Spring 5-1-2022

Optimization of Orbital Trajectories Using NeuroEvolution of Augmenting Topologies

Nathan Wetherell
nathan.wetherell@uconn.edu

Optimization of Orbital Trajectories Using NeuroEvolution of Augmenting Topologies

Nathan Wetherell

University of Connecticut

April 29th, 2022

Thesis Advisors

Dr. Bryan Weber (Chair)

Dr. Cara Battersby

Dr. Jonathan Trump

Honors Advisor

Dr. Jason Lee

i. Abstract

This project aims to determine the feasibility of using NeuroEvolution of Augmenting Topologies (NEAT), an advanced neural network evolution scheme, to optimize orbital transfer trajectories. More specifically, this project compares a genetically evolved neural network to a standard Hohmann transfer between Earth and Mars. To test these two methods, an N-body simulation environment was created to accurately determine the result of gravitational interactions on a theoretical spacecraft when combined with planned engine burns. Once created, this simulation environment was used to train the neural networks created using the NEAT Python module. A genetic algorithm was used to modify the topology of the network in addition to the traditional weight and bias modifications to produce a highly effective individual or batch of individuals that can process various positional and velocity inputs to generate an efficient orbital transfer. The performance of these neural networks was measured by comparing the transfer burn plans they generate to the standard Hohmann transfer using a variety of factors such as transfer time and fuel consumption. This paper presents a background in neural networks, genetic algorithms, and NEAT, discusses the methods chosen for this specific project, and summarizes and draws conclusions from the results of the neural network training.

Ultimately, it was found that the created program was effective in training neural networks to optimize for either time, delta-v, or a combination of both. More specifically, the neural networks consistently created solutions that were more time-efficient than the standard Hohmann transfer and could make equally effective solutions when considering time and delta-v in equal weights. The program struggled when attempting to optimize for delta-v, opening an area of possible improvement in future project iterations.

ii. Table of Contents

iii. Table of Figures

iv. Table of Tables

v. Basic Files and Program Flow

Table 1 - Basic file names, descriptions, and categorization

File Name	Purpose	Type
BurnCalculations.py	Determine change in velocity and fuel consumption for given burn; calculates circularization burns	Calculation
BurnPlanner.py	Handles storing and retrieving of custom burn data files	Special Function
CircularizationCalculations.py	Handles requests for circularization calculations; sends requests to BurnCalculations.py	Calculation
config-feedforward.txt	Configuration file for NEAT; sets mutation, reproduction, speciation, and population parameters	Configuration
Constants.py	Stores constants for N-body simulation and control values for Hohmann transfer burn	Configuration
HohmannTransferCalculations.py	Calculates Hohmann Earth orbit to Mars orbit transfer and insertion burns	Calculation
HorizonsDataSaver.py	Saves large data files from JPL Horizons to pickle files	Special Function
MainRunner.py	Connects various other files, calculates fitness scores, stores and retrieves data; runs control simulations	Runner
MasterRunner.py	User-facing file; runs simple GUI; runs relevant files for NEAT training and manual simulations	Runner (Master)
NbodyCalculations.py	Calculates all necessary quantities for N-body simulation and dispenses simulation history to other files	Calculation
NBodyRunner.py	Runs the NbodyCalcualtions.py functions; gathers relevant data for use by other files	Runner
NeatRunner.py	Runs the NEAT framework for neural network training; connects fitness function, visualization, and data inputs; saves	Runner
NeatTester.py	Tests existing, trained neural networks using saved training files	Special Function
NeatVisualizer.py	Generates plots for speciation and fitness scores over time; creates visual representation of neural network	Plot Generation
Outfile.csv	Stores training data for easy analysis	Data Storage

parallelRunner.py	Handles multiprocessing for NeatRunner.py	Runner
Plotting.py	Plots live and post-run information for the N-body simulation	Plot Generation
UserVariables.py	Configuration file for craft, fitness score weighting, simulation, plotting, and neural network training parameters	Configuration

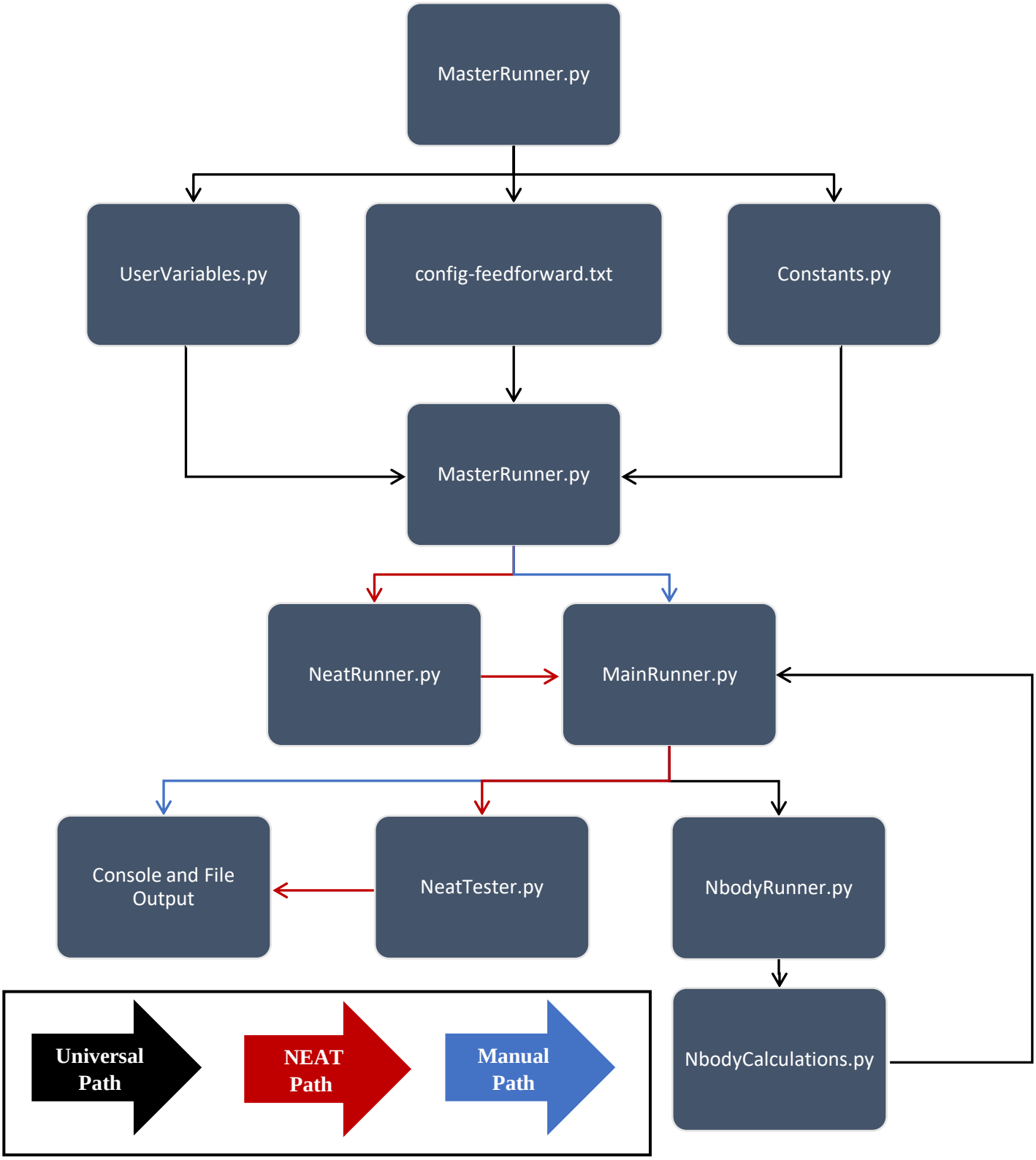

Figure 1 - Program flow for files given MasterRunner.py start

1 Introduction

1.1 Project Scope

This project aimed to determine the feasibility of using NeuroEvolution of Augmenting Topologies (NEAT), an advanced neural work evolution scheme, to optimize orbital transfer trajectories. More specifically, this project compared a NEAT evolved neural network to a standard Hohmann transfer between Earth and Mars. To test these two methods, an N-body simulation environment was created to accurately determine the result of gravitational interactions on a theoretical spacecraft when combined with planned engine burns. Once created, this simulation environment was used to train the neural network, modifying the topology and other network parameters to produce a highly effective individual or batch of individuals for comparison to the standard Hohmann transfer. The data from these comparisons were then analyzed to gauge the feasibility of using NEAT for orbital transfer optimization given different mission profiles and parameters.

1.2 Current State of the Literature

Past studies that attempted to use genetic algorithms to determine optimal orbital trajectories have found that genetic algorithms cannot consistently generate more efficient transfers than classical closed-form solutions[1]–[4]. One reason for this may be the constraints placed on the simulations, including the absence of gravitational assist trajectories and the low number of burns (the ignition of fuel to accelerate the craft) available to the program. A gravitational assist trajectory takes advantage of the relative motion of a planet and its gravity to change the velocity of a spacecraft. These maneuvers are typically used to increase the speed of a craft with minimal fuel expenditure, but they can also change its direction or even slow it down. In one of the studies which excluded this principle [2], the researchers constructed a model composed of two planetary bodies through which a simulated craft could maneuver. The orbit was determined using a range of four to seven burns. Another study [1]performed a similar analysis with the goal of rendezvousing with another spacecraft in a circular orbit. Both of these studies found that the solutions provided by this process were generally less efficient than the classic, analytical solutions they used as controls. Despite these results, the studies did provide insight into the optimal construction of genetic algorithms. They concluded that allowing a broad range of starting values for each orbital parameter resulted in the algorithm converging to a global rather than a local optimization point.

