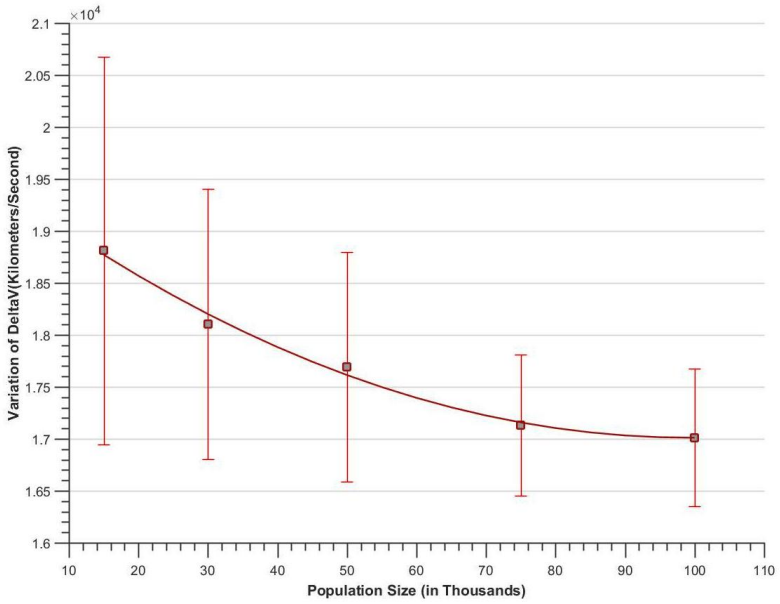


Figure 4.2.: Plot of average ΔV for two leg missions against the variation of population size

Three leg missions

Fig 4.3 shows the graph of average ΔV for three leg solutions against the population size. Considerable improvement in the average ΔV for population of 100,000 compared to that of 15,000 ($3.7000 * 10^4 - 2.9000 * 10^4 = 8km/s$) is observed and the improvement is greatest among all the three scenarios of one leg, two leg and three leg missions. Even with appreciable improvement, the average ΔV of three leg solution is larger than two leg solutions, approximately by $16km/s$. Population size of 100,000 is not enough to find a solution whose average ΔV is lesser than $29km/s$. The standard deviation of ΔV´values are irregular with the maximum of $19km/s$ at population size of 30,000 and minimum of $9.5km/s$ at population size of 75,000.

Fig 4.4 shows the comparison between the one, two, three leg solutions.

This gives an average ΔV expectation from any other search algorithm, i.e the solutions obtained by any other search algorithm should have an average ΔV better than the one presented in Fig 4.4 for one leg, two leg, and three leg missions respectively. It is extremely difficult to find the three leg missions with ΔV values close to one leg and two leg missions.

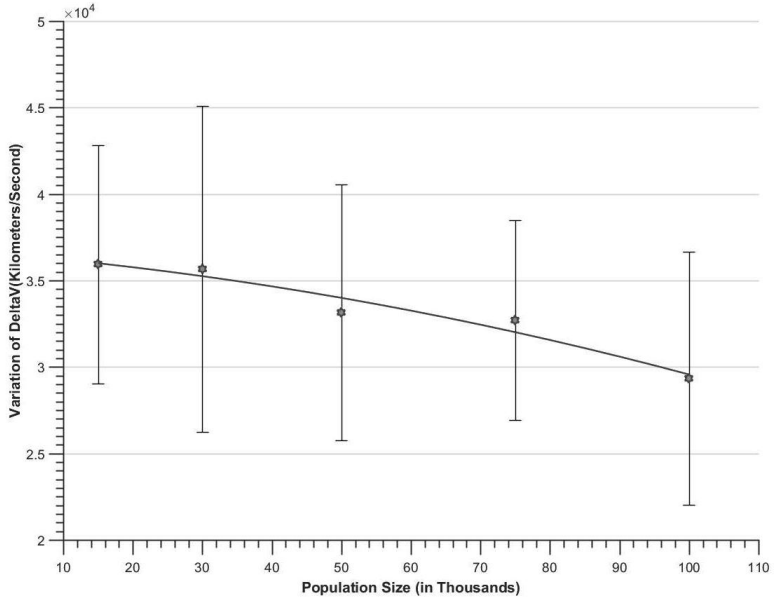


Figure 4.3.: Plot of average ΔV for two leg missions against the variation of population size

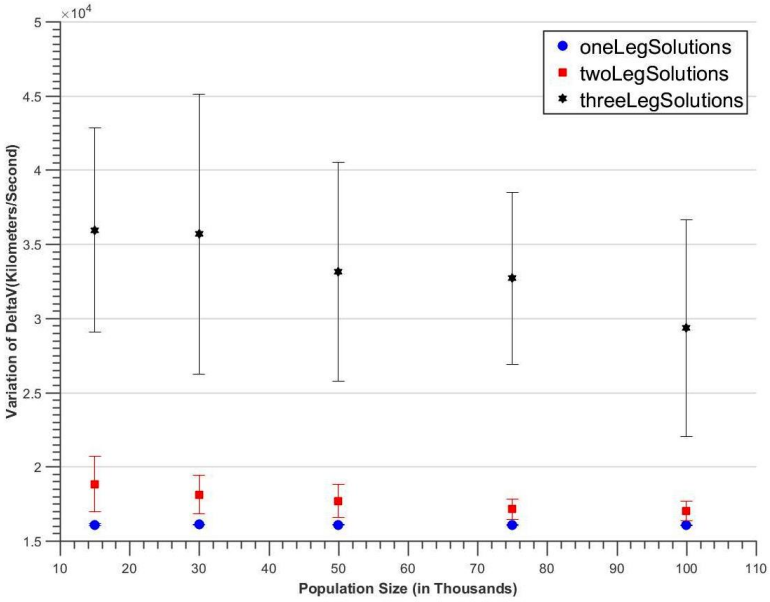


Figure 4.4.: Comparison of average ΔV of one,two and three leg missions

4.4.2. Solution Fitness

The fitness of the solution defines how good or suitable the solution is. It is not necessary that the solution with lesser ΔV should always have higher fitness value, because the solutions which violate the boundary conditions are penalized on fitness (explained in section 3.2). There could be solutions with very good ΔV but they might have violated the constraints (for example, maximum allowable thrust value). The solutions which violate the constraints are not discarded, instead these solutions are penalized and retained in the optimization process. The search is to find solution with optimum fitness and also very less ΔV values. Fig 4.5 shows the comparison between the average solution fitness values of the one, two and three leg missions.

A similar behaviour in the fitness graphs to that of ΔV graphs in the previous section, is observed. There is no significant difference in the solution fitness for one leg missions at population size 15,000 and at 100,000. This shows that it is very easy for Random Search to find the good solutions for one leg missions.

Solution fitness for two leg missions show a gradual growth with the increased population size.

Three leg mission solutions do show greater improvement, but it is also accompanied by the fact that the average solution fitness values of three leg missions are considerably less compared to the one leg and two leg missions. This is shown in Fig 4.5. Maximum fitness values for all one, two and three leg missions are found at population size of 100,000.

4.4.3. Gravity-Assist Sequencing

In this section the gravity-assist partners are considered with variation of the population size for both two leg and three leg solutions. The plots are considered to visualize the preferred gravity-assist partner/sequence for the solutions.

From Fig 4.6 it is seen that with the increase in the population size the selected gravity-assist partner tends to be Mars. This shows that the best solutions always had Mars as the gravity-assist partner. It can also be noticed that with population size of 15,000 and 30,000 there are 6 solutions and 1 solution with gravity-assist partner as Earth respectively.

A similar analysis with three leg solutions reveals the gravity-assist sequence. Fig 4.7, Fig 4.8, Fig 4.9, 4.10 and Fig 4.11 show the plot of gravity-assist sequence for the solutions with population size 15,000, 30,000, 50,000, 75,000 and 100,000 respectively.

Mars-Mars and Earth-Earth are the stand out sequences for the solutions with population size of 15,000. With increase in population size, Earth-Earth

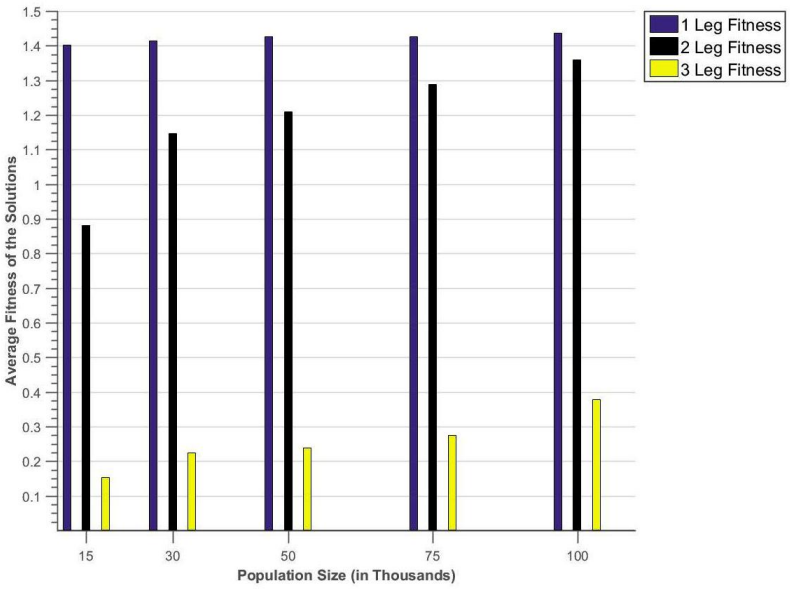


Figure 4.5.: Comparison of average solution fitness of one,two and three leg missions

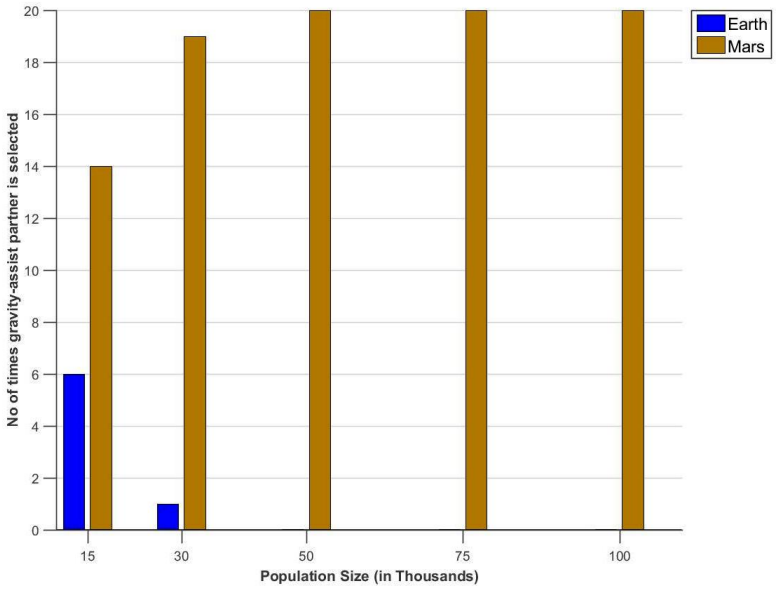


Figure 4.6.: Plot of gravity assist partners for two leg solutions against population size of random search technique

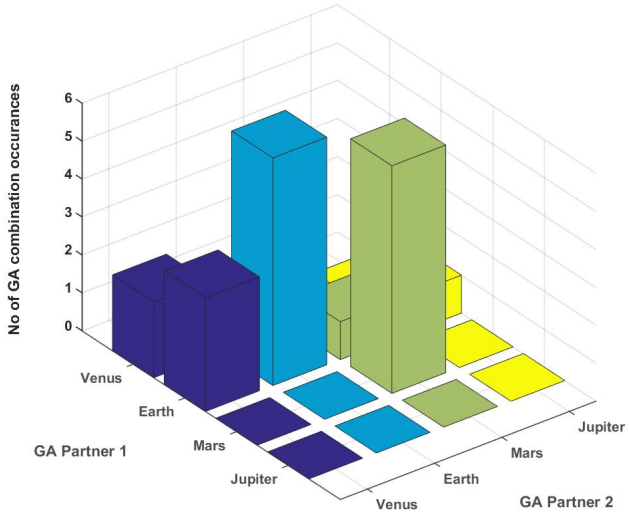


Figure 4.7.: Gravity-assist partner selection for three leg missions at population size 15,000

sequence is favoured.

4.4.4. Distribution of Launch Dates

Fig 4.12 shows the parameter Launch Date selected for the one leg mission solutions. The lower and upper boundary for the launch dates is set to be 56,000 and 56,100 respectively (also shown in Tab 3.6). 75% of solutions (16 of 20 solutions) did show that the launch dates are clustered between 56,060 and 56,080 and the behaviour remained similar with the increased population size. But no such behaviour is observed in the two leg and three leg mission solution launch dates as shown in Fig 4.13 and in Fig 4.14 respectively. The launch dates are evenly distributed throughout the parameter space, indicating the increase in complexity of the solution space. Random Search could not find the solutions having the Launch Date parameter clustered at one region of parameter space, i.e similar solutions are not present over the search space unlike one leg solutions.

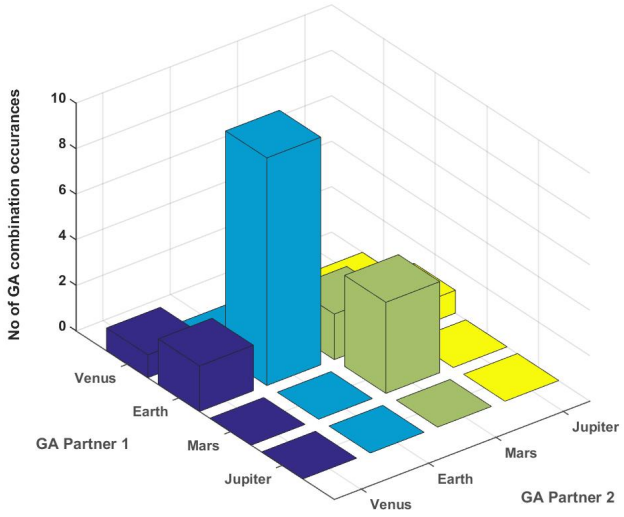


Figure 4.8.: Gravity-assist partner selection for three leg missions at population size 30,000

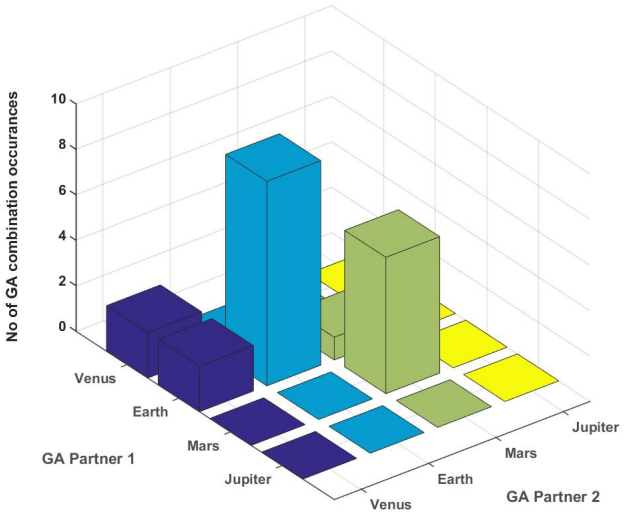


Figure 4.9.: Gravity-assist partner selection for three leg missions at population size 50,000

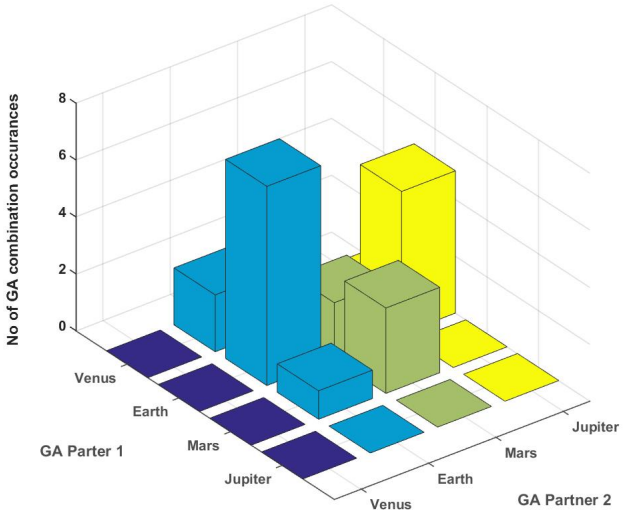


Figure 4.10.: Gravity-assist partner selection for three leg missions at population size 75,000

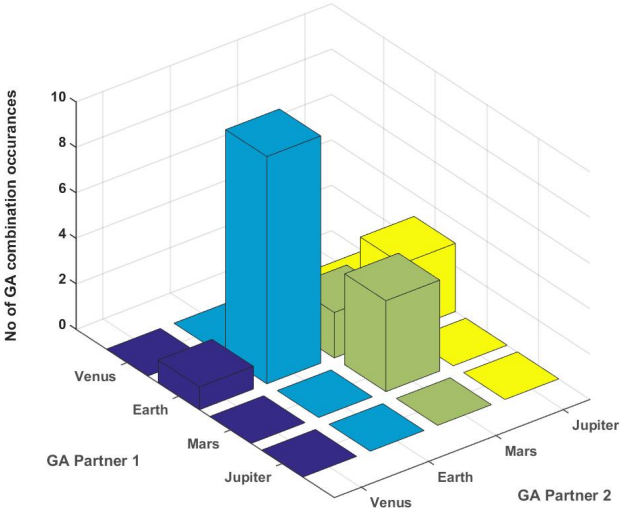


Figure 4.11.: Gravity-assist partner selection for three leg missions at population size 100,000

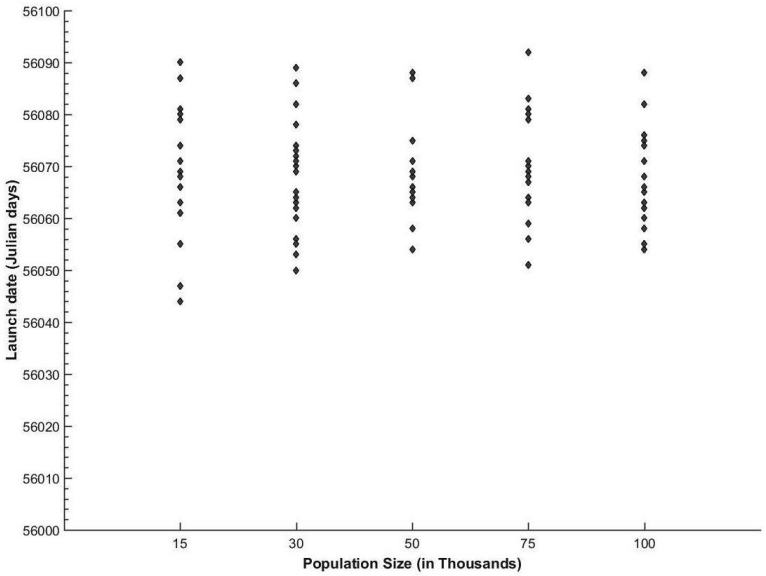


Figure 4.12.: Launch Date distribution for one leg missions

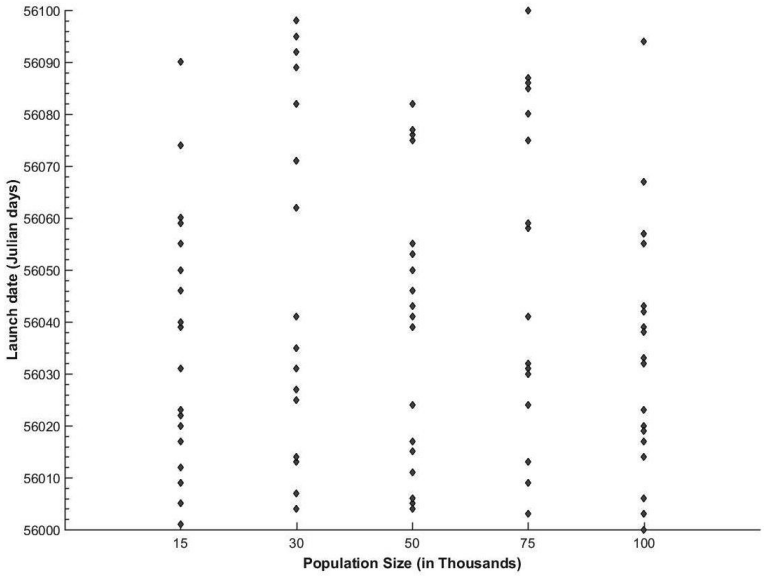


Figure 4.13.: Launch Date distribution for two leg missions

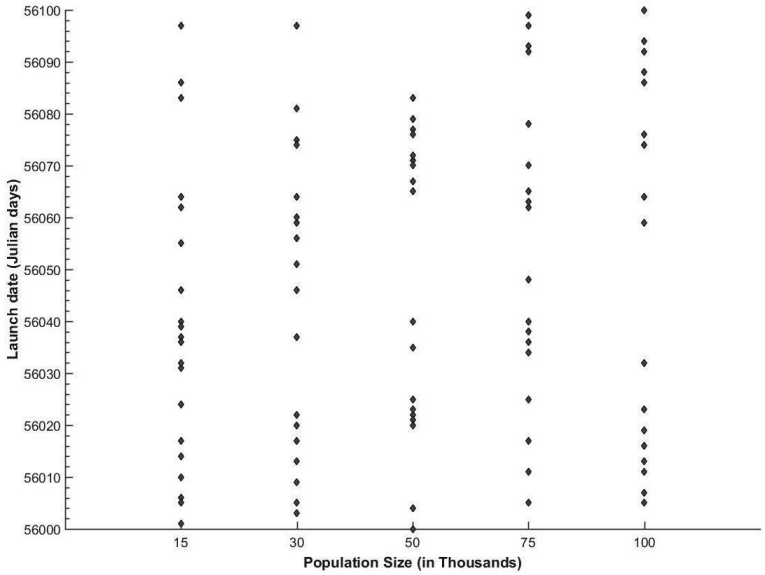


Figure 4.14.: Launch Date distribution for three leg missions

4.4.5. Distribution of Time of Flight

Time of Flight is another global variable which influences the quality of the trajectory and solution fitness values, hence the distribution of Time of Flight over the parameter space is considered. The upper and lower boundaries of the flight times is set to 1,000 and 3,000 days respectively as shown in Tab 3.4. Fig 4.15 shows the flight times for one leg mission solutions. The flight times are clustered at the higher values of the flight time and at one particular region of the parameter space. 89% of solutions flight times lie between 2980 days to 3000 days.

The distribution of flight times for two leg missions is clustered between 2500 to 3000 days as shown in Fig 4.16, which included 98% of solutions. This indicates that with gravity assist the spacecraft would require lesser time to complete the mission. A more even distribution of flight times is observed for three leg mission solution as shown in Fig 4.17. The flight times are evenly spread out between 2000 and 3000 days except for one solution.

One common observation is made from all the parameter plots, the parameters for one leg mission converge quickly compared to that of two and three leg solutions.

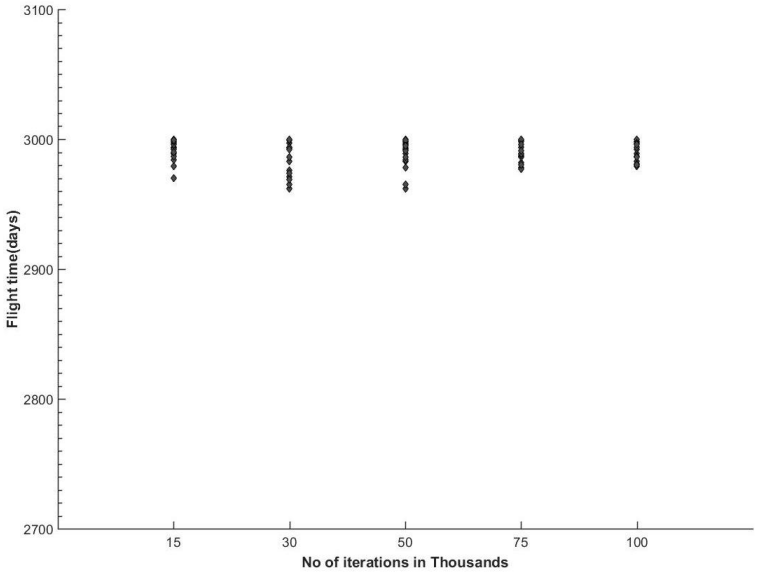


Figure 4.15.: Distribution of Time of Flight for one leg missions

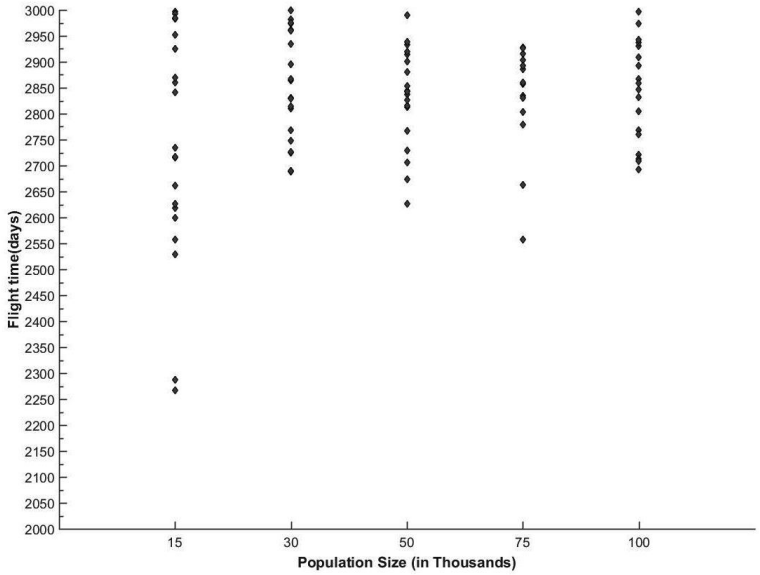


Figure 4.16.: Distribution of Time of Flight for two leg missions

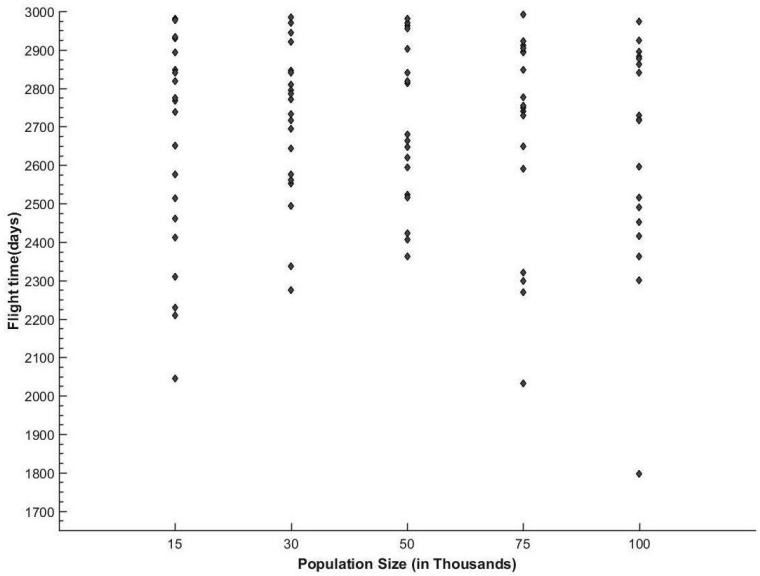


Figure 4.17.: Distribution of Time of Flight for three leg missions

4.4.6. Best Results

Tab 4.2 contains the best results of ΔV and solution fitness obtained in the Random Search analysis.

4.5. Discussion

4.5.1. ΔV Values

One leg missions : Given the result that improvement in one leg solutions with increase in population size is minimal, shows that the similar solutions for one leg missions are present abundantly over the search space. However this also indicates that better solutions, for example ΔV less than $14.827 km/s$ (best solution present in Tab 4.2) are difficult to be found, giving an overview that near optimal solutions are sporadically spread on the search space. ΔV results hint that the search space for one leg solutions contain plateaus (flat regions of solution space).

Average ΔV for one leg solutions remained nearly constant for all population sizes. Therefore usage of lesser population size to obtain quicker results would be more beneficial in case of one leg mission scenario.