1.3 Goals

To build on the current research in this field, specific goals were established beyond the basic specifications outlined in Section 1. In regard to the N-body simulation, the ability for the simulation to run at a rate of at least 15 days/second while maintaining

stability and consistency was one key goal. The consistency of this simulation was measured by the variation of an object put in a theoretically circular and stable orbit when the simulation was run for several in-program years. Acceptable variations depended on the orbit's proximity to the host planet and its orbital period.

The baseline Hohmann transfer used as the control to determine the neural network performance was established using analytical solutions to a typical planetary transfer problem. This solution was modified for inclusion into the less ideal but more realistic simulation environment, including accounting for the elliptical rather than assumed circular orbits of both Earth and Mars, but was based on the ideal analytical calculation.

Regarding the evolution of the neural network, the established genetic algorithm needed to consistently produce neural networks that determined feasible solutions to the Earth-Mars transfer problem presented to it. This neural network evolution process was required to determine this solution regardless of the planets' relative starting positions and velocities. Developing a general neural network that could solve any given Hohmann or more generalized transfer was considered a stretch goal due to the extensive training times required. These networks were evolved using a custom-made fitness function that provided the evolutionary criteria for the NEAT process. This fitness function was required to be easily modifiable to fine-tune the neural network training and standardized against the control Hohmann transfer for easy comparison.

2 Methods - Theory and Implementation

Multiple concepts in the fields of orbital mechanics and computational physics were necessary to accomplish the central goals of this project. Namely, the concept of an N-body simulation was used to establish an environment for the neural network to learn and adapt within. Concerning the creation of the neural network, concepts such as fitness functions, evolution parameters, and parameter tuning were used to better optimize the training regime. A standard Hohmann transfer was used to establish a control for the performance and calculated fitness of the neural network to be compared to.

2.1 N-Body Simulations

2.1.1 Theory

An N-body simulation seeks to predict the motion of a system of particles by calculating the forces between them. In the case of this project, the only force being examined is gravity, meaning that the most relevant type of N-body simulation is the gravitational N-body simulation (simply referred to as an N-body simulation in the rest of this paper).

A gravitational N-body simulation relies on Newton's Law of Universal Gravitation, shown in Eq. (1). In this equation, G is defined as the gravitational constant (equal to $6.67430 * 10^{-11} \frac{Nm^2}{kg^2}$), m_1 and m_2 are the masses of the two interacting particles, and r is the distance between the two objects. The result of this equation is the force acting on the two particles, F, where the force is applied on along the direction of the vector form of r. A typical N-body simulation applies these formulas repeatedly to all particles in the system and sums the forces on each of the individual particles. The particle-specific force is converted to an acceleration of that particle using its mass. The particle is assumed to be accelerated by its net force in the correct direction.

$$F = G\frac{m_1 m_2}{r^2} \tag{1}$$

Arising out of this simple version of an N-body simulation are two primary issues that must be addressed. The first is that the increasing number of particles increases the number of force equations that need to be calculated exponentially. For example, a two-particle system will have one equation to determine all net forces, a three-particle system has three equations, a four-particle system has six equations, and a five-particle system has ten equations. In a more general form, this takes the form of Eq. (2), where N is the number of particles in the system and E represents the number of equations necessary to solve the system. Essentially, as the number of particles in the system increases, the processing time needed to solve the system accurately increases exponentially.

$$E = \frac{N * (N-1)}{2} \tag{2}$$

The second primary issue with standard gravitational N-body simulations is the heavy reliance on the time step duration and the associated performance impact. Any N-body simulation relies on time steps which are the frequency of the calculations of the equations. Shorter time steps are favorable as the particles are allowed to drift less between force calculations, meaning more accurate solutions are generated. Shrinking time steps to only allow for almost imperceptible drift, however, results in a heavy impact on the performance and speed of the simulation. Doubling the number of time steps also doubles the time it takes for the simulation to complete. If any repeated testing needs to be done, a balance between simulation accuracy and speed must be struck.

One method for getting around the second issue of time step reliance is to use a Runge-Kutta Method. These methods essentially use a spread of points around the point of interest to more accurately solve a given differential equation in the form of an initial value problem. In the case of a gravitational N-body simulation, a Runge-Kutta Method can more accurately determine the result of a moving body subject to gravitational forces over a discretized set of time steps compared to the Euler method that uses a single solution of Newton's Law of Universal Gravitation. The equation solved by each iteration of the Runge-Kutta Method is shown as Eq. (3), where a_{x2} and a_{y2} are the acceleration of the object in the x and y-directions, G is the gravitational constant, M_1 is the mass of object 1, r is the total separation of particles 1 and 2, and dx and dy are the separations between the two particles in the x and y-directions, respectively.

$$a_{x2} = \frac{-GM_1}{r^3} * dx, \quad a_{y2} = -\frac{GM_1}{r^3} * dy \tag{3}$$

$$y_{new} = y_0 + \frac{dt}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4}$$

Instead of just solving these equations once and applying the acceleration over time to generate the final position as in Euler's Method, to determine the new positional vector of the particle, a weighted average of four of these solutions is used in the most common Runge-Kutta Method known as RK4. This weighted average takes the form of Eq. (4) where y_0 is the initial position of the particle, dt is the change in time (the time step), and k_1, k_2, k_3, and k_4 are the slopes of the position function used to determine the final position. k_1 is the slope of the

function (the velocity) at the beginning of the time interval (t); this is the same slope as is found using Euler's Method. k_2 is the estimated value of the slope of the particle velocity at the midpoint of the function. This midpoint value is found using the original slope, k_1. The midpoint is then calculated using the slope k_2 and the slope at this new midpoint, k_3, is found. Finally, the slope at the ending time, $t + dt$, is found by using the initial position, y_0, and the slope designated as k_3. This final slope is assigned to k_4. These four values are then averaged using specific weights and applied to the time step length and the original position to determine the final position of the particle, y. This process can be seen visually in Figure 2.

$$y_{new} = y_0 + \frac{dt}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5}$$

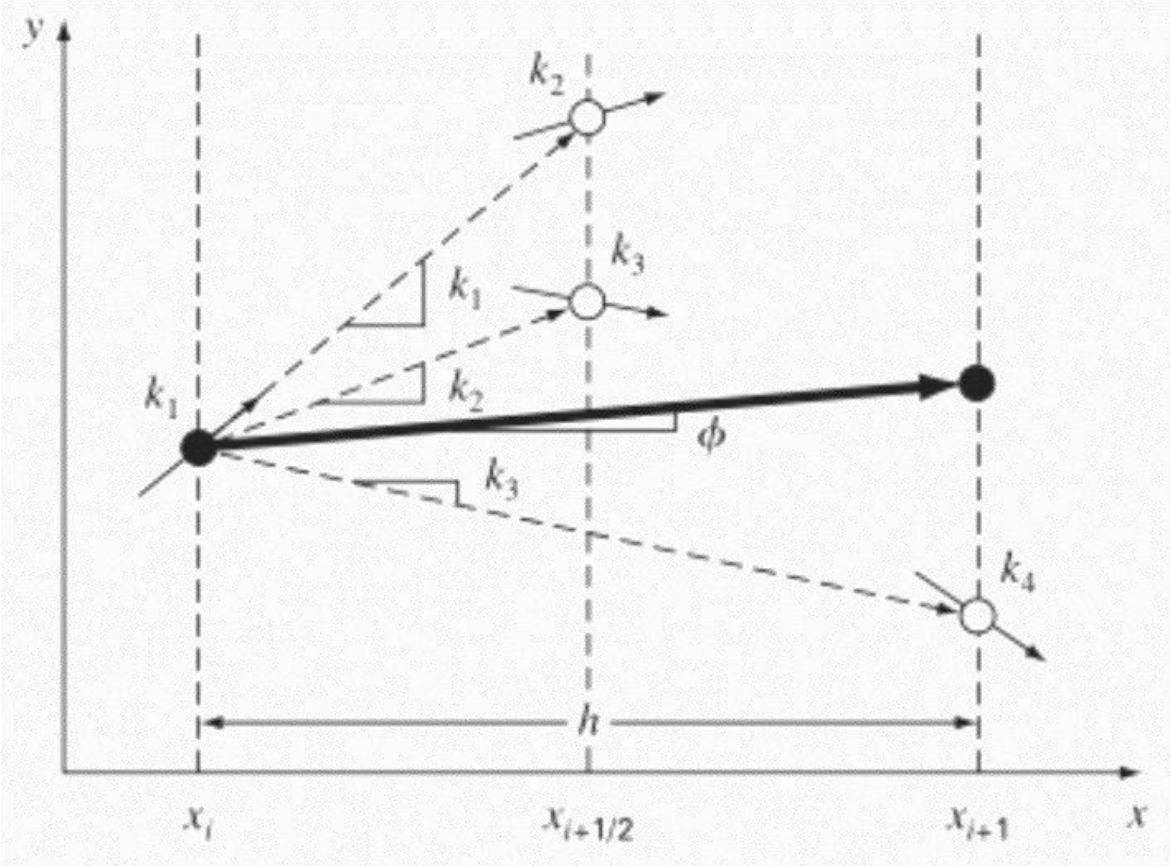

Figure 2 - Visual representation of Runge-Kutta Method (RK4) [5]

While the RK4 method is much more accurate than Euler's Method in determining the final position of the particle given the same time step, this comes at the cost of being much more computationally intensive. Not only does the computer have to calculate a large number of particle-particle interactions, as

mentioned previously, but it now has to perform four calculations per particle pair as opposed to just one. For small-scale, slow-moving simulations, this computational intensity is often too great to warrant the implementation of a Runge-Kutta method. However, for large-scale, fast-paced, and high-accuracy simulations, this trade-off in computational intensity is deemed acceptable for the significant gain in accuracy, even approaching the accuracy of symplectic integrators.