Best results for Random Search		
One Leg Solution Results		
Population Size	**Best ΔV (km/s)**	**Best Fitness**
15,000	16030.004243625563	1.4708311393602898
30,000	16024.966418826680	1.4603690764626747
50,000	16014.517032525131	1.4688988247562682
75,000	16029.624328665328	1.4602607387382092
100,000	16032.828217764461	1.4627058216992459
Two Leg Solution Results		
Population Size	**Best ΔV (km/s)**	**Best Fitness**
15,000	14827.565538464045	1.1635356611337260
30,000	16610.716963893559	1.3821693212695241
50,000	16297.648562527758	1.6331424093723463
75,000	16070.615199403195	1.6701705561448272
100,000	16342.140910787613	1.6787110653758155
Three Leg Solution Results		
Population Size	**Best ΔV (km/s)**	**Best Fitness**
15,000	23901.258303415889	0.48718287051979214
30,000	21021.884796253857	0.46648377230661703
50,000	20943.715091983093	0.49364516478097936
75,000	22524.009544591485	0.43960431789251980
100,000	19800.985262807335	0.79739096792548014

Table 4.2.: Results with best ΔV and solution fitness for one, two and three leg missions

Two leg missions : The two leg mission trajectories include gravity assists. Due to energy gain from the assisting planet the solutions for two leg missions should have lesser average ΔV compared to one leg mission trajectories. But this is not the case, as shown in Fig 4.2. The indication of higher average ΔV in the graph is due to the fact that gravity assist includes additional variables such as hyperbolic excess velocity vectors (both inbound and outbound), turning angle(δ), gravity-assist partner itself, which makes the search more difficult. Consequently two leg mission search space is more complex compared to one leg mission trajectories. Complex search space means, chances of finding a good solution is considerably lowered.

Three leg mission : The three leg mission trajectories are associated with couple of gravity assists. Similar to two leg missions, expectation is to obtain very good ΔV measurements but the results show otherwise. Combination of the gravity assist partners increases the number of variables, i.e the solutions with lesser average ΔV is more difficult to be found.

From ´Fig 4.4 it is seen that average ΔV of two leg and three leg solution improves with population size. This indicates that the search space topographies of two leg and three leg solutions are more featured and complex compared to one leg solutions.

4.5.2. Gravity-Assist Sequencing

Mars is the favourite gravity-assist partner for two leg solutions. This is intuitive as well, because the chosen mission for the calculation is from Earth to Jupiter, making Mars the most likely gravity-assist partner. The increased population size definitely finds solution of better quality associated with gravity assist at Mars. Few solutions with Earth as gravity-assist partner are obtained at lower population sizes. This is due to the small population size.

Gravity-assist sequence for three leg solutions tend to cluster towards Earth-Earth sequence with the increased population size. This means good quality solutions are associated with gravity-assist sequence of Earth-Earth. At lower population sizes Mars-Mars sequence is also favoured.

4.5.3. Launch Dates and Time of Flight

Fig 4.12 and Fig 4.15 shows that one leg solutions favour particular set of launch dates and flight times respectively. But no such behaviour is observed in case of gravity-assist trajectories. The launch dates are evenly spread through out the parameter space making it difficult to extract any conclusive information.

The flight times for two leg and three leg missions span through larger ranges and the range increases with increase in gravity assists.

4.6. Conclusion

In this chapter, experimental results with Random Search is reported along with the discussion. It is conclusive from the results that population size of 100,000 is not sufficient to obtain better gravity-assist trajectories compared to non gravity-assist trajectories. Random Search analysis gives an insight to the average fitness values, ΔV and nature of the search space topography. It is also used to set up the bench mark value for other search algorithms.

- Diversity of the search criterion is guaranteed, as the search randomly gets the result from the solution or search space.
- Global optimum is definitely found if the population size is increased to infinite, in which case the solution or search space is completely explored but with astronomical time limits.

5. Implementation of Simulated Annealing

5.1. Introduction

Many combinatorial optimization problems can only be solved approximately even on present day computers. In practice any large scale combinatorial optimization problem cannot be solved for optimality and requires prohibitive amount of time. One can actually use an optimization algorithm, yielding globally optimal solution in a possibly prohibitive amount of time or an approximation algorithm to obtain acceptable solution in acceptable amount of computation time. In trajectory optimization problems such as discussed in this thesis, it is desirable to implement general approximation techniques to find near optimal solution. Thus the quality of the final solution is traded off with the computation time. Understanding of the optimization problem and basics of Simulated Annealing is required for the implementation of Simulated Annealing algorithm. [16]

5.2. Implementation of Simulated Annealing

Implementing the algorithm of Simulated Annealing requires the input parameters, definition of neighborhood, temperature schedule and objective function. As explained in chapter 3 the input parameters which determines the overall mission are.

- Launch Date
- Time of Flight
- Gravity-assist partner id
- Number of Revolutions

5.2.1. Neighborhood State

In order to perturb the input parameters, there is a need for definition of neighborhood. Upon perturbation, the parameters move to neighborhood state within the defined neighborhood. The input parameters have different ranges of values which makes the definition of a single neighborhood function for all the parameters difficult. It is also possible to define different neighborhood strategies for different input parameters. Such a definition would bring in an additional challenge of investigating the combination of neighborhood strategies, which would be worth investigating. In this thesis work, a single neighborhood strategy is used to define the neighborhood for all the mission input parameters.

Calculation of neighborhood state X_N from current state X_O

A random variable X is said to be normally distributed with mean μ and variance σ^2, if its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{(2\pi)}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right] \tag{5.1}$$

[19]

Gaussian distribution curve with maximum variation of 3σ is shown in the figure 5.1.

Neighborhood of a parameter is defined by the user input variable neighborhood percentage. Neighborhood percentage is the percentage of the range of values through which the parameter is restricted by the mission settings. The Tab 5.1 gives the mission settings and ranges for the input parameters.

Equating the product of neighborhood percentage and the range of the parameter with the interval $(\mu + 3\sigma - (\mu - 3\sigma))$ the value of σ is determined. To calculate neighborhood state X_N from current state X_O, Gaussian distribution curve is used, considering the mean value to be the current state X_O and the variance σ (already calculated with the help neighborhood percentage). [17]

The equation 5.1 can be rewritten as

$$f(x) = \frac{1}{\sigma\sqrt{(2\pi)}} \exp\left[\frac{-(X_N - X_O)^2}{2\sigma^2}\right] \tag{5.2}$$

$f(x) =$ Randomly generated number

Normalizing the randomly generated f(x) with $\frac{1}{\sigma\sqrt{(2\pi)}}$ the below equation is

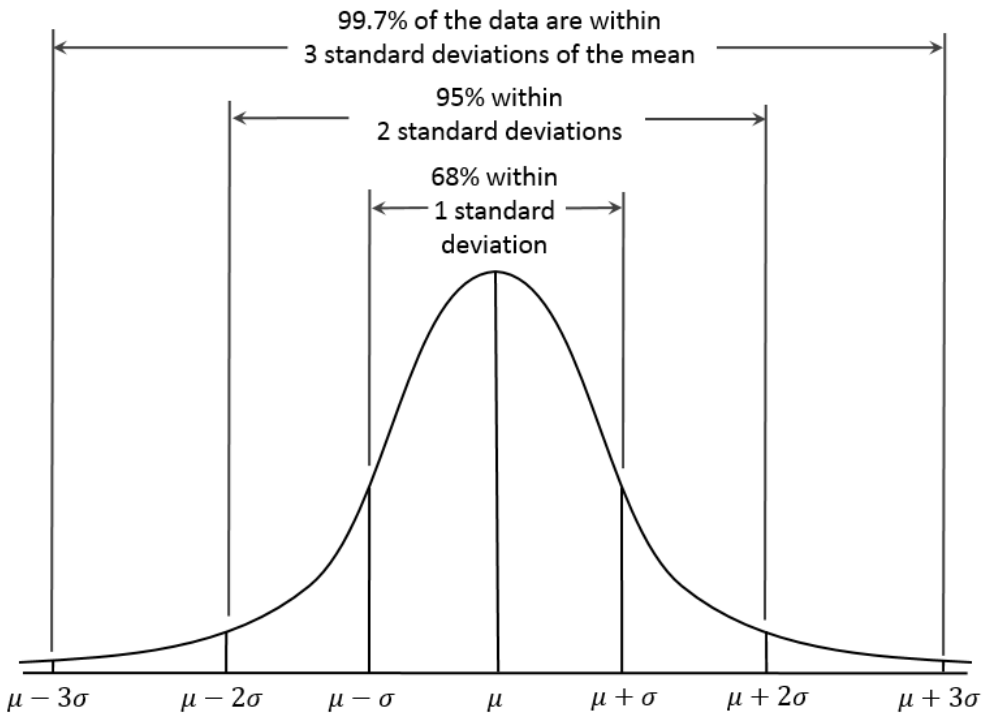


Figure 5.1.: Bell shaped curve of Normalized Gaussian Distribution

Parameter	**Restricted by**	**Range**
Launch Date	Launch window	100 (Maximum value of launch window)
Gravity-assist partner id	Maximum available partners	8 (considering 8 planets of solar system)
Time of Flight	Maximum and Minimum value of flight time	2,000 (Maximum value - Minimum value)
Number of Revolutions	Maximum value set by user	6

Table 5.1.: Mission input parameters and its ranges

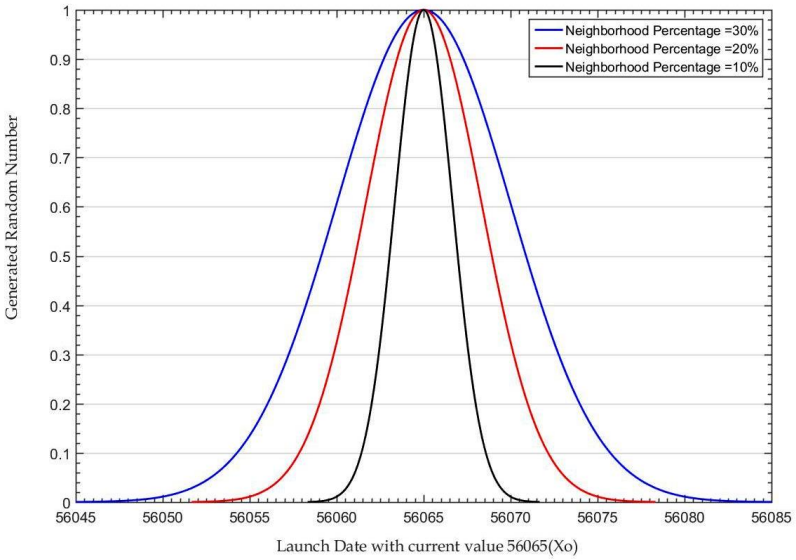


Figure 5.2.: Gaussian distribution curves for various neighborhood percentages for launch date parameter with current state $X_O = \mu$ =56065

obtained.

$$X_N = X_O + \sqrt{-\ln(f(x)) * 2\sigma^2} \tag{5.3}$$

Note : The second part of right hand side of equation 5.3 is always added to the current state X_O, this restricts the movement of neighborhood state to one direction . This is due to the fact that difference between X_N and X_O in equation 5.2 is squared. However to avoid this issue, second part of right hand side of the equation 5.3 is added to or subtracted from the current state X_O randomly ensuring that neighborhood state X_N can move in either direction from X_O.

Fig 5.2 shows the consideration of various neighborhood percentages for determining the new state X_N from the current state X_O for the Launch Date parameter. The new state value X_N lies with in the borders of the bell curves. If X_N turns out to be a fractional value, then it is accordingly rounded off to next whole number or to the previous whole number based on the value of decimal part being above or below 0.5 respectively. The same method is used to obtain neighborhood state of other variables as well.

Every variable has different range of values that it could take. For example Time of Flight has a range of 2,000 days and Launch Date has 100 days

range. But the Gravity-assist partner id can take only 8 values and Number of Revolutions can take 7 values. If the neighborhood percentage value is set lesser than 10% the algorithm may run into danger that the variables with less range (Gravity-assist partner id and Number of Revolutions) will remain on the same value, or at least the chances to remain on the same value would be more, because next state depends on the range of values a variable could take. If the neighborhood percentage is greater than 30% then the variables with larger range (Time of Flight) will have the next state at far distances from the current state. In such cases, the question that rises is "is next state really a neighbor of the current state". Considering these factors boundary values for the neighborhood percentage is set as 10% and 30%.

5.2.2. Initial Temperature and Cooling Schedule

According to Kirkpatrick et al.[21] the initial temperature could be set as the largest energy difference or largest cost function value (here solution fitness) difference of two neighboring solutions. It is also explained that initial temperature can be computed such that expected cost of the best solution that can be found at this temperature (initial temperature) is cost of the solution given by heuristic algorithm [24].

Cooling schedule is followed as defined in section 3.9.3. **Note : In this thesis work only exponential and linear temperature reduction techniques are considered.**

5.2.3. Number of Iterations

The number of iterations in which the algorithm finds the solution is limited by the prohibitive time limits. Simulated Annealing algorithm needs a long period of time to find the global optimum. A similar problem is approached with an evaluation number of 2,4,6 millions [5], but the aim of this thesis work is to verify if the algorithm is capable of finding good solutions with in a time span of 24 hours. For this reason the number of iterations are chosen to be 100,000, 200,000 and 500,000.

5.2.4. Epoch Length

Epoch length is the parameter that determines the amount of time search algorithm spends in every given temperature value. Epoch length is set proportional to the number of iterations (explained in section 3.9.4). **Note : Epoch length is set to be 100 for all the experiments in this thesis work.**

5.2.5. Acceptance Probability

The probabilistic acceptance of Simulated Annealing algorithm is explained in section 3.9.2.

Problem with Acceptance Probability

Initially the probability criterion defined in section 3.9.2 was implemented, however this resulted in a random search and not a search according to Simulated Annealing algorithm. The reason for this issue was the range of values the fitness function (cost function) could take, this is very much specific to this particular optimization problem (low-thrust gravity-assist trajectory optimization). Fitness function (cost function) defined here can take values ranging from orders of 10^0 to 10^{-20}. The difference in the cost function i.e ΔE in equation 3.5 can essentially take very small values which effectively discards the influence of temperature parameter. This would then push the probability to 1, accepting all the solutions, making the search a random one and not according to Simulated Annealing algorithm.

Adaptation of Acceptance Probability

The probability criterion defined in section 3.9.2 essentially has two properties.

- The acceptance probability should decrease with decrease in temperature.
- The acceptance probability should decrease with increase in difference of cost function (ΔE), i.e if the cost function of the new state is far worse compared to current one, then the probability to accept new state should be equally less. Lesser the difference in cost function, higher would be the probability to accept such a solution.

Any changes to the probability criterion should not affect the above two properties to ensure that the Simulated Annealing algorithm logic is retained. Considering the above factors, acceptance probability is adapted as shown below.

$$P(\Delta E) = \left(1 - \left[\frac{\Delta E}{J_{\mathrm{O}}}\right]\right) e^{\left(\frac{-1}{kT}\right)} \tag{5.4}$$

where :

$P(\Delta E)$ = Probability of acceptance of new solution.
ΔE = Difference of the cost function between the current and the new solution.
k = Constant.
T = Current Temperature.
J_O = Cost function value of current solution.

In equation 5.4 the cost function difference (ΔE) is divided by the cost function of current solution to make sure that the influence of the temperature (control parameter) on the acceptance probability is not affected due to low values (typically of orders 10^{-20}) of the cost function difference.

Note : Value of constant k in equation 5.4 is set to 1 in this thesis work.

5.3. Open Issues in Implementation

Implementation of Simulated Annealing appears to be fairly simple, however it is not the case with respect to this particular problem. The solution approach could have been different, but with the approach taken in this thesis, few noticeable issues exist and could be set up as task for future work.

5.3.1. Neighborhood definition

The definition of neighborhood is dependent on the range of values the parameter can take. To understand how this could be an issue, consider the neighborhood for parameter Time of Flight.

Assuming the current state of Time of Flight randomly chosen to be 1000 days. According to the neighborhood function explained in section 5.2.1 there will be an addition or subtraction of a certain quantity from this current value, leading to a new state. As defined in section 5.2.1 for a neighborhood percentage of 30, maximum change that can occur with the mission settings set as in Tab3.4 is an addition of 300 days.

Considering a similar correction of 300 days for a Time of Flight randomly chosen as 2700 days. Considering $\frac{300}{1000} = 30\%$ and $\frac{300}{2700} = 11.11\%$, the percentage change with respect to current value is larger for smaller values of the parameter (Time of Flight) compared to larger values of the parameter.

The root cause of this issue is down to the mission settings itself. Having a large range of parameter values for one particular mission makes the definition of the problem unclear thus making the definition of neighborhood ambiguous.

Simulated Annealing Parameters
Initial Temperature
Cooling Schedule
Neighborhood Percentage
Number of Iterations

Table 5.2.: Free parameters of Simulated Annealing

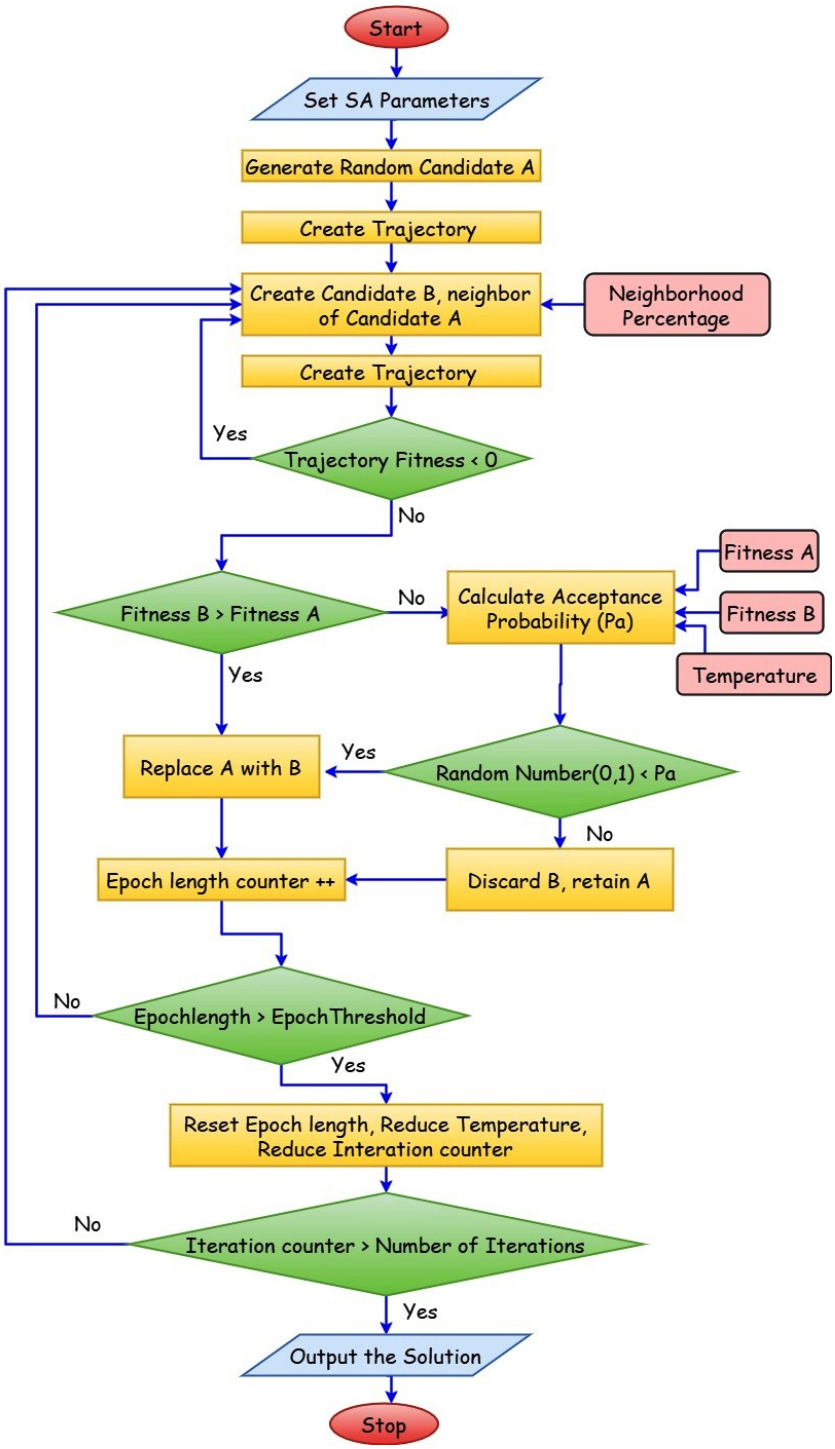
5.3.2. Change in Acceptance Probability

In equation 5.4 the difference in cost function ΔE is divided by the current solution fitness J_O to normalize the values which are extremely low, i.e of orders of magnitude 10^{-10} to 10^{-20}. This makes sure that the probability of a move to a cost function value of orders 10^{-20} to be extremely small. This would reduce the movement in search areas of bad solutions and accelerate the search. This would also avoid the algorithm to search in certain areas which is entirely filled with bad solutions but one very good solution. With the changes in probability criterion, there is a good chance that very good solutions present in and around group of bad solutions can be overlooked.

5.4. Free Parameters

The free parameters of Simulated Annealing algorithm is shown in Tab 5.2 . Tuning the free parameters affects the quality of search optimization.

5.5. Flow Diagram



6. Experiments, Results and Discussion

6.1. Introduction

In chapter 5 implementation of the Simulated Annealing is explained, also the available free parameters of this algorithm. A set of search experiments are conducted by varying the free parameters of the algorithm in order to find the global optimum or near optimal solutions. With the experimental results it is possible to verify the suitability of the search algorithm.

In the previous search algorithm (Random Search), 20 calculations are executed for each parameter setting of the algorithm. This is done to obtain consistency in the results, but the same approach is not followed for Simulated Annealing algorithm. Simulated Annealing searches for the best solution using already available information of the solution candidate and the fitness values. Simulated Annealing algorithm is expected to perform better than Random Search. However obtaining the optimal settings for Simulated Annealing algorithm is a tedious and time consuming task as every different setting has to be experimentally verified.

Note : For all the experiments, the mission settings remain same and are set as shown in Tab 3.4. Each experiment with every different parameter setting is executed ten times to obtain consistency in the results.

6.2. Experiment 1 : Cooling Schedule : Linear, Exponential

Aim : To identify the cooling schedule which performs better, and to be used in further experiments.

Settings : To start with Simulated Annealing, results from Random Search analysis is taken into consideration. It is clear from the Tab 4.2 that the maximum fitness value found is approximately 1.68. Following the explanation from section 5.2.2 the largest cost function difference (best fitness value) is used to setup the initial temperature. Temperature value 5, which is approximately

Simulated Annealing Parameters	
Initial Temperature	5
Cooling Schedule	Linear
Neighborhood Percentage	30%
Number of Iterations	100,000

Table 6.1.: Simulated Annealing parameters for experiment 1

three times the value of maximum fitness (1.68) is chosen with the initial guess that the search space could consider the solutions with fitness value three times that of maximum fitness found in Random Search analysis.

From Fig 4.5 it is clear that the best average fitness values are found at the population size of 100,000, this is a good initial guess to set the Simulated Annealing's number of iterations.

Cooling schedule is chosen as linear drop.

Neighborhood percentage can be chosen any value (10%, 20%, 30%) for this experiment as there are no previous results based on which the neighborhood percentage could be decided. Neighborhood percentage is chosen as 30%.

Simulated Annealing parameters are setup as shown in Tab 6.1.

Experiment is repeated with cooling schedule set to exponential drop.

6.2.1. Results

Tab 6.2 shows the results for both exponential and linear cooling schedules, clearly in all the cases, i.e one, two and three leg missions, solution fitness values obtained using exponential cooling schedule are higher compared to fitness values obtained using linear cooling schedule. Fig 6.1 and Fig 6.2 shows the variation of solution fitness (blue) and temperature (red) against the iterations, i.e behaviour of the search algorithm for exponential and linear cooling schedules respectively. It is interesting to note that the algorithm behaves similar to Hill Climbing logic as the temperature value reduces to 0.3 (approximately as seen on plots).

6.2.2. Discussion

Exponential cooling schedule performed better compared to linear cooling, this behaviour is because the algorithm spends most of the time at lower temperature ranges. At lower temperature ranges the probability of acceptance of poor solution candidates is less (explained in section 3.9.2). Even though the results

Linear Temperature drop		
Solution	**Best ΔV (km/s)**	**Best Fitness**
One leg	16041.257215133706	1.4423041632861515
Two leg	16460.023559148838	1.1552092286396589
Three leg	24924.827843995896	0.5834871371799919
Exponential Temperature drop		
Solution	**Best ΔV (km/s)**	**Best Fitness**
One leg	14708.591135715053	1.46027771462329300
Two leg	16165.386369181258	3.71163414407390180
Three leg	24021.871017490645	0.67893509516217443

Table 6.2.: Results with best ΔV and solution fitness for linear and exponential temperature drop

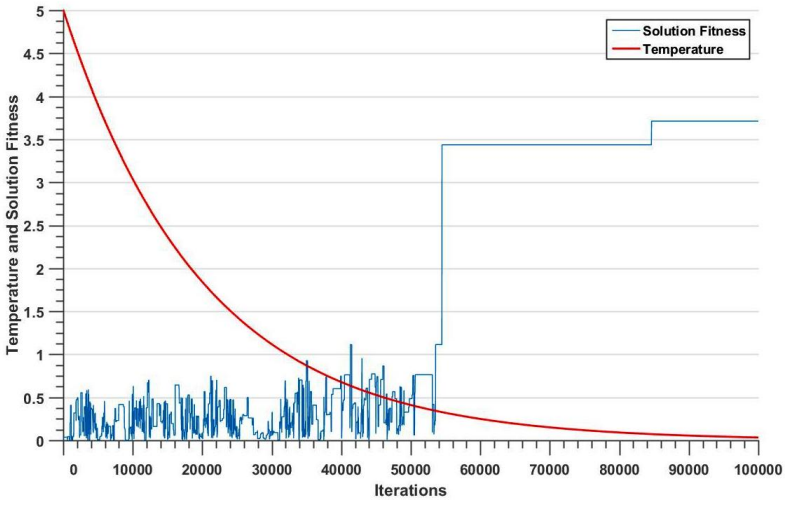


Figure 6.1.: Simulated Annealing with exponential temperature drop

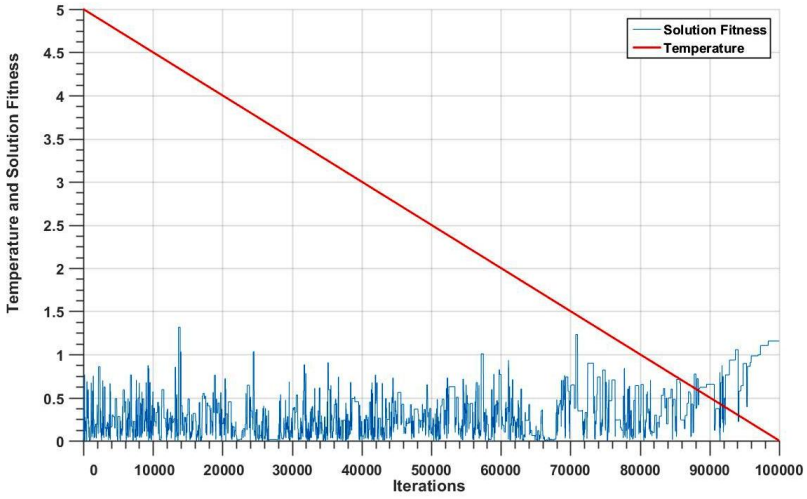


Figure 6.2.: Simulated Annealing with linear temperature drop

with exponential cooling are better, exponential cooling is always accompanied by the risk of convergence towards the local optimum.

As explained in section 3.9.2 Simulated Annealing algorithm has both global and local search phases. In exponential cooling schedule, the algorithm has more time to improvise on the obtained result, i.e the algorithm spends more time in local search phase. In linear cooling schedule, the algorithm spends more time in global search phase. Therefore, the cooling schedule should be appropriately adjusted to have a right mixture of both local search and global search phases. This process requires verification of several other cooling schedules with several experiments. However in this thesis work, a detailed research on cooling schedule is not considered as it is not the main goal of this thesis.

Based on the best fitness results, exponential cooling schedule is considered for further experiments.

The best fitness found from this experiment is 3.711. This encouraged to investigate higher initial temperatures.

6.3. Experiment 2 : Variation of Initial Temperatures

Aim : To find the best initial temperature.

Settings : In experiment 1 the initial temperature of Simulated Annealing is set up approximately three times the best fitness found in Random Search

Simulated Annealing Parameters	
Initial Temperature	5, 10, 15
Cooling schedule	Exponential
Neighborhood Percentage	30%
Number of Iterations	100,000

Table 6.3.: Simulated Annealing parameters for experiment 2

method. The same approach is used here to investigate the initial temperature up to approximately three times that of best fitness found in experiment 1, which is shown in Tab 6.2.

The best fitness value found in experiment 1 is approximately 3.7. $3.7 * 3 = 11.1$, this makes the expectation of best fitness to be around 11, to make sure that the solutions with fitness greater than 11 are not missed by the search algorithm, initial temperature value of 15 is investigated. To investigate if a pattern exists with variation of initial temperature, the values of 5 and 10 are also investigated. Retaining the other parameters as in the previous experiment, Simulated Annealing parameters are setup as shown in Tab 6.3.