2.1.2 Implementation

The number of effective particles in the system was reduced using three primary methods to combat the previously discussed problems of exponentially increasing number of particle-particle force equations needed to be calculated for the gravitational N-body simulation. The first of these methods involved reducing the total number of particles in the system by removing unnecessary planetary bodies. Though future iterations of this program aim to include the ability for the neural network to make use of gravitational assist trajectories either using the Moon or other celestial objects, for the training segment, these solutions were deemed unlikely and overly complicated. Therefore, the only gravitationally interacting objects included in the simulation were the Sun, Mars, Earth, and the simulated spacecraft (referred to as the craft).

To further simplify the simulation and increase its speed, the Sun was considered stationary at the origin of the two-dimensional coordinate grid. At the same time, Earth and Mars were placed "on rails". Rather than assuming the traditional perfectly circular orbits at 1 and 1.524 AU for the Earth and Mars, respectively, accurate positional data over all points in the simulation were downloaded from NASA's JPL Horizons database. The positional data for the two planets were downloaded at one-minute intervals from January 1st, 2020, to December 31st, 2030, to allow for a considerable time interval over which to train the neural network. When the time step interval is selected in the UserVariables.py file, the program automatically grabs the correct data from a locally saved Horizons file for the two planetary bodies. It stores them in an easily accessible array rather than repeatedly querying and downloading the data from the JPL Horizons database. The overall result of this process is the reduction from a standard 4-body simulation to a unidirectional 4-body simulation, reducing the number of RK4 runs from six to three for each time step.

The final large-scale simplification of the simulation was implementing a patched-conics approach. This approach relies on the fact that every gravitationally interacting body has an effective sphere of influence (SOI) in which it is the dominant gravitational body. For example, in our local area around

Earth, the gravitational pull from the Sun is neglected when determining the acceleration due to gravity due to Earth's proximity. This sphere of influence for a two-body system consisting of one smaller mass object (a planet) and one larger mass object (the Sun) is approximately determined using Eq. (6) where r_{SOI} is the radius of the smaller object sphere of influence, a is the semimajor axis of the smaller object's orbit around the larger object, m is the mass of the smaller object, and M is the mass of the larger object. Applying this formula to both the Earth and Mars yield spheres of influence of around 925000 and 576000 kilometers, respectively. The number of RK4 runs was drastically reduced by only considering the gravitational forces exerted by the object that the craft was in the sphere of influence of. A patched-conics-like procedure arose out of this restriction, with the craft initially residing in Earth's sphere of influence before transitioning to the Sun's and, if the transfer went well, to Mars'. This simplification resulted in the further reduction of the unidirectional 4-body simulation to a unidirectional 2-body simulation, reducing the number of RK4 runs from three to one for each time step.

$$r_{SOI} \approx a\left(\frac{m}{M}\right)^{\frac{2}{5}} \tag{6}$$

2.2 Hohmann Transfers

2.2.1 Theory

The simplest and most energy-efficient method to transfer between two circular orbits is a Hohmann transfer orbit. This type of transfer only uses two, assumed instantaneous, engine burns. First, a craft uses an initiation burn to begin the transfer between the initial circular orbit and the desired circular orbit. The resulting trajectory of this maneuver is elliptical, touching both the original and desired orbits. When the craft reaches the radius of the desired orbit at the furthest or closest point in its elliptical transfer orbit, depending on if it is an outward or inward Hohmann transfer, it performs a circularization burn. After this circularization burn, the craft is placed into a circular orbit at the desired altitude. This transfer process is shown in Figure 3, where labels 1 and 3 are the circular orbits, and label 2 is the elliptical transfer orbit.

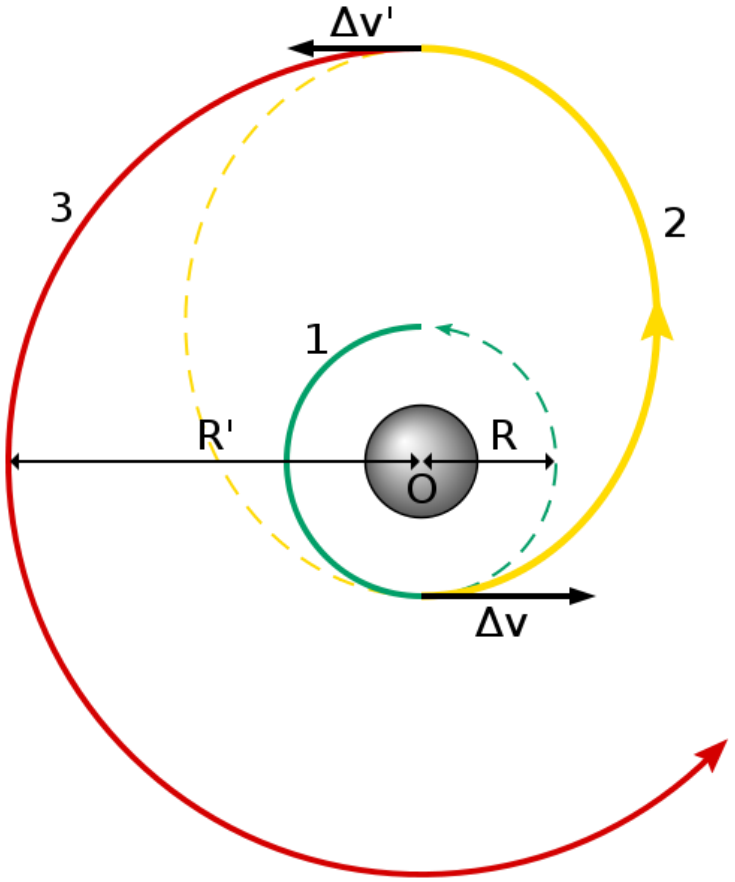

Figure 3 - Hohmann transfer between two circular orbits [6]

For a standard outward Hohmann transfer between two perfectly circular orbits, the required delta-v (Δv, the change in velocity required for the maneuver) of a given craft at each of the two burn points is given by Eq. (7). In this equation, v_{final} is the orbital velocity of the circular final orbit, $v_{orignal}$ is the orbital velocity of the circular starting orbit, $v_{t,a}$ is the orbital velocity at the apoapsis, the furthest point in the elliptical transfer orbit, $v_{t,p}$ is the orbital velocity at the periapsis, the closest point in the elliptical transfer orbit. Each of the velocities of the circular orbits is found using Eq. (8), where G is the gravitational constant, M is the mass of the body the craft is orbiting, and r is the radius of the orbit. The velocities at periapsis and apoapsis of the elliptical transfer orbit are dependent on their respective radii (r_p and r_a) and the angular momentum (h_t), a conserved quantity, of the orbit as shown in Eq. (9). Angular momentum of an elliptical orbit is calculated using Eq. (10).

$$\Delta v = \left|v_{final} - v_{t,a}\right| + \left|v_{t,p} - v_{orignal}\right| \tag{7}$$

$$v_{circular} = \sqrt{\frac{GM}{r}} \tag{8}$$

$$v_{t,p} = \frac{h_t}{r_p} \quad v_{t,a} = \frac{h_t}{r_a} \tag{9}$$

$$h_t = \sqrt{\frac{2GMr_a r_p}{r_a + r_p}} \tag{10}$$

While the basic Hohmann transfer is very simple, any variations in the transfer's parameters or desired outcome significantly increase its complexity. For instance, shifting between orbits around two different bodies, as is the case in the Earth-Mars transfer, necessitates an additional escape and insertion burn or at least modified transfer burns. The escape burn requires the craft to be put on a hyperbolic escape trajectory out of the Earth's sphere of influence. In theory, this could be accomplished more simply using a separate escape burn but combining the escape burn and transfer initiation burn into one is more practical, as lighting assumed-impulsive engines fewer times is more safe and reliable. The transfer orbit velocities are calculated in the same way as in the basic Hohmann transfer.

To calculate this more complex transfer orbit, the hyperbolic excess velocity necessary to be put on the correct trajectory towards the orbit of Mars from orbit around Earth is found using Eq. (11), where G is the gravitational constant, M is the mass of the transfer burn central body (the Sun), r_p is the radius of the periapsis of the elliptical transfer orbit relative to the Sun, and r_a is the radius of the apoapsis of this elliptical transfer orbit. This hyperbolic excess velocity is then used to find the periapsis velocity necessary for the escape and transfer using Eq. (12). In this equation, M_E is the mass of Earth, or orbiting body in the general case, while r is the original radius of the orbit around Earth. Finally, the delta-v required for this escape and transfer maneuver is calculated using Eq. (13), where v_{park} is the orbital velocity of the original orbit around the host planet, Earth, determined using Eq. (8).

$$v_\infty = \sqrt{\frac{GM}{r_p}} * \left(\sqrt{\frac{2r_a}{r_a + r_p}} - 1\right) \tag{11}$$

$$v_p = \sqrt{v_\infty^2 + \frac{2GM_E}{r}} \tag{12}$$

$$\Delta v_{trans} = v_p - v_{park} \tag{13}$$

Using this analysis, the craft is placed onto a trajectory towards the orbit of the outer planet, Mars. Once the craft arrives at the sphere of influence of the new host planet, a braking burn comparing the differences in required velocity between the desired orbit around Mars and the currently elliptical, heliocentric orbit is performed. In this case, a component-wise analysis of the burn is simpler to perform. For the first component, the delta-v required to circularize the orbit around the Sun is found using a combination of Eq. (7), Eq. (8), and Eq. (9) shown in Eq. (14). In this equation, the mass, M, is that of the Sun and the result of performing this burn prograde along the elliptical transfer orbit is the placement of the craft into a circular orbit around the Sun at the orbital radius of Mars. Finally, the circularization burn around Mars, assuming the burn occurs at the SOI of Mars, is found using Eq. (15) where M_M is the mass of Mars, r_{SOI} is the radius of the SOI of Mars, and v_M is the orbital velocity of Mars with respect to the Sun. The total delta-v for this combination heliocentric circularization and Mars circularization burns is labeled as $\Delta v_{insertion}$ in Eq. (16).

$$\Delta v_{helio} = \sqrt{\frac{GM}{r_a}} - v_{t,a} \tag{14}$$

$$\Delta v_{circ} = \left| \sqrt{\frac{GM_M}{r_{SOI}}} - (v_{t,a} - v_M) \right| \tag{15}$$

$$\Delta v_{insertion} = \Delta v_{helio} + \Delta v_{circ} \tag{16}$$

2.2.2 Implementation

Some key assumptions were made to apply the previously discussed theoretical circular-circular Hohmann transfer to the more accurate elliptical-elliptical Earth-Mars system presented in the simulation. Primarily, the elliptical orbits were assumed to be circular, with the average orbital radii of the Earth and Mars being $149.596 * 10^6$ kilometers and $227.293 * 10^6$ kilometers, respectively. While not entirely accurate in the Horizons-based N-body simulation, these

circular orbit approximations provided a strong launching point for the Hohmann transfer calculations.

When determining the actual delta-v needed for the initial transfer, Δv_{trans}, the timing of the launch was key. Earth-Mars Hohmann transfers take around nine months to complete, and launch windows occur about every twenty-five months. Given this launch window time difference and the upcoming August 2022 launch window, the next Hohmann transfer window was determined to be the end of September 2024. The control Hohmann transfer was tested against using the N-body simulation using this window timing, and it was found that the originally calculated total transfer delta-v of 2080 m/s was not sufficient for the craft to reach the SOI of Mars. Using the actual positions of Earth at the start of the transfer window and the orbital parameters of Mars at the end of the nine-month transfer, the transfer delta-v was calculated to be 2443 m/s. With this new value, the craft was able to pass into the SOI of Mars and perform a 2547.8 m/s insertion burn to circularize its orbit around Mars. It is important to note that these values hold only for the specific September 2024 window due to transfer windows occurring at different points on the elliptical orbital paths of the planets. For example, a transfer occurring during the mid-November 2026 window will require slightly different delta-vs to reach the SOI of Mars as Earth and Mars are moving at different speeds and are at different points in their elliptical orbits when compared to September 2024.

2.3 Neural Networks

2.3.1 Theory

When advanced computing is discussed, those seeking to mimic aspects of biology such as evolution and even the human brain are the first to be discussed. Combining these systems into one, neural networks have become the gold standard for creating artificial intelligence where extensive training data is not necessarily available, an area in which machine learning algorithms are impossible to implement.

The key to understanding neural networks is realizing how they seek to mimic a brain. Just as the brain has several different types of neurons for inputting information, making decisions, and outputting actions, so does a neural network. Where a human or other animals would have sensory neurons for information input, the neural network has its input layer, shown in green in Figure 4. Where an animal has motor neurons for communicating with muscles to take actions, the neural network has its output layer, shown in yellow in Figure 4. These inputs and outputs can consist of any type of data the designer sees fit. However, the input information is most effectively used for simulation

parameters such as positional and velocity data. Output information typically takes the form of the neural network somehow modifying an object in that simulation. For example, a commonly trained neural network seeks to vertically balance a pole on a hand just as a person may try to balance a broom. The input to this neural network is the broom's current position, velocity, and acceleration, while the output would be the position to move the hand to prevent the broom from falling.

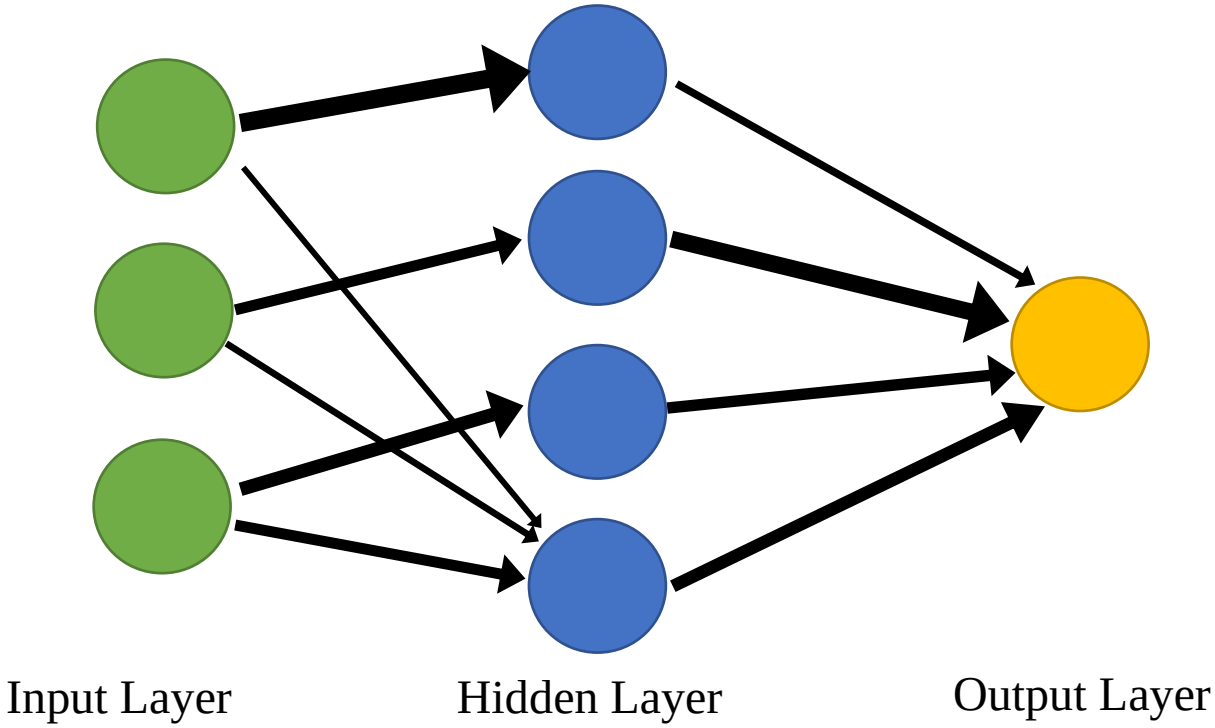

Figure 4 - Basic neural network topology with input, hidden, and output layers

In between sensory and motor neurons are interneurons that effectively work to make the decision from the input and relay it to the output. In neural networks, these are organized into the hidden layer shown in blue in Figure 4. The hidden layer does not consist of simple data input or output signals as in the other layers but rather of mathematical functions of weights and biases. When an input node is connected to a node in the hidden layer, the number associated with this input node, the tip of the broom's x-position on an x-y coordinate plane, for example, is multiplied by the hidden node's weight and its bias is added to it. In the example, if the broom tip's x-position is 0.7 and the hidden node has a weight of 6.2 and a bias of 0.1, the resulting output from the hidden node is $0.7 * 6.2 + 0.1 = 4.44$. This number may directly connect to the output layer, making the hand move 4.44 units, or it may be connected to another node in the hidden layer, further refining the decision before sending it to the output layer. In this

way, the neural network takes in numerical data, applies mathematical functions to it, and presents a numerical output.