6.3.1. Results

The results of ΔV and solution fitness are considered with variation of initial temperature.

Average ΔV : Fig 6.3 shows the average ΔV values for one, two and three leg missions plotted against the different initial temperatures.

Average ΔV for three leg solutions is extremely large (in orders of 10^6) compared to average ΔV for one and two leg solutions at the initial temperature value of 5, hence it is not seen in Fig 6.3. Fig 6.4 shows the extremely large values of average ΔV for three leg solutions. Maximum average ΔV values for one leg and two leg solutions at initial temperature 15 are $16.08km/s$ and $17.2km/s$ respectively. Minimum average ΔV values for one leg, two leg solutions are $15.92km/s$ and $16.78km/s$ respectively found at initial temperature of 5. The average ΔV values for one leg and two leg solutions did not show much of variation with respect to the initial temperature of the search algorithm. Average ΔV values for three leg missions improved from $30km/s$ to $25.8km/s$ for temperature varying from 10 to 15.

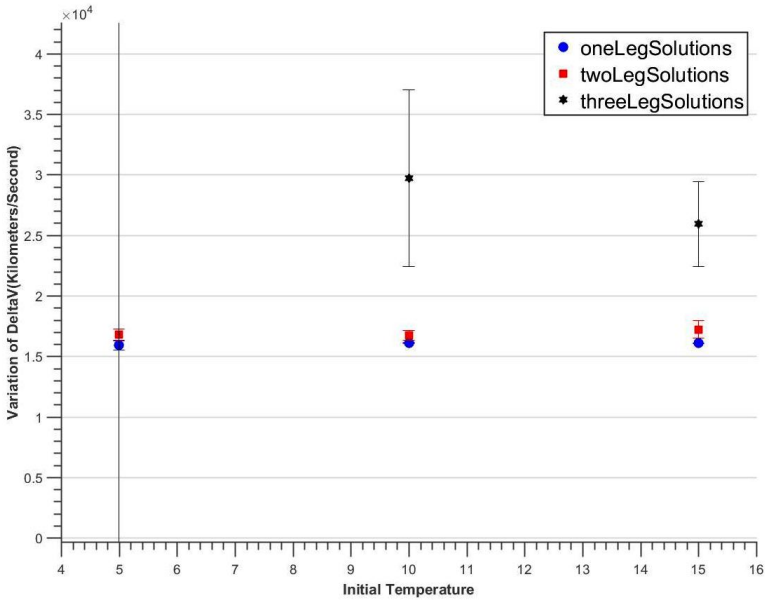


Figure 6.3.: Average ΔV comparison of one, two and three leg solutions with different initial temperatures

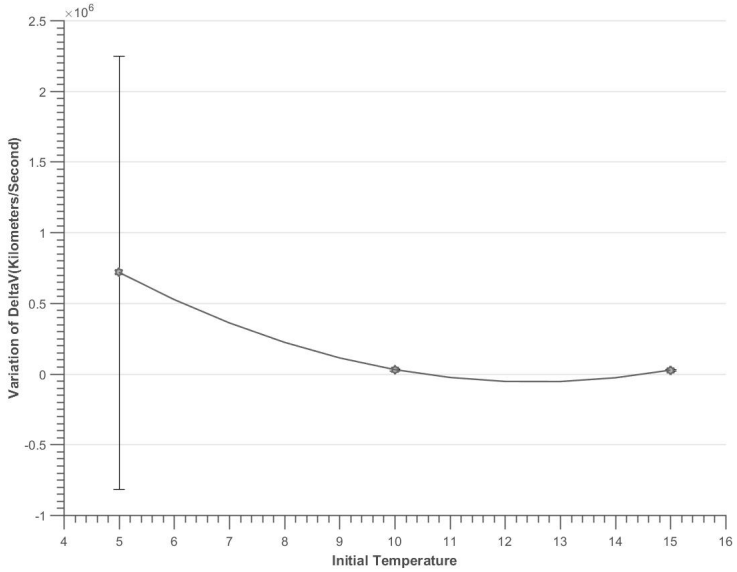


Figure 6.4.: Average ΔV values for three leg solutions with various initial temperatures

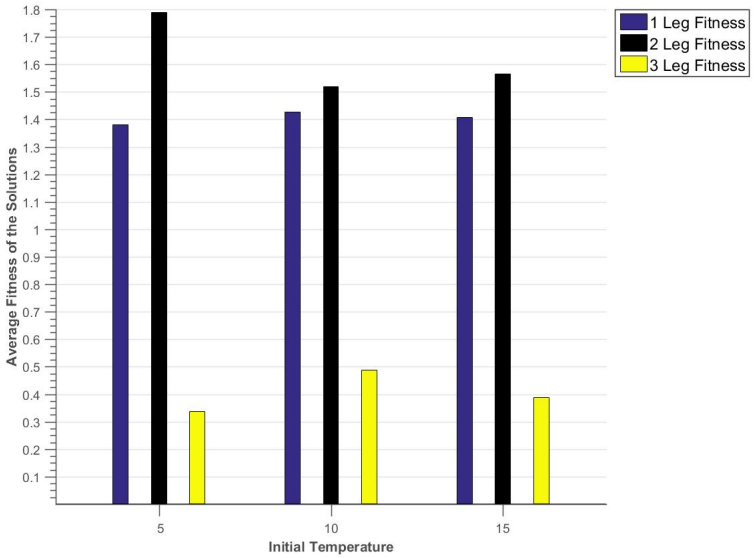


Figure 6.5.: Average solution fitness comparison of one, two and three leg solutions with different initial temperatures

Solution Fitness : Fig 6.5 shows the average fitness values for the one, two and three leg solutions. The fitness values for one leg trajectories show hardly any improvement and remain approximately same (1.4) for all three initial temperature values, but two leg solutions favour temperature value of 5 and three leg solutions favour temperature value of 10.

6.3.2. Discussion

The variation of initial temperature causes changes in the amount of time the system spends in local search and global search phases (explained in 3.9.2), also affects how quickly the temperature is reduced. With the increase in initial temperature, the expectation is to find solutions with better fitness and lesser ΔV. With higher initial temperature the algorithm rapidly moves into different parts of search space and chances of finding the solution with better fitness are more. However the results show that one leg, two leg and three leg solutions all behaved differently. One leg solutions did not favor any of the initial temperature (even though there is minor improvement of 0.04 at the temperature of 10), two and three leg solutions had better fitness values at initial temperature value of 5 and 10 respectively.

It is also noticeable that two leg solutions have an average fitness of 1.5 or more at all three initial temperatures. This shows that Simulated Annealing algorithm can find two leg solutions with good fitness at ease, i.e the search space consists of many local optimums.

From Fig 6.5 it is clear that the maximum solution fitness is found at initial temperature of 5, hence the initial temperature value is set as 5 for further experiments.

6.4. Comparison with Random Search

From Experiment 2 the results of Simulated Annealing algorithm are available, at this point, a comparison with results from Random Search analysis is possible.

From Fig 6.3 the average ΔV of $16.2km/s$, $17.2km/s$ and $25.8km/s$ for one leg, two leg and three leg solutions respectively are observed at an initial temperature of 15. All three results are comparable with the results obtained in Random Search, as shown in Fig 4.4 and three leg solutions did show an improvement of $(29.4km/s - 25.8km/s = 3.6km/s)$.

Comparing the best average fitness values from the Fig 4.5 of Random Search analysis and Fig 6.5 of Simulated Annealing, average fitness improvement of two leg solutions $(1.79 - 1.35 = 0.44)$ and three leg solutions $(0.495 - 0.39 = 0.105)$ is witnessed.

Results from further experiments are not compared with Random Search results because the fitness graph and the ΔV graph from experiment 2 shows that Simulated Annealing do perform better than the bench mark set by Random Search algorithm. Simulated Annealing can be used in low-thrust gravity-assist optimization.

6.5. Experiment 3 : Variation of Neighborhood Percentages

Aim : To find the best neighborhood percentage.

Settings : The three available neighborhood percentages are 10%, 20% and 30%. From the previous experiments, it is clear that Simulated Annealing performs well with initial temperature of 5 and exponential cooling schedule, also retaining the number of iterations as in previous experiments the calculations are performed for all the three neighborhood percentages.

Simulated Annealing parameters are setup as shown in Tab 6.4

Simulated Annealing Parameters	
Initial Temperature	5
Cooling schedule	Exponential
Neighborhood Percentage	10,20,30%
Number of Iterations	100,000

Table 6.4.: Simulated Annealing parameters for experiment 3

6.5.1. Results

The results of average ΔV with neighborhood percentage of 10% are extremely large and in orders of 10^8. Hence, only fitness function values are considered to evaluate the best possible neighborhood percentage.

Solution Fitness Fig 6.6 shows the average fitness of the solutions with various neighborhood percentages.

It is clear from the Fig 6.6 that the algorithm yielded extremely poor results for neighborhood percentage of 10%. The results improved with increase in neighborhood percentage. The fitness of one leg solutions which did not show major deviation to any of the previous parameters of Simulated Annealing is also affected by the neighborhood percentage. Even in Random Search results, shown in Fig 4.5, average solution fitness for one leg solutions is higher compared to that obtained with neighborhood percentage of 10%. Average solution fitness for two leg and three leg solutions are 1.2 and 0.05 respectively. Average three leg solution fitness of 0.05 is the lowest value compared to average three leg solution fitness obtained from every other experiment. Interesting observation is that, the three leg solutions and one leg solutions have slightly better fitness with values of (0.38-0.36=0.02) and (1.44-1.38=0.06) respectively at neighborhood percentage of 20% compared to fitness at neighborhood percentage of 30%.

The neighborhood percentage 30% provided the best results in terms of average fitness.

6.5.2. Discussion

The decrease in neighborhood percentage deteriorates the quality of results. The small neighborhood restricts the movement in parameter space hence the movement in search space, i.e the algorithm cannot escape the region of bad solutions quickly.

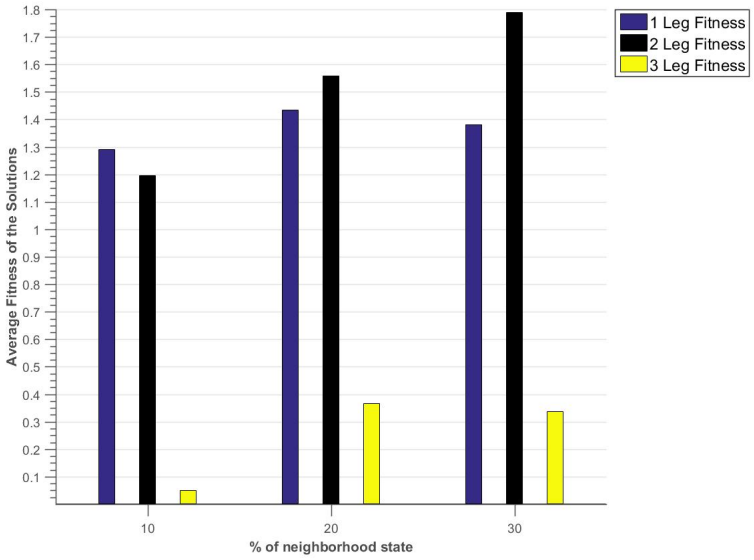


Figure 6.6.: Average solution fitness comparison of one, two, three leg solutions with various neighborhood percentage

The average fitness value of two leg solutions, increases along with the neighborhood percentage. The average fitness values of one leg and three leg solutions improve at neighborhood percentage of 20% but tend to decrease with further increase of neighborhood percentage, i.e at neighborhood percentage of 30%. Each solution space one leg, two leg and three leg solution space needs to be explored at different neighborhood percentages to understand the effect of restriction of parameter space.

However the best fitness values are found at neighborhood percentage of 30% hence this value is retained in further experiments.

6.6. Experiment 4 : Variation of Number of Iterations

Aim : To find the best suitable number of iterations.

Settings : In this experiment the number of iterations are set to be 100,000, 200,000, and 500,000. The rest of the Simulated Annealing parameters are set as shown in Tab 6.5. The number of iterations are set to maximum of 500,000 because the goal of the optimization tool is to identify optimal trajectories with

Simulated Annealing Parameters	
Initial Temperature	5
Cooling schedule	Exponential
Neighborhood Percentage	30%
Number of Iterations	100,000, 200,000, 500,000

Table 6.5.: Simulated Annealing parameters for experiment 4

in the limited time frame. Generally Simulated Annealing is run for millions of iterations.

6.6.1. Results

Solution Fitness

Fig 6.7 shows the average solution fitness of one, two and three leg solutions for various number of iterations. Increase in number of iterations resulted in increased fitness of all the solutions i.e one leg, two leg and three leg solutions. Average fitness of two leg solutions show massive improvement from 1.78 to 2.47 for number of iterations of 100,000 and 500,000 respectively. Average fitness for three leg solutions doubled (from 0.34 to 0.68) for number of iterations of 100,000 and 500,000. Smallest improvement of (1.47-1.39 = 0.08)is associated with average one leg solution fitness.

6.6.2. Discussion

The improvement in the solution fitness with increase in number of iterations is no surprise. The increment in number of iterations increased the chances to fetch solutions with better fitness, because the algorithm can now search in larger pool of solutions and larger area of search space. Even though the one leg solution fitness improved from 1.39 to 1.47 for number of iterations 100,000 to 500,000 respectively, it is definitely not feasible to spend huge amount of time on finding very good one leg solution. In search of no gravity-assist trajectory a Random Search method is more suitable. This also indicates that the search space topography for one leg solutions has more plateaus and local optimum are sporadic. A similar argument is made in Random Search analysis as well.

Although 500,000 iteration produced very good fitness values, the average execution time was nearly 2.5 days (212534 seconds), due to time constraints,

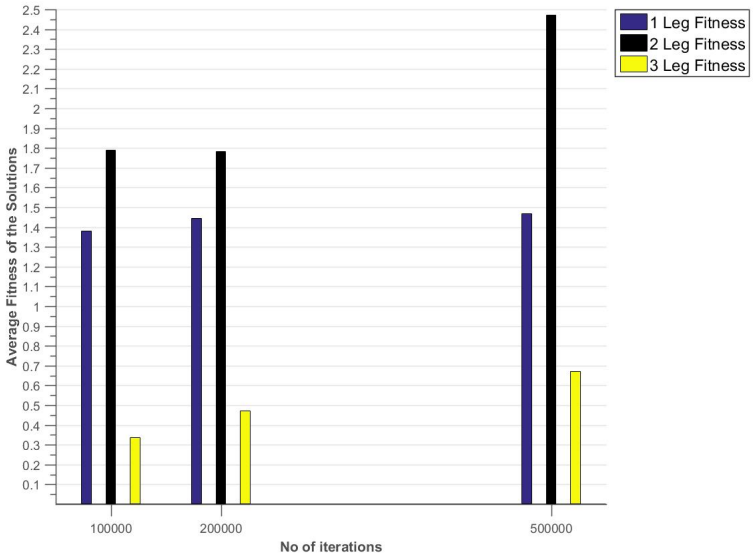


Figure 6.7.: Average solution fitness comparison of one, two, three leg solutions with variation of number of iterations

iteration of 200,000 (approximate average execution time of 0.96 days(83634 seconds)) is selected for next experiment.