When a neural network is created, the connections between the layers and the weights and biases of the hidden nodes are randomized or set by a defined function. The performance of such a neural network is typically very poor and needs to be improved over time. This improvement is accomplished by using a genetic algorithm, a computational process that mimics the process of evolution and natural selection. When a new neural network is created, it is tested within the simulation in which it is intended to operate. Based on its performance in this simulation, it is assigned a number known as its fitness value. After all generated neural networks are tested, their fitness values are compared, and those with the lowest fitness scores are removed from the population. The remaining members of the population are either allowed to breed, sharing some characteristics of their networks, or reproduce on their own with a fixed mutation rate. Over time, this will theoretically increase the average performance of the neural network population and make it better at performing the assigned task within the simulation environment.

A variation on this standard neural network creation and evolution process is NeuroEvolution of Augmenting Topologies (NEAT). The primary feature of NEAT that makes it perform better in certain situations is its ability to not only change the weights and biases of nodes through the evolution process but also to change the number of hidden nodes and their connections. While the topology, the structure of the layers, of a typical neural network is fixed, those evolved using NEAT can change it over time, allowing the AI to either add or remove nodes and connections as is beneficial to its performance. In theory, this allows for the evolution of very complex neural networks, even with the initial population having no hidden nodes. This approach has several benefits, including removing the need for the designers of a program to determine the correct number of hidden nodes. While the rule-of-thumb for this process is to take the average of the number of input and output nodes, this is a pure guess that seeks to strike a balance between complexity and ability. Instead of following guesswork, the AI will perform this work itself, undergoing the same neuroevolution process seen in biology. However, one key downside to this approach is the time it takes to train the neural network. Since more variables are involved in the evolution process, training can take several orders of magnitude longer. This trade-off must be considered when determining if NEAT is the correct approach to a problem.

2.3.2 Implementation

After considering the wide range of evolution schemes for neural networks, NEAT was chosen for its removal of the issue of setting an initial number of hidden nodes that remain static over time. To implement this scheme into the Python-based N-body simulation, the NEAT Python module was used due to its simplicity compared to constructing a NEAT scheme from scratch. Implementing the NEAT files into the simulation framework necessitated the introduction of several new variables used to modify the evolution parameters. Instead of relying on a single configuration file, these files were split into the UserVariables.py, Constants.py, and config-feedforward.txt files visible in Figure 1.

The second key requirement for the implementation of the neural network framework was the construction of a fitness function. This function was used to evaluate the performance of the neural network and the control Hohmann transfer. Three initial key factors were included in the fitness function: the delta-v, distance, and time. The delta-v factor was a measure of the total delta-v used by the craft and was compared to the control delta-v to generate the delta-v score, as shown in Eq. (17). The distance factor measured the closest the craft was to Mars on its transfer path and was compared to the control closest distance to generate the distance score, as shown in Eq. (18). A logarithmic function was chosen for its asymptotic behavior to ensure that the score could not become negative while the gradient increased as the craft approached Mars. Finally, the time factor was added as a measure of time it took for the craft to reach its closest point to Mars and was compared to the time the Hohmann transfer took to reach its closest point, as shown in Eq. (19).

$$score_{\Delta v} = \frac{\Delta v_{control}}{\Delta v} \tag{17}$$

$$score_{dist} = log_{10}\left(\frac{closest_{control}}{closest} + 0.01\right) + 3 \tag{18}$$

$$score_{time} = \frac{time_{control,closest}}{time_{closest}} \tag{19}$$

After testing the neural networks trained by this fitness function, several additions and modifications were made to produce a high-functioning network consistently. The first of these was the modification of the time function from measuring the time to the closest approach to measuring the time to reach the SOI of Mars. This change was in response to behavior that the neural networks

favored, such as moving directly away from Mars. This exploit resulted in its closest point being measured as the initial point, increasing the value of its time score to the detriment of the others. This new time score formula is shown as Eq. (20). Another issue was identified with the distance score, where the neural networks began prioritizing the behavior of getting as close to Mars as possible but overshooting it significantly. This issue was remedied by including another score: the final distance score. This score measured the craft's final distance from Mars and was compared to the final distance from Mars for the Hohmann transfer, as shown in Eq. (21). Another addition was a factor pertaining to the craft's relative speed at the SOI (RSSOI) of Mars. This RSSOI factor was in response to the neural networks developing a behavior where they would have the craft move very quickly through the SOI of Mars and therefore were not able to circularize its orbit with the remaining delta-v. The calculation for this score is shown in Eq. (22). It should be noted that the RSSOI score is used internally by the neural network to determine the fitness between generations but is not used in the final data analysis when comparing to the control Hohmann transfer.

$$score_{time} = \frac{time_{control,SOI}}{time_{SOI}} \tag{20}$$

$$score_{f-dist} = log_{10}\left(\frac{dist_{final,control}}{dist_{final}} + 0.01\right) + 3 \tag{21}$$

$$score_{RSSOI} = \frac{RSSOI_{control}}{RSSOI} \tag{22}$$

With the scores established, a weighting system to compare the scores was created. This system took the form of a simple weight for each of the scores that was multiplied by the scores before they were summed to determine the final fitness, as shown in Eq. (23). In this equation, each of the scores is represented by $score_i$ while its respective weight is shown as $weight_i$. The key advantage to this score and weight system is the ability to easily modify the weights when experimenting with the fitness function. With only five numbers to tune, the testing matrix became simpler than modifying the score functions themselves or attempting to make changes blindly. If one particular feature, such as the approach distance, was being consistently left behind, an increase in the value of $weight_{distance}$ was able to remedy this problem. The final factor, shown in Eq. (23), is m_{circ}: the circularization multiplier. Even with the addition of the RSSOI score, the neural network was often prioritizing behavior that saw it pass through the SOI without the ability to circularize its orbit with its remaining fuel. Therefore, a factor that rewarded the ability to circularize was introduced and

calculated using Eq. (24). In this equation, g_{curr} is the current generation, g_{max} is the maximum number of training generations, and m_{max} is the maximum circularization multiplier. This scaling ability with the generation ensures that the behavior of circularizing a Mars-centric orbit in its SOI is prioritized later in the evolution of the neural network, allowing time for other key features such as delta-v optimization to occur first.

$$fitness = m_{circ} * \sum score_i * weight_i \tag{23}$$

$$m_{circ} = 1 + \frac{g_{curr}}{g_{max}} * (m_{max} - 1) \tag{24}$$

One important factor considered during the implementation of the neural network framework into the code was the frequency at which the neural network was given information about the simulation. In fast-paced, multiple action simulations such as the previously mentioned broomstick balancing example, data is constantly fed to the neural network. The neural network makes decisions in the moment, based on its topology, on whether or not to take action. In other scenarios where behavior is more consistent, and the inputs into the network do not directly depend on the network's output, data may only be given intermittently or even only the initial state's data given to determine actions taken by the network. The latter approach was initially taken in implementing the neural network framework into this project. More specifically, the data input into the neural network consisted only of the craft, Earth, and Mars positions and velocities at the initial state. Since the orbital speed of Earth and Mars are roughly constant along their paths, the neural network was thought to be able to garner all necessary information from this initial state after some training on the data set.

However, this initial assumption was quickly proven incorrect, as the neural networks that were only given access to the initial data converged to far less optimal solutions than the control. A new version of the code was designed to feed the neural networks information about the craft and other bodies at every time step to alleviate this performance issue. Instead of deciding the duration and timing of the burns at the starting point, a total of six outputs from the network, the new, live data version only produced three outputs. These outputs are whether to execute a burn at the current time step, the burn's angle, and the burn's delta-v. While increasing the algorithm's complexity slightly, the performance of the live version compared to the initial data version, and even compared to the control, was superior and is discussed below.

To feasibly train a complicated neural network from the ground up using NEAT, taking advantage of all of the local computing power available is necessary. Rather than running a single training simulation at one time as was initially done, multiple individuals were trained at once using multiprocessing. The Python Multiprocessing module allows for a program's tasks to be split among all of the cores available on the local CPU. This practice is primarily applicable in heavily paralyzed workloads, meaning the result of one run is independent of the result of another. Since the simulation of one individual in a population during the neural network training is completely independent of the other members of that population, this workload is considered almost entirely parallelized. In the end, this practice meant that twelve individuals could be trained at one time, drastically increasing the training speed of the neural network. This increased training speed made the entire process more efficient and meant that more individuals could realistically be trained for more generations in the same time frame.

3 Methods – Testing Matrix

In order to test the overarching question of whether a neural network evolved using NEAT could effectively solve orbital transfer problems, the optimal values of each of the three main varying-weight parameters, RSSOI, delta-v, and transfer time, needed to be determined for different desired behaviors. To accomplish this goal, a testing matrix was developed that varied each parameter from 0 to 1 in intervals of 0.25. These tests were divided into three main groups, one for each possible mission profile that certain combinations of these parameter weights could be looking to solve.

Table 2 - Testing matrix for Cargo Variant, grey indicates constant

Cargo Variant Parameter Weights				
RSSOI	Time	Delta-V	Distance	Circularization Multiplier
0.25	0	0.5	1	3
0.5	0	0.5	1	3
1	0	0.5	1	3
0.25	0	0.75	1	3
0.5	0	0.75	1	3
1	0	0.75	1	3
0.25	0	1	1	3
0.5	0	1	1	3
1	0	1	1	3

The first of these is the Cargo Variant group shown above in Table 2. This variant is the most similar to the Hohmann transfer and can be directly compared to it. These tests

optimize for delta-v and would be applicable in any situation where the transferable payload mass was deemed more important than the time it took for the payload to arrive, something that is more applicable to cargo rather than human loads. These tests involved keeping the time parameter weight at a constant of zero, not allowing it to impact the fitness that was derived solely from the amount of delta-v that the craft consumed on its trajectory.

Table 3 - Testing matrix for Human Variant, grey indicates constant

Human Variant Parameter Weights				
RSSOI	Time	Delta-V	Distance	Circularization Multiplier
0.25	0.5	0	1	3
0.25	1	0	1	3
0.5	0.5	0	1	3
0.5	1	0	1	3
0.75	0.5	0	1	3
0.75	1	0	1	3
1	0.5	0	1	3
1	1	0	1	3

The second main group is the Human Variant group shown above in Table 3. The Human Variant tests optimize for transfer time rather than delta-v as in the Cargo Variant. While the amount of delta-v must still be within the total possible delta-v for the tested craft, its weight when determining the fitness score is zero for all of these tests. While this variant has a very different behavior from the Hohmann transfer, it is still useful to compare the two to identify the time savings possible with different orbital trajectories.

Table 4 - Testing matrix for General Variant, grey indicates constant

General Variant Parameter Weights				
RSSOI	Time	Delta-V	Distance	Circularization Multiplier
0.25	0.5	0.5	1	3
0.5	0.5	0.5	1	3
0.75	0.5	0.5	1	3
1	0.5	0.5	1	3
0.25	1	1	1	3
0.5	1	1	1	3
0.75	1	1	1	3
1	1	1	1	3

The last of these groups is the General Variant group shown above in Table 4. Rather than optimizing for only delta-v or time, this group optimizes for a combination of both. This group comprises the rest of the tests that do not fall into either of the previous schemes. While the desired mission profile will cause the time and delta-v weights to be modified, each will still likely be a factor in the transfer optimization. This General Variant group allows for a deeper analysis of how neural networks can optimize for certain types of transfers more effectively than classic closed-form solutions.

Three trials were run for each parameter weight set to complete these testing matrices. Each run consisted of 100 generations of 72 individuals trained on the same starting date and relative position as the control Hohmann transfer. The average and maximum fitness of these three trails were recorded and normalized against the Hohmann transfer fitness for the given parameter weights for later analysis. The tests were run for a craft starting at an orbital radius of 127200 kilometers around Earth on September 30th, 2024, corresponding to a Hohmann transfer window. The craft selected for this analysis, though the framework exists for any craft to be used, was the Dawn spacecraft. Given the specific impulse of its ion engine, which was assumed to be impulsive for the sake of simplicity, and the onboard fuel mass, the delta-v available was calculated to be around $14000\,\frac{m}{s}$.

4 Results/Analysis

4.1 N-Body Simulation Validation

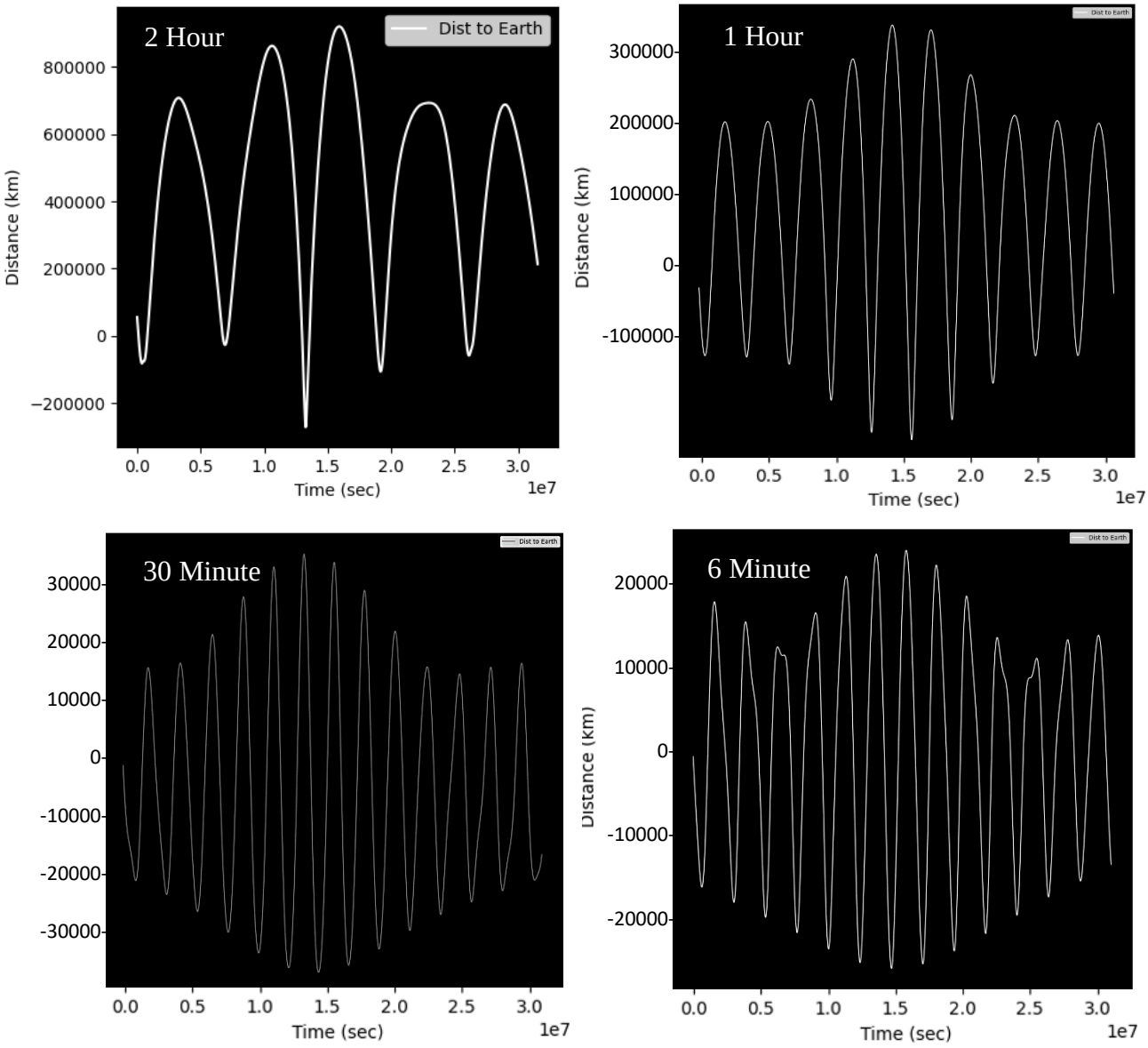

Figure 5 - Variation of geocentric orbit at orbital radius of the Moon using 2 hour, 1 hour, 30 minute, and 6 minute time steps

The N-body simulation was validated using studies of stable orbits around Earth and the Sun. Each orbit was run for one year, and the variation from the expected, perfectly circular orbit was measured. Figure 5 above displays the results when using two hour, one hour, thirty-minute, and six-minute time steps to simulate a body placed in an ideally circular geocentric orbit at the orbital radius of the Moon (384000

kilometers). In this figure, the distance variation from the ideal orbit is measured on the vertical axis while the time, in seconds, is measured on the x-axis.

As seen by the magnitude of the variation of each of these tests, shorter time steps yield more precise and less varying orbits. The maximum variation when a time step of two hours was used was over 800000 kilometers, twice the orbital radius, and therefore an extremely inaccurate simulation. As the time step was decreased by half, the variation was decreased by more than half, resulting in a 300000 km variation when a time step of one hour was used. A ten-times reduction in variation was achieved when shrinking the time step from one hour to thirty minutes. From here, returns were diminishing, with a five times decrease in step size from thirty to six minutes leading to less than a two times decrease in orbital variation.

Ultimately, the simulation was considered valid for time steps less than ten minutes as the orbital variation was typically less than five percent of the orbital radius. Analysis of time steps as low as one minute was performed, with the variation hovering around this five percent mark, indicating some minor inaccuracy in the simulation or experimental setup or the presence of orbital resonance of the body with the gravitational influence of the Sun. For experimental purposes, a time step of six minutes was selected to keep variations small while allowing for six times faster processing when compared to using a one-minute time step.

4.2 Control Results

The generated control orbit was selected to take the form of a Hohmann transfer and consisted of two burns. The first combined the escape from Earth's gravitational influence with the insertion burn into the elliptical transfer orbit. This burn used a total of $2443\ m/s$. The second burn, in contrast to a typical Hohmann transfer, circularized the craft's path relative to Mars rather than to the Sun. This circularization occurred such that the craft was placed into a stable circular orbit at the sphere of influence of Mars. This burn used $2547.8\ m/s$ due to the nearly tangential trajectory relative to Mars' SOI shown in Figure 6. The craft uses a total of $4990.8\ m/s$ of delta-v and takes a total of 279 days to reach the sphere of influence of Mars. Since the scores used to calculate the neural networks' finesses are normalized using these values, each score for the control is equivalent to 1. The overall fitness of the control is simply the sum of the weights multiplied by the circularization multiplier. This control fitness is used when analyzing the fitness of the neural networks to provide a point of normalization and comparison. For a point of visual comparison, the transfer trajectory

shown in the left image of Figure 6 as the white line is extremely smooth. The lack of sharp turns in the craft's trajectory is one indicator of a low delta-v transfer.