6.7. Experiment 5 : Investigation of solutions with gradual reduction of Neighborhood Percentage

The above 4 experiments did give an insight into the settings of Simulated Annealing for finding the optimal solution in limited time frame.

In the above experiments it is seen that two leg solutions are most sensitive to variation of parameters in Simulated Annealing. For this reason, in this experiment only two leg solutions are considered.

In experiment 3, the neighborhood percentage of 10% produced very poor solutions. For every experiment, the gravity-assist partner of the solutions are plotted against the variation of Simulated Annealing parameter (explained in appendix A.1). When gravity-assist partner selection plot for experiment 3 is considered, few interesting results are obtained.

Fig 6.8 shows gravity-assist partner selection for two leg solutions and Fig 6.9 shows the gravity-assist partner selection for three leg solutions.

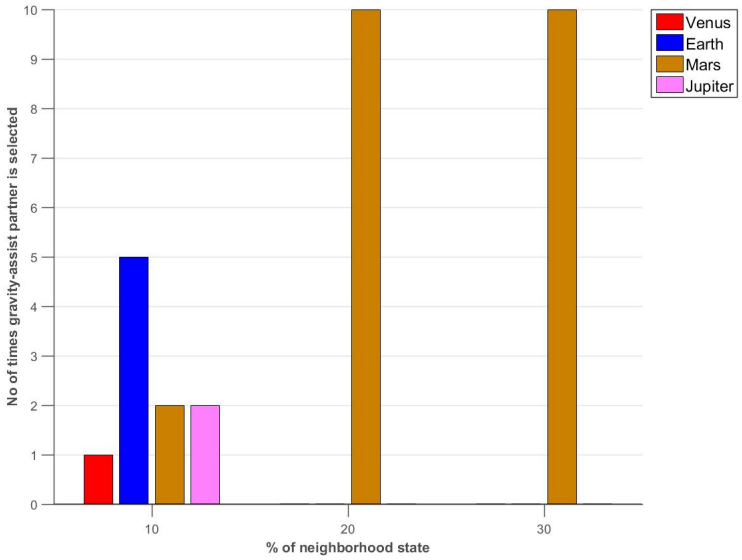


Figure 6.8.: Gravity-assist partner selection for two leg solutions with variation of neighborhood percentage

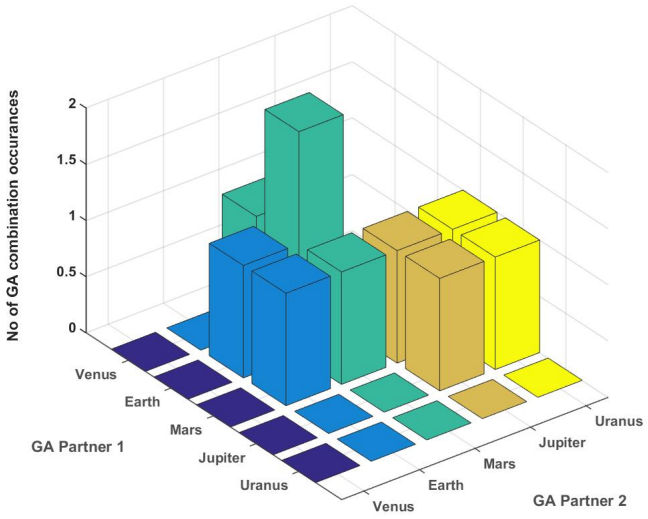


Figure 6.9.: Gravity-assist partner selection for three leg solutions with neighborhood percentage of 10%

Simulated Annealing Parameters	
Initial Temperature	5
Cooling Schedule	Linear
Neighborhood Percentage	30% to 10%(linear reduction during the search)
Number of Iterations	200,000

Table 6.6.: Simulated Annealing parameters for experiment 5

The gravity-assist partner for two leg solutions in most of the cases remained Mars or Earth (also seen in Random Search analysis Fig 4.6). With the neighborhood percentage set to 10% the solutions had Venus and Jupiter as gravity-assist partners. In case of three leg solutions the neighborhood percentage of 10% resulted in few interesting solutions having gravity-assist partner as Uranus which was not the case for any of the parameter settings in any other experiments. A mission to Jupiter having a gravity assist at Uranus would definitely be a solution with less fitness, this can also be seen in Fig 6.6. The reason for having such a solution is setting the neighborhood percentage to a small value of 10%. It is important to notice that the algorithm is actually exploring the regions of search space which it would ideally have less chance to explore when the neighborhood percentage set to 30%.

This behaviour led to the experiment 5 in which the neighborhood percentage is gradually decreased with decrease in temperature. At the start of the experiment, neighborhood percentage is set to 30% and decreased gradually to 10% along with temperature value reaching zero. The gradual reduction of neighborhood percentage should fetch a group of solutions with less fitness, however chances of finding one good solution or solution with very good ΔV compared to previous experiments are good. The linear drop of temperature has more chances of finding this one very good solution compared to that of exponential temperature drop, hence the cooling schedule is changed to linear drop.

Simulated Annealing parameters are setup as shown in Tab 6.6.

6.7.1. Results

Tab 6.7 shows the ΔV values and fitness values obtained for this experiment. The results with considerably low ΔV values are obtained, however the average fitness of the solutions is 0.8358. The results are presented in a tabular form, not shown in graphical form in order to emphasise that gradually decreasing

Results for decreasing Neighborhood	
Two Leg Solution Results	
ΔV (km/s)	Fitness
13772.767831084866	1.0599586244024346
16687.453408551224	1.2882398534941744
17316.772608693682	0.7464247110813087
17503.755181522993	0.73163656048123349
14014.141335586428	0.99185104064892149
27759.565236015893	0.2044411372450281
17626.691789192326	0.69502864617295446
14561.50885485572	0.94541023230817667
17663.147731921737	0.95755582925458316
19179.869680025367	0.73842469535665056

Table 6.7.: ΔV and solution fitness for two leg missions

neighborhood could produce 8 good results in 10 executions and 3 results equal to or less than $14.57 km/s$.

6.7.2. Discussion

This method did improve the search, the solution with ΔV of $13.772 km/s$ is not found in any of the previous experiments. By reducing the neighborhood percentage gradually, algorithm could explore the regions which it did not explore with a fixed neighborhood percentage. This encourages to investigate various other methods of decreasing the neighborhood. In this thesis work, the neighborhood percentage is reduced only in a linear fashion.

The search for optimal trajectories with two gravity assist might need more number of iterations, but for one gravity assist it is definitely possible to locate a good solution with adjustment in the parameters of Simulated Annealing. This indicates that Simulated Annealing is a suitable algorithm for optimization tool with strict time lines.

7. Conclusion

7.1. Conclusion

In this thesis a preliminary work is done towards the search of near optimal low-thrust gravity-assist trajectories. A Random Search analysis is performed to verify if it is possible to obtain acceptable solutions. A well known global optimization algorithm, Simulated Annealing is implemented and series of experiments are conducted to analyse if near optimal solutions are found.

It is concluded from the results that Random Search is not an efficient search method for this particular combinatorial optimization problem. Even though a few good results were obtained, the search method could not deliver consistent results, due to the complex nature of the search space. With every gravity-assist there are infinite possible values with which the hyperbolic excess velocity vector V_∞ could be rotated at and it is no surprise that Random Search could not perform well in case of gravity-assist trajectory search. However the method could still be used in search of non gravity-assist trajectories, as this was the only case in which Random Search produced consistent and acceptable results or solutions with good fitness values.

Simulated Annealing search provided better results compared to Random Search method with respect to two leg mission trajectories. There were no significant improvements when one leg mission and three leg mission trajectories were considered. It cannot be concluded that Simulated Annealing search is not suitable for this optimization problem, considering the point that, number of iterations tested in this thesis is far less (nearly one tenth) compared to other trajectory optimization problems involving Simulated Annealing. The behaviour of Simulated Annealing algorithm indicated that it would definitely lead to solutions with better fitness values with increased number of iterations.

The best ΔV value out of all the calculations performed in this thesis work is $13.772 km/s$ and was obtained when the neighborhood percentage was gradually decreased along with the temperature during the search. This result gives a hope that with further adjustments to the free parameters of the Simulated Annealing (i.e the way temperature is reduced, the way neighborhood percentage is reduced) consistent and good results can be obtained even with reduced number of iterations.

7.2. Scope for Future Work

It would be extremely beneficial to have an optimization tool which can optimize a low-thrust gravity-assist trajectory with in a restricted time frame. Initial efforts for such an attempt is done in this thesis work. There are several points on which the future work of this thesis could be based on.

Neighborhood : A new complex definition of neighborhood could be implemented to overcome the issues with single neighborhood function. Neighborhood function for individual decision variables or separate neighborhood functions for variables having similar ranges can be defined.

Acceptance Probability : Adaptation of the acceptance probability defined for implementation of Simulated Annealing does not include Boltzman constant. The probability equation could be further improved to implement Boltzman constant in the equation.

Cooling Schedule : In this thesis work, only couple of cooling schedules are investigated. Investigation of various other cooling schedules can give more information on obtaining results much quicker.

Epoch length : Epoch length discussed here is kept constant, effects of variation of Epoch length is one further area to look into.

A. Appendix

A.1. Gravity-Assist Sequencing for Simulated Annealing

In this section, the plots of various gravity-assist partners for every solution is plotted against the variation of Simulated Annealing parameter. Objective of such a plot is to identify if any of the sequence is favored by the solutions.

A.2. Experiment 2 : Variation of Initial Temperatures

Fig A.1 shows the gravity-assist partner selection for two leg solutions against variation of initial temperature.

Fig A.2, Fig A.2, Fig A.2 shows the gravity-assist partner selection for three leg solutions for initial temperature values of 5, 10 and 15 respectively.

A.3. Experiment 3 : Variation of Neighborhood Percentages

Fig A.5 shows the gravity-assist partner selection for two leg solutions against variation of neighborhood percentage.

Fig A.6, Fig **??**, Fig A.8 shows the gravity-assist partner selection for three leg solutions for neighborhood percentage values of 10%, 20% and 30% respectively.

A.4. Experiment 4 : Variation of Number of Iterations

Fig A.9 shows the gravity-assist partner selection for two leg solutions against variation of number of iterations.

Fig A.10, Fig A.10, Fig A.10 shows the gravity-assist partner selection for three leg solutions for number of iteration values of 100,000, 200,000 and 500,000 respectively.

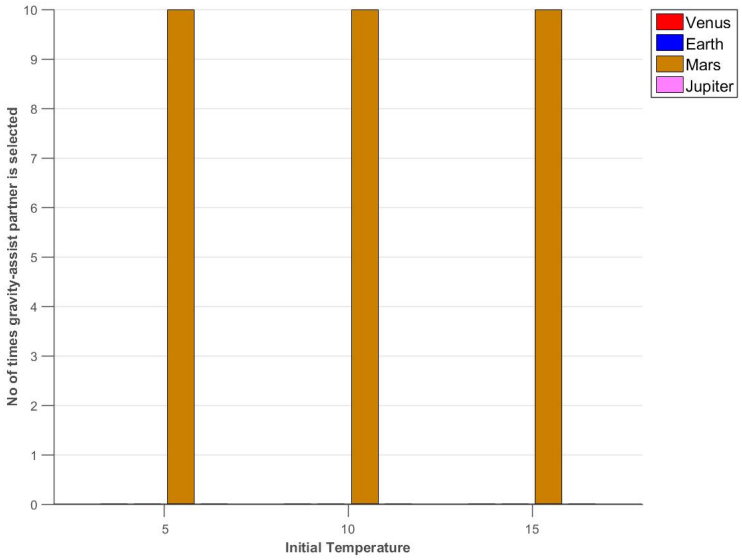


Figure A.1.: Gravity-assist partner selection for two leg solutions with variation of initial temperature.

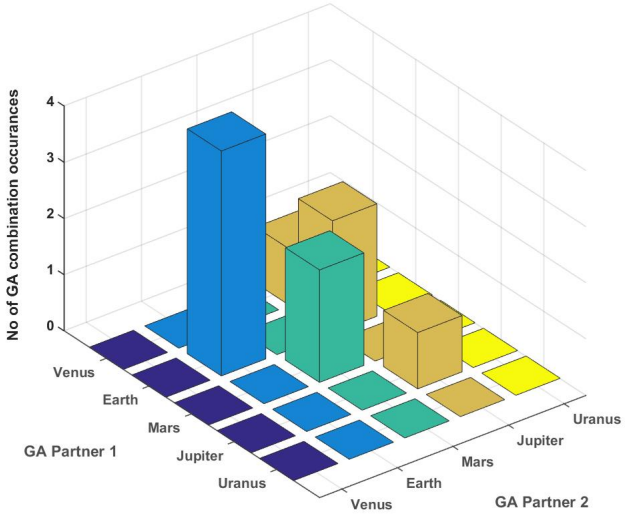


Figure A.2.: Gravity-assist partner selection for three leg solutions with initial temperature 5.

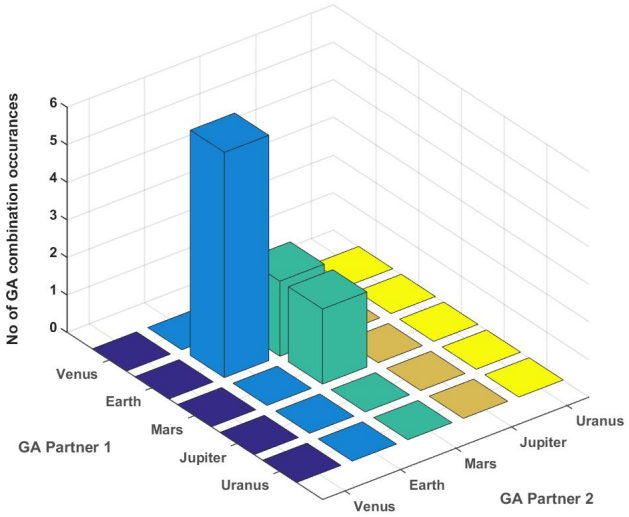


Figure A.3.: Gravity-assist partner selection for three leg solutions with initial temperature 10.

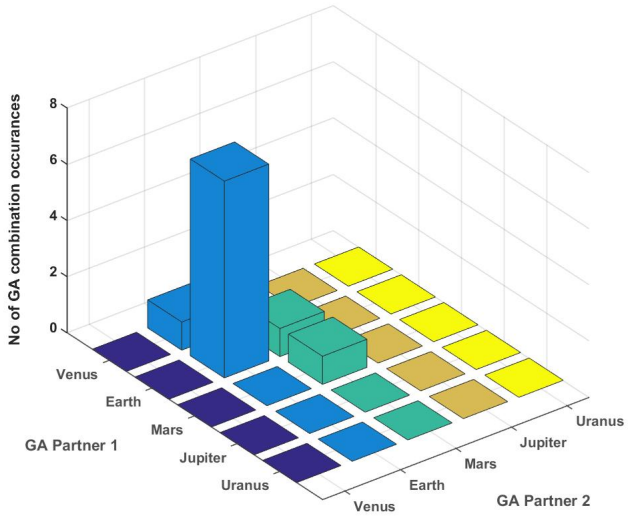


Figure A.4.: Gravity-assist partner selection for three leg solutions with initial temperature 15.

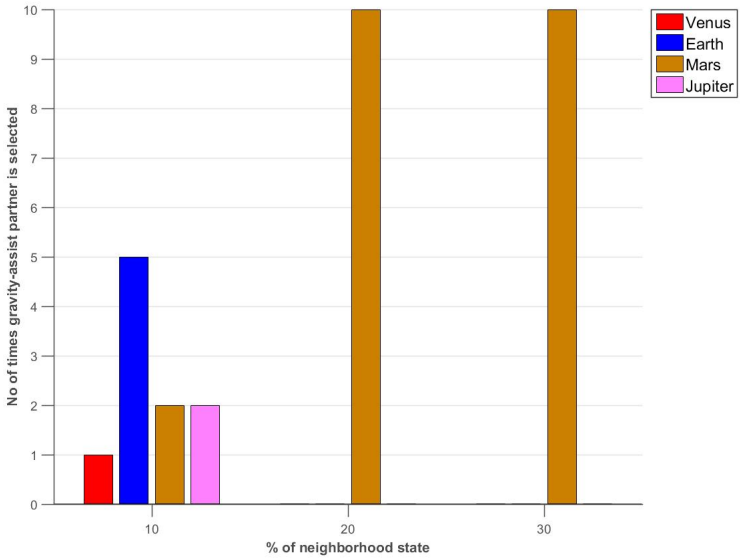


Figure A.5.: Gravity-assist partner selection for two leg solutions with variation of neighborhood percentage.

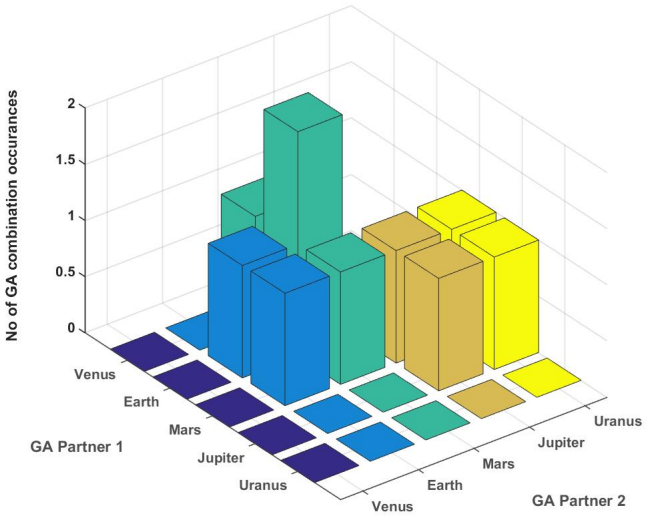


Figure A.6.: Gravity-assist partner selection for three leg solutions neighborhood percentage of 10%.

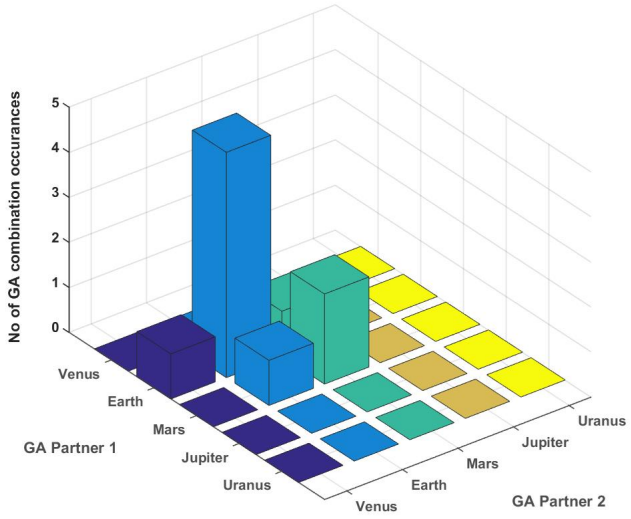


Figure A.7.: Gravity-assist partner selection for three leg solutions neighborhood percentage of 20%.

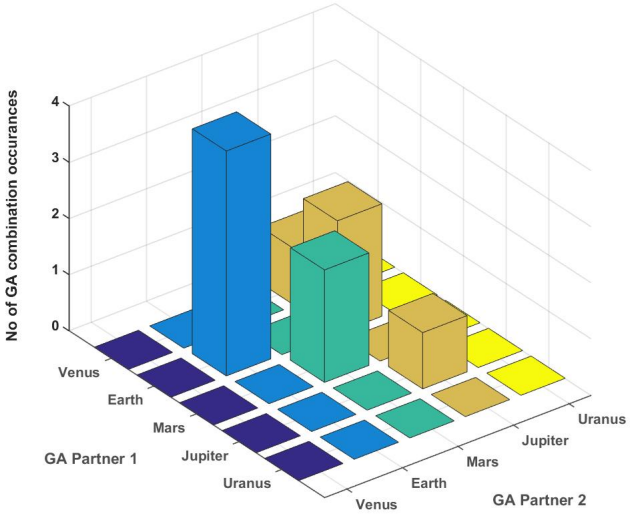


Figure A.8.: Gravity-assist partner selection for three leg solutions neighborhood percentage of 30%.

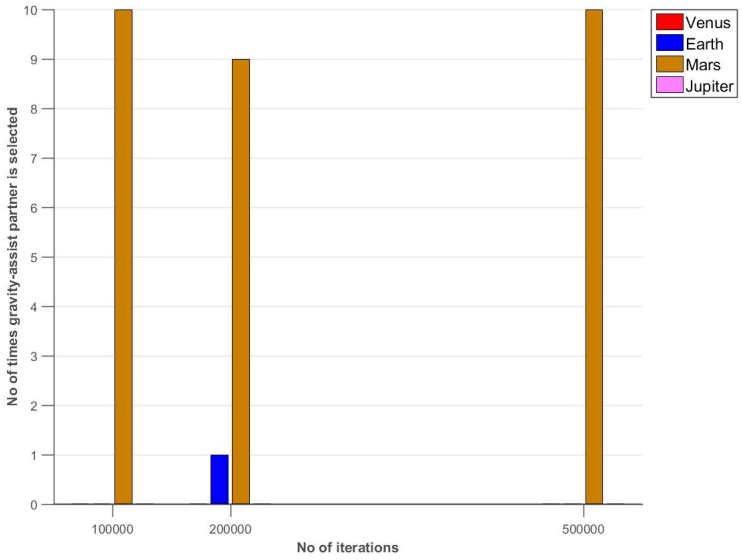


Figure A.9.: Gravity-assist partner selection for two leg solutions with variation of number of iterations.

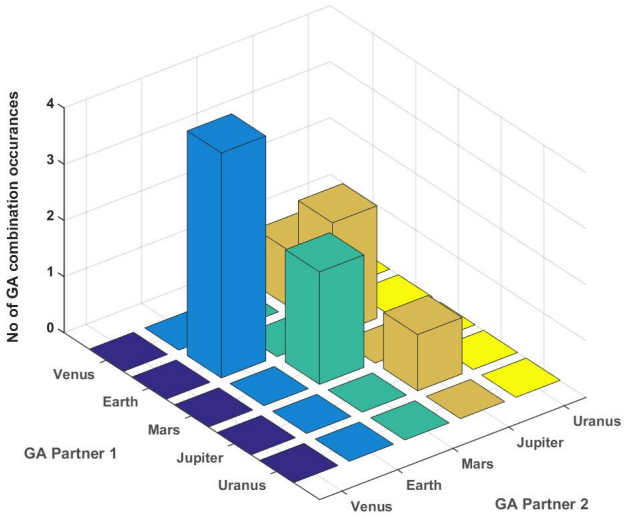


Figure A.10.: Gravity-assist partner selection for three leg solutions with number of iteration value 100,000.

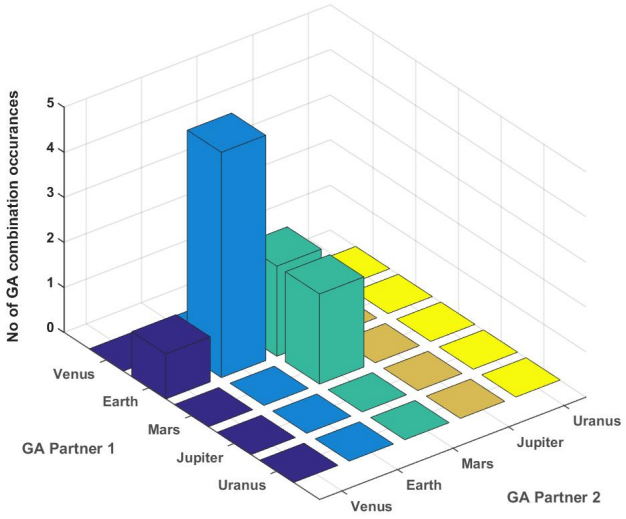


Figure A.11.: Gravity-assist partner selection for three leg solutions with number of iteration value 200,000.

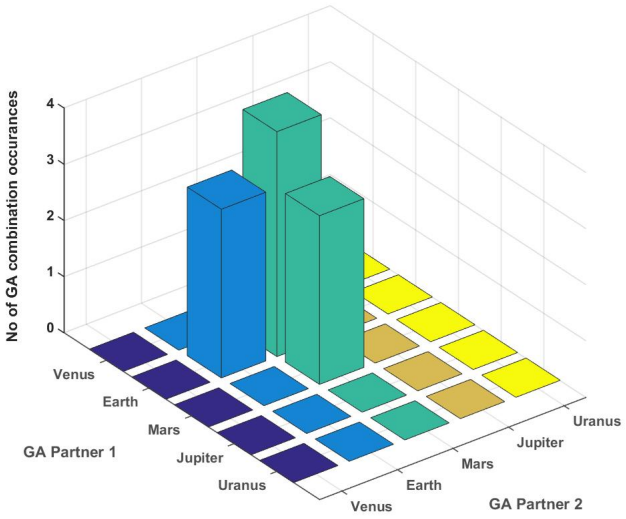


Figure A.12.: Gravity-assist partner selection for three leg solutions with number of iteration value 500,000.