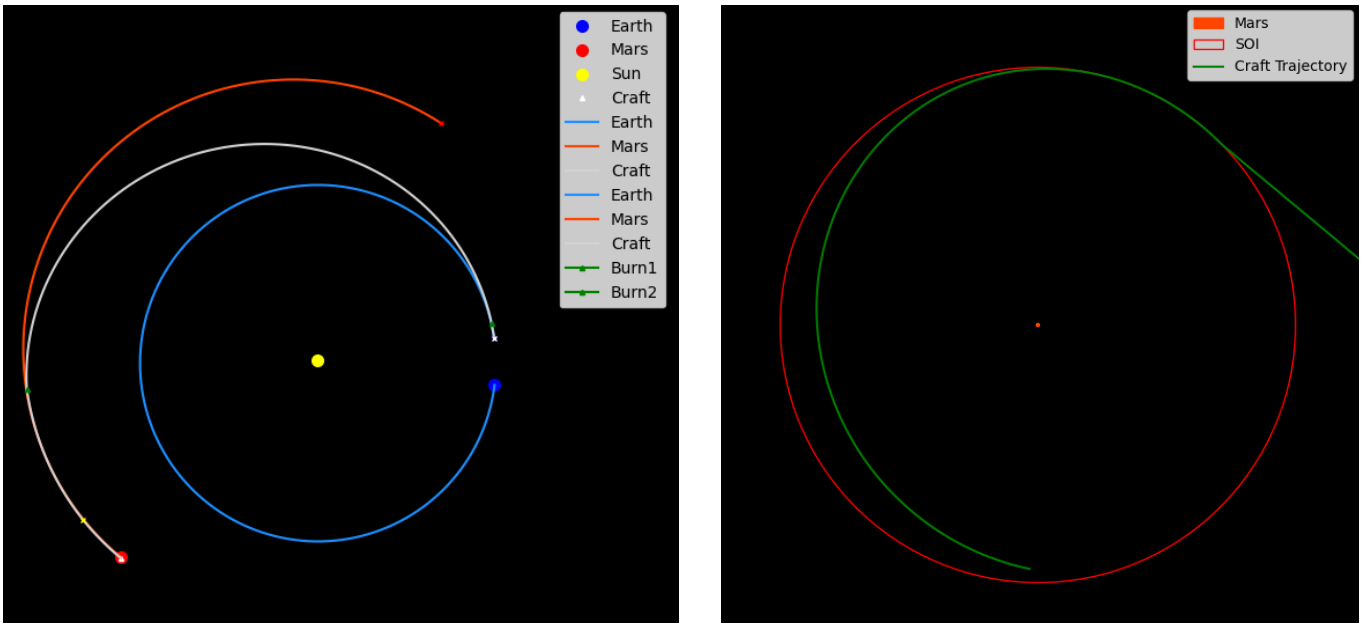

Figure 6 - Control Hohmann transfer trajectory and circularization path around Mars

4.3 Testing Matrix Results

Note: RSSOI score is not included in the reported fitness

Table 5 - Cargo Variant testing results data and normalized analysis

Cargo Variant Testing Results									
RSSOI	Time	Delta-V	Distance	Circ. Mult.	Control Fitness	Avg Fitness	Best Fitness	Avg Norm. Fitness	Best Norm. Fitness
0.25	0	0.5	1	3	4.5			0	0
0.5	0	0.5	1	3	4.5	3.30	4.05	0.73	0.90
1	0	0.5	1	3	4.5			0.00	0.00
0.25	0	0.75	1	3	5.25			0.00	0.00
0.5	0	0.75	1	3	5.25	3.81	4.59	0.73	0.87
1	0	0.75	1	3	5.25			0.00	0.00
0.25	0	1	1	3	6			0.00	0.00
0.5	0	1	1	3	6	5.12	5.14	0.85	0.86
1	0	1	1	3	6	3.57	5.11	0.59	0.85

Table 6 - Human Variant testing results data and normalized analysis

Human Variant Testing Results									
RSSOI	Time	Delta-V	Distance	Circ. Mult.	Control Fitness	Avg Fitness	Best Fitness	Avg Norm. Fitness	Best Norm. Fitness
0.25	0.5	0	1	3	4.5	3.12	4.90	0.69	1.09
0.25	1	0	1	3	6	7.85	8.20	1.31	1.37
0.5	0.5	0	1	3	4.5	2.21	2.22	0.49	0.49
0.5	1	0	1	3	6	5.19	7.51	0.87	1.25
0.75	0.5	0	1	3	4.5	1.68	2.22	0.37	0.49
0.75	1	0	1	3	6	3.45	3.46	0.58	0.58
1	0.5	0	1	3	4.5	2.25	2.29	0.50	0.51
1	1	0	1	3	6	3.46	3.46	0.58	0.58

Table 7 - General Variant testing results data and normalized analysis

General Variant Testing Results									
RSSOI	Time	Delta-V	Distance	Circ. Mult.	Control Fitness	Avg Fitness	Best Fitness	Avg Norm. Fitness	Best Norm. Fitness
0.25	0.5	0.5	1	3	6	5.94	5.95	0.99	0.99
0.5	0.5	0.5	1	3	6	5.95	6.00	0.99	1.00
0.75	0.5	0.5	1	3	6	5.96	6.03	0.99	1.00
1	0.5	0.5	1	3	6	5.99	6.12	1.00	1.02
0.25	1	1	1	3	9	8.90	8.93	0.99	0.99
0.5	1	1	1	3	9	7.34	9.35	0.82	1.04
0.75	1	1	1	3	9	5.46	8.87	0.61	0.99
1	1	1	1	3	9	8.88	8.92	0.99	0.99

4.4 Variant Analysis

Each of the three variants was tested using the testing matrices found in Section 3 and completed in Section 4.3. These tests were performed to determine the optimal relative weights of the RSSOI score and the parameters of interest (some combination of time and delta-v) as well as to analyze how the neural networks adapt to different mission profiles. The trends of the performance of each of the variants, when compared to the control, are discussed below.

4.4.1 Cargo Variant – Fuel-Optimized

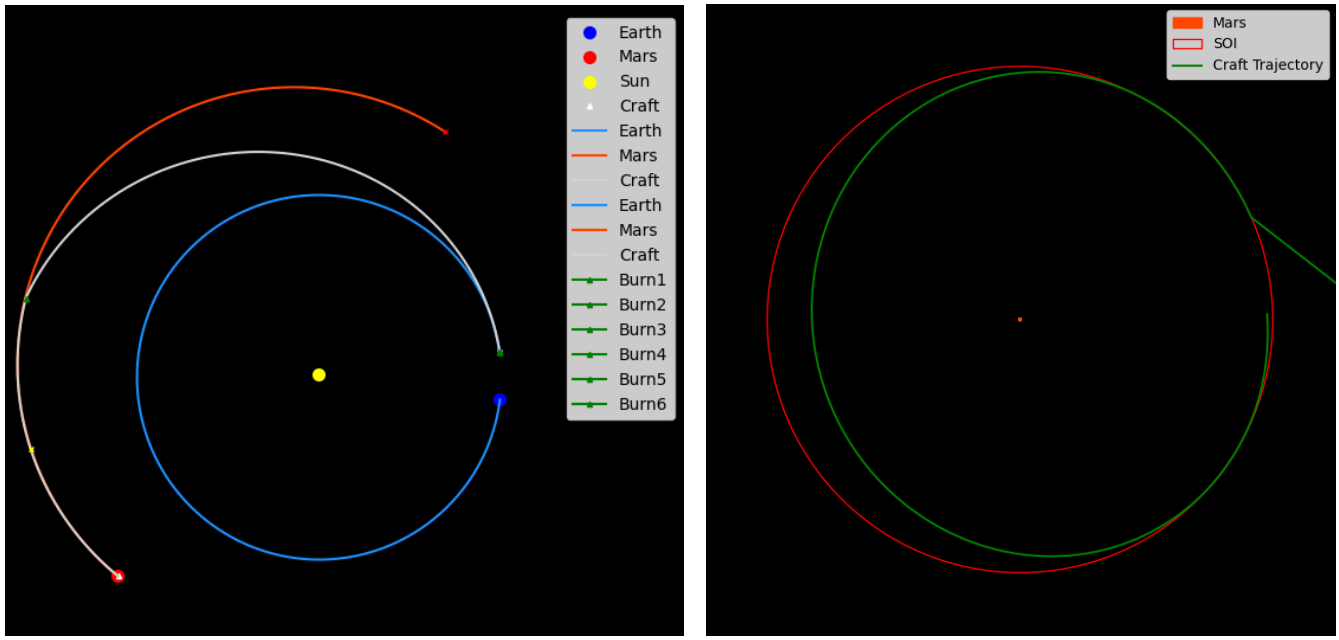

Figure 7 - Cargo Variant best trajectory and Mars circularization path

When tasked with optimizing for delta-v, the neural networks produced solutions that consistently underperformed relative to the Hohmann transfer. The highest normalized fitness achieved over any of the runs was only 0.90, corresponding to a total delta-v of around $7129\frac{m}{s}$ compared to the Hohmann transfer's $4991\frac{m}{s}$. When analyzing the components of the transfer, this often corresponded to comparable escape burn delta-vs, while the insertion and circularization burns required significantly higher delta-vs. This pattern is likely due to the delicate relationship between the escape delta-v and the circularization delta-v. Through manual testing, it was found that shifting the escape delta-v by around $10\frac{m}{s}$ led to hundreds of additional meters per second of delta-v being needed for the circularization burn. Since a tangential velocity vector relative to the SOI of Mars leads to the smallest possible required circularization burn, any deviation from this tangential vector as used by the Hohmann transfer requires additional delta-v. This departure from the tangential velocity vector relative to the SOI is shown on the right side of Figure 7. The left image in Figure 7 also shows a more abrupt craft trajectory (indicated by the white line) change around at the location of the sixth burn when compared to that of the smooth Hohmann transfer.