A.5. Part of the source code for implementation of Simulated Annealing

```
/*#########################################################################
# run_sequence_optimization is a overloaded method to execute
# the Optimization algorithm. The same structure which was
# used for Differential Evolution is retained for implementing
# Simulated Annealing as well. This method basically implements
# the logic of the Simulated Annealing algorithm. A sequence
# candidate which contains the slots for the parameter values
# is randomly filled and then the search process begins with
# this initial candidate. run_sequence_optimization calls
# createNeighborhood method which will move the initial candidate
# to its neighbouring location in the parameter space and the
# fitness would be evaluated. This way the search shall follow
# Simulated Annealing algorithm to find the optimal set of parameters.
#########################################################################*/


int run_sequence_optimization( cCalc &CalcObject,
    vector<cSequenceCand> &SequencePop, int leg_counter,
ofstream &log_obj, ofstream &log_obj1)
{
   double iteration_number = 1;
    //Counter for the number of evaluations in the search.
   int population_counter = 0;
   int replacement_counter = 0;
   //Counts how often down hill movement happens in the search.

   SimulatedAnnealing SA_Optimizer;
   //Object of Simulated Annealing class for calling Simulated
       Annealing methods.
   double maxTemperature = CalcObject.getMaxTemperature();
   //Maximum temperature to start with as set by the user in input
       text file.
   int iMaxNumberOfIterations = CalcObject.getNumberOfIterations();
   //Maximum nuber of Iterations or search evaluations to be
       performed. Also set by user.
   int iEpochlength = 100;
   //Number of evaluations at each temperature level.
   double currentTemperature = maxTemperature;

   vector<cSequenceCand> TrialSequencePop(1);
   //Sequence candidate which stores the values of parameters(Vector).
```

```
for(iteration_number = 1; iteration_number
    <=iMaxNumberOfIterations; iteration_number++)
 // initially fixing the number of iterations
{
    for (int iEpoch =1; iEpoch<=iEpochlength; iEpoch++)
   {
      log_obj <<"-----------------------------------------" << endl;
      log_obj <<"Trial Number: " << iteration_number
               <<"Epoch length: " << iEpoch << endl;
      log_obj <<"-----------------------------------------" << endl;
      cout <<"-----------------------------------------" << endl;
      cout <<"Trial Number: " << iteration_number
            <<"Epoch length: " << iEpoch << endl;
      cout <<"-----------------------------------------" << endl;

      TrialSequencePop.at(population_counter).vTrajectories.resize(leg_counter);
      //number of leg becomes size of vector containing the
          trajectories of a
         //given sequence candidate
      TrialSequencePop.at(population_counter).vGA_partner_id_sequence.resize(leg_counter-1)
      TrialSequencePop.at(population_counter).d_theSequenceFitness =
          -1;
      //Fitness is assigned negative if the fitting parametrs are
          not found and
         //calculations are repeated.

      do
      {
      // Function to create the Neighborstate, only the adress of
          original and new vector needs to be
      // fed in, rest all will be read in from the input file.
      SA_Optimizer.createNeighborState(TrialSequencePop,
          SequencePop, CalcObject, leg_counter, currentTemperature,
          maxTemperature);
      cMission TrialMission(CalcObject,
          leg_counter,TrialSequencePop.at(population_counter).MissionVariables[0],
       TrialSequencePop.at(population_counter).MissionVariables[1],
           TrialSequencePop.at(population_counter).MissionVariables[2]);
      //sequence is: Nrev_mission, FlightTime_mission,
          LaunchDate_mission
      TrialMission.createTrialSequence(leg_counter,
          TrialSequencePop.at(population_counter), TrialMission,
          CalcObject,
          log_obj,TrialSequencePop.at(population_counter).vGA_partner_id_sequence);
```

```
        //if succcessful the sequence fitness is changed to > 0, if
            not, fitness remains -1 (see above),
        //i.e. a new vector is created, new trial vector, new
            sequence, until successful;
           //errors result in "return"
        }
        while(
            TrialSequencePop.at(population_counter).d_theSequenceFitness
            < 0);
        if(CalcObject.theDocumentation)
        {
        cout << "---------------------------------" << endl;
        }
        log_obj << "-----------------------------" << endl;

         int trial_counter =0; //counts through trial population
        log_obj << "PopulationFitness("<<trial_counter+1<<")" <<
            SequencePop[trial_counter].d_theSequenceFitness <<
        "TrialPopulationFitness("<<trial_counter+1<<")" <<
            TrialSequencePop[trial_counter].d_theSequenceFitness <<
            endl;
        if(SequencePop[trial_counter].d_theSequenceFitness <=
            TrialSequencePop[trial_counter].d_theSequenceFitness)
        {
              log_obj << "Population Member: " << trial_counter+1 << "
                  is replaced." << endl;
              //do replacement, i.e. trialpopulation member becomes
                  new population member
SequencePop[trial_counter].MissionVariables[0] =
    TrialSequencePop[trial_counter].MissionVariables[0];
SequencePop[trial_counter].MissionVariables[1] =
    TrialSequencePop[trial_counter].MissionVariables[1];
SequencePop[trial_counter].MissionVariables[2] =
    TrialSequencePop[trial_counter].MissionVariables[2];
SequencePop[trial_counter].vGA_partner_id_sequence =
    TrialSequencePop[trial_counter].vGA_partner_id_sequence;
SequencePop[trial_counter].d_theSequenceDeltaV =
    TrialSequencePop[trial_counter].d_theSequenceDeltaV;
SequencePop[trial_counter].d_theSequenceFitness =
    TrialSequencePop[trial_counter].d_theSequenceFitness;
              //global variables have been stored up to here
              //next: storing the data of each segment's (counted
                  through by "copy_counter" trajectory)
              int copy_counter =0;
```

```
            for(copy_counter = 0; copy_counter < leg_counter;
                copy_counter++)
            {
SequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[0]
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[0];
SequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[1]
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[1];
SequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[2]
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[2];

SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_a =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_a;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_b =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_b;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_c =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_c;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_d =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_d;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_e =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_e;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_f =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_f;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_g =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_g;

SequencePop[trial_counter].vTrajectories[copy_counter].radius_hist =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].radius_hist;
SequencePop[trial_counter].vTrajectories[copy_counter].theta_hist =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theta_hist;
SequencePop[trial_counter].vTrajectories[copy_counter].thrust_hist =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].thrust_hist;

SequencePop[trial_counter].vTrajectories[copy_counter].DeltaV =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].DeltaV;
SequencePop[trial_counter].vTrajectories[copy_counter].Fitness =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Fitness;

SequencePop[trial_counter].vTrajectories[copy_counter].theStartBody =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theStartBody;
SequencePop[trial_counter].vTrajectories[copy_counter].theGA_partner_id
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theGA_partner_id;
```

```
SequencePop[trial_counter].vTrajectories[copy_counter].theEndBody =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theEndBody;
SequencePop[trial_counter].vTrajectories[copy_counter].theTurningAngleDelta
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theTurningAngleDelta;
SequencePop[trial_counter].vTrajectories[copy_counter].theR_peri_flyby
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theR_peri_flyby;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_x
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_x;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_y
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_y;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_x
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_x;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_y
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_y;
              }
        }
        else
        {
//If the fitness is worse, we accept the solution based on the
    probability depending on temperature.
int trial_counter =0;
uniform_real_distribution<double> unif_real_dis(0,1);
random_device rd;
mt19937 gen(rd());
double a_random_double = unif_real_dis(gen);
double deltaE = 1-((SequencePop[trial_counter].d_theSequenceFitness -
    TrialSequencePop[trial_counter].d_theSequenceFitness)
/SequencePop[trial_counter].d_theSequenceFitness);
double prob_function =deltaE*exp(-( 1)/(currentTemperature));
log_obj << "prob_function "<< prob_function << endl;

           if(a_random_double < prob_function )
             {
```

```
log_obj << "Population Member: " << trial_counter+1 << " is replaced."
    << endl;
SequencePop[trial_counter].MissionVariables[0] =
    TrialSequencePop[trial_counter].MissionVariables[0];
SequencePop[trial_counter].MissionVariables[1] =
    TrialSequencePop[trial_counter].MissionVariables[1];
SequencePop[trial_counter].MissionVariables[2] =
    TrialSequencePop[trial_counter].MissionVariables[2];
SequencePop[trial_counter].vGA_partner_id_sequence =
    TrialSequencePop[trial_counter].vGA_partner_id_sequence;
SequencePop[trial_counter].d_theSequenceDeltaV =
    TrialSequencePop[trial_counter].d_theSequenceDeltaV;
SequencePop[trial_counter].d_theSequenceFitness =
    TrialSequencePop[trial_counter].d_theSequenceFitness;
//global variables have been stored up to here
//next: storing the data of each segment's (counted through by
    "copy_counter" trajectory
              int copy_counter =0;
              for(copy_counter = 0; copy_counter < leg_counter;
                  copy_counter++)
              {
SequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[0]
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[0];
SequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[1]
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[1];
SequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[2]
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].ControlVariables[2];

SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_a =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_a;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_b =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_b;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_c =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_c;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_d =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_d;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_e =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_e;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_f =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_f;
SequencePop[trial_counter].vTrajectories[copy_counter].Coeff_g =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Coeff_g;
```

```
SequencePop[trial_counter].vTrajectories[copy_counter].radius_hist =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].radius_hist;
SequencePop[trial_counter].vTrajectories[copy_counter].theta_hist =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theta_hist;
SequencePop[trial_counter].vTrajectories[copy_counter].thrust_hist =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].thrust_hist;

SequencePop[trial_counter].vTrajectories[copy_counter].DeltaV =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].DeltaV;
SequencePop[trial_counter].vTrajectories[copy_counter].Fitness =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].Fitness;

SequencePop[trial_counter].vTrajectories[copy_counter].theStartBody =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theStartBody;
SequencePop[trial_counter].vTrajectories[copy_counter].theGA_partner_id
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theGA_partner_id;
SequencePop[trial_counter].vTrajectories[copy_counter].theEndBody =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theEndBody;
SequencePop[trial_counter].vTrajectories[copy_counter].theTurningAngleDelta
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theTurningAngleDelta;
SequencePop[trial_counter].vTrajectories[copy_counter].theR_peri_flyby
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].theR_peri_flyby;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_x
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_x;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_y
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_departure_y;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_x
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_x;
SequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_y
    =
    TrialSequencePop[trial_counter].vTrajectories[copy_counter].v_inf_arrival_y;
              }
              replacement_counter++;
```

```
//increase the value of replacement counter to get to know how many
    replacements have happened.
           }
        }
        log_obj1<< SequencePop[trial_counter].d_theSequenceFitness
               << "                    "
            << TrialSequencePop[trial_counter].d_theSequenceFitness
               << "                    " << currentTemperature << endl;
        }
      // To reduce the Tempeature.
      currentTemperature
          =(currentTemperature-(maxTemperature/iMaxNumberOfIterations));
        // To reduce the temperature linearly
      // currentTemperature = maxTemperature*exp(-(iteration_number /
          (iMaxNumberOfIterations/5)));
      // to be used for Exponential reduction of Temperature, exp(-5)
          itself is a very small number,
      // hence max number of iterations a divided by 5.
      }
   log_obj1 << "No of Downhill movements " << replacement_counter <<
       endl;

return iteration_number;
}
```

B. List of Figures

C. List of Tables

D. Bibliography

[1] *MODIFIED JULIAN DATE*. http://tycho.usno.navy.mil/mjd.html.

[2] Anastassios E.Petropoulos, James M.Longuski: *Automated Design of Low-Thrust Gravity-Assist Trajectories — School of Aeronautics and Astronautics, Purdue University, West Lafayette, Indiana*. https://engineering.purdue.edu/people/james.m.longuski.1/ConferencePapersPresentations/2000AutomatedDesignofLow-ThrustGravity-AssistTrajectories.pdf.

[3] Anastassios E.Petropoulos, James M.Longuski: *Shape-Based Algorithm for Automated Design of Low-Thrust, Gravity-Assist Trajectories— Purdue University, West Lafayette, Indiana— Journal of Spacecraft and Rockets*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.6620&rep=rep1&type=pdf, September-October 2004.

[4] Bradley J.Wall, Bruce A.Conway: *Shape-Based Approach to Low-Thrust Rendezvous Trajectory Design — Journal of Guidance, Control, and Dynamics*. https://arc.aiaa.org/doi/abs/10.2514/1.36848, January - February 2009.

[5] Chit Hong Yam, Francesco Biscani, Dario Izzo: *Global Optimization of Low-Thrust Trajectories via Impulsive Delta-V Transcription — European Space Agency, ESTEC, The Netherlands*. http://www.esa.int/gsp/ACT/doc/MAD/pub/ACT-RPR-MAD-2009-GlobalOptLowThrust.pdf.

[6] Gunter Dueck, Tobias Scheuer: *Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing — IBM Scientific Center, Heidelberg, WestGermany*. http://www.sciencedirect.com/science/article/pii/002199919090201B, September 1989.

[7] J. Peraire, S. Widnall: *Variable Mass Systems: The Rocket Equation — Lecture 14, 16.07 Dynamics, Fall 2008, Version 2.0*. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec14.pdf.

[8] James A. Van Allen: *Gravitational assist in celestial mechanics — a tutorial — Department of Physics and Astronomy, University of Iowa.* http://www.uv.es/jfgf/AJP00448.pdf, November 2002.

[9] John E.Prussing, Bruce A.Conway: *OrbitalMechanics.* Oxford University Press, February 1993.

[10] Ledesma Sergio, Jose Ruiz, Guadalupe Garcia: *Simulated Annealing Evolution — University of Guanajuato, Department of Computer Engineering, Salamanca, Mexico.* http://www.intechopen.com/books/simulated-annealing-advances-applications-and-hybridizations/simulated-annealing-evolution.

[11] Marco Sabbadini, Giorgio Saccoccia, Margherita Buoso: *Electric Propulsion Technology Programmes — ESA Publications Division, The Netherlands.* http://www.esa.int/esapub/br/br187/br187.pdf.

[12] Moon-Won Park, Yeong-Dae Kim: *A Systematic procedure for setting parameters in Simulated Annealing Algorithms — Department of Industrial Engineering, Korea Advanced Institute of Science and Technology.* https://pdfs.semanticscholar.org/78d0/363647d5ecb68956c9ee3fab80005618b083.pdf, June 1997.

[13] NASA,Space Shuttle: *NASA - Space Shuttle Main Engines.* https://www.nasa.gov/returntoflight/system/system_SSME.html.

[14] N.Baba: *Convergence of a random optimization method for constrained optimization problems — Journal of Optimization Theory and Applications.* https://link.springer.com/article/10.1007%2FBF00935752?LI=true, April 1981.

[15] Paul W.Keaton: *Low-Thrust Rocket Trajectories.* http://physicsware.net/pdf/Low-Thrust.pdf, November 2002.

[16] P.J.M. van Laarhoven, E.H.L. Aarts: *SimulatedAnnealing :Theory and Applications.* D.Reidel Publishng Company, Philips Research Laboratories, Eindhoven, Netherlands, February 1987.

[17] Prof. C. Balaji: *Design and Optimization of Energy Systems — Dept of Mechanical Engineering, Indian Institute of Technology, Madras.* http://nptel.ac.in/reviewed_pdfs/112106064/lec40.pdf.

[18] Rafael Marti: *Multi-Start methods — University of Valencia, Spain.* http://www.uv.es/rmarti/paper/docs/multi2.pdf.

[19] Ribeiro, Maria Isabel: *Gaussian Probability Density Functions: Properties and Error Characterization — Institute for Systems and Robotics*. http://users.isr.ist.utl.pt/~mir/pub/probability.pdf, February 2004.

[20] Sergio Ledesma, Gabriel Avina, Raul Sanchez: *Practical Considerations for Implementation of Simulated Annealing*. http://cdn.intechopen.com/pdfs/4631.pdf.

[21] S.Kirkpatrick, C.D.Gelatt, Jr.M.P.Vecchi: *Optimization by Simulated Annealing*. http://www2.stat.duke.edu/~scs/Courses/Stat376/Papers/TemperAnneal/KirkpatrickAnnealScience1983.pdf, May 1983.

[22] Stuart Geman, Donald Geman: *Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images*. http://www.stat.cmu.edu/~acthomas/724/Geman.pdf, November 1984.

[23] Thomas Weise: *Global Optimization Algorithms - Theory and Application*. http://www.it-weise.de/projects/book.pdf, June 2009.

[24] Varanelli, James M.: *On the Acceleration of Simulated Annealing*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.9310&rep=rep1&type=pdf, May 1983.

[25] Volker Mailwald: *About Combining Tisserand Graph Gravity-Assist Sequencing With Low-Thrust Trajectory Optimization — German Aerospace Center (DLR), Institute of Space Systems, System Analysis Space Segment, Bremen, Germany*. http://elib.dlr.de/104263/1/6th-ICATT-paper_02.pdf, March 2016. International Conference on Astrodynamics Tools and Techniques.

[26] Volker Maiwald: *A new method for optimization of low-thrust gravity-assist sequences*. http://adsabs.harvard.edu/abs/2017CEAS..tmp...5M, February 2017. (Accessed on 05/21/2017).

[27] Yaghout Nourani, Bjarne Andresen: *Comparison of Simulated Annealing Cooling Strategies — Orsted Laboratory, University of Copenhagen, Denmark*. http://iopscience.iop.org/article/10.1088/0305-4470/31/41/011/pdf, July 1998.

[28] Zelda B. Zabinsky: *Random Search Algorithms*. http://courses.washington.edu/inde510/516/AdapRandomSearch4.05.2009.pdf, April 2009.