It should be noted that if the tests were to be configured such that the craft executed a circularization burn when it was at the tangential point to any potential circular orbit around Mars, the results of this study would likely be different. Overall, due to the limitations of circularization at the RSSOI and the

Hohmann transfers low delta-v usage, the program could not develop a neural network that found a better solution than the control given the desired parameter weights.

4.4.2 Human Variant – Time-Optimized

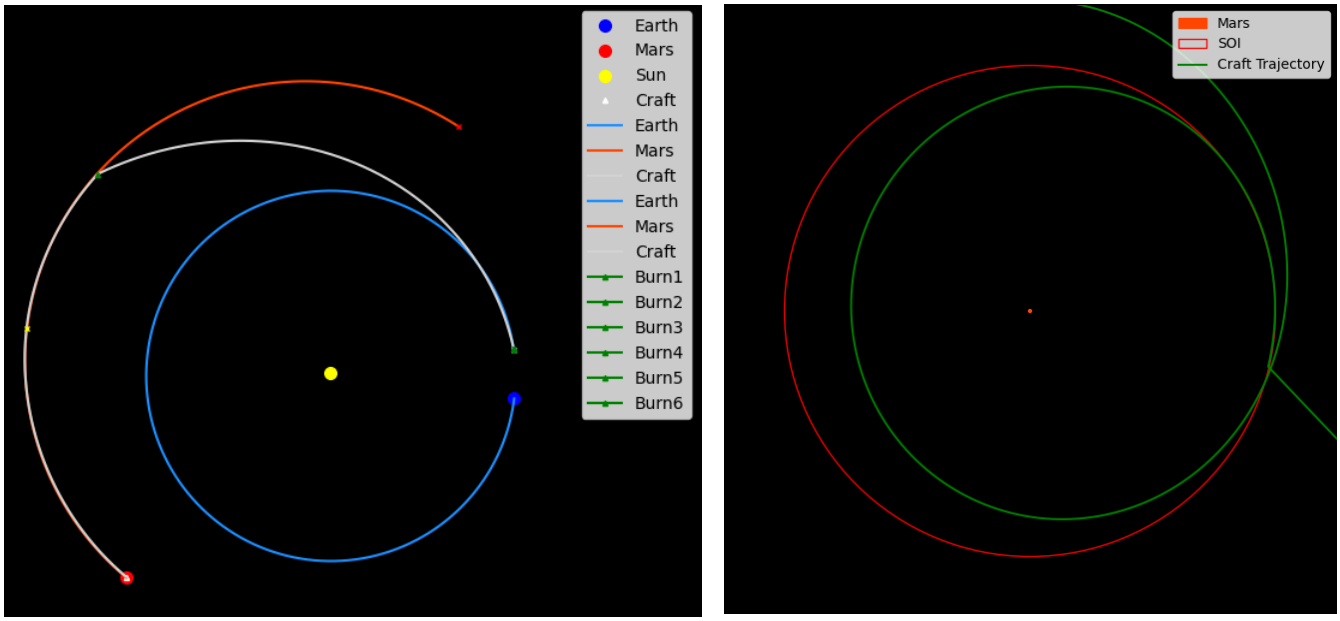

Figure 8 - Human Variant best trajectory and Mars circularization path

After gathering the data using the testing matrix for the Human Variant in

Table *6*, several key trends became apparent. Overall, given the input parameter weights, the output consisted of three schemes that produced a best fitness greater than the control. Since these were human and therefore time-optimized trials, three of the solutions presented faster transits than the Hohmann transfer's 279 days. The speed of this transit can be seen in the left image of Figure 8 by the earlier interception point of Mars compared to the Hohmann transfer. Additionally, the lack of consideration of delta-v optimization beyond only having access to limited but relatively abundant fuel, results in a non-tangential intersection of the craft's trajectory and the SOI of Mars, as shown on the right side of Figure 8. Interestingly, these more effective transfers are clustered near the top half of the table, where the RSSOI weights are the lowest.

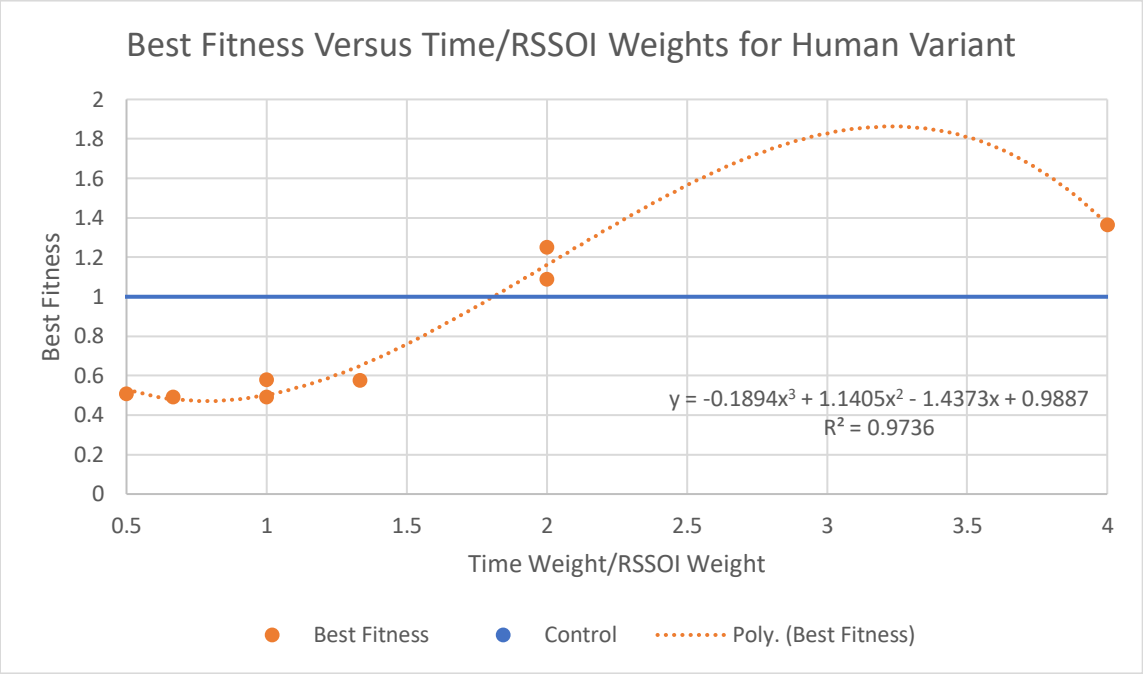

Figure 9 - Best fitness versus time to RSSOI weight ratio for Human Variant tests. Hohmann transfer fitness (1) is indicated by the horizontal blue line

Another way of viewing this data is that when the ratio of the time weight to the RSSOI weight is low, the best fitness produced by the scheme is low and when the ratio is high the fitness increases. The result of plotting the best fitness produced by the given tests and the ratio is shown in Figure 9. In this plot, it can be seen that as the time to RSSOI weight ratio increases, so does the best fitness. Based on the limited gathered data, it appears to follow a roughly third-order polynomial relationship. This trend could be explained by the fact that the RSSOI score is tied to the delta-v of the system. The purpose of the RSSOI score is to lead the neural networks toward solutions where circularization is possible. When looking to reach the SOI of Mars as fast as possible, however, the RSSOI score may push the neural network where the speed at the SOI is weighed heavier than the target parameter for optimization: the transfer time. When the time to RSSOI weight ratio is high enough, however, the networks are incentivized to still optimize for time, even to the detriment of the RSSOI score. An example of this is when the ratio is 4, the time weight dominates, and the best-produced individual has a normalized fitness of around 1.37. When the RSSOI score dominates, however, as is the case when the ratio is 0.5, the produced best fitness is only 0.51.

In future iterations of this project, it would be beneficial to experiment with removing the RSSOI score entirely. While this could result in drastically

underperforming individuals due to non-circularizable solutions becoming more common, it could also result in circularized orbits that have much lower transfer times. Another possible solution to this issue would be to use multistage training. This training scheme would have the starting generations trained with a high RSSOI weight until there is a high presence of circularized solutions, and then the RSSOI weight would drastically decrease. This would increase the dominance of the time weight score over the generation while the circularization multiplier would still ensure that circularized solutions, which have already been developed, remain. A possible downside to this approach is that this would optimize for the first local minimum point found. Essentially, once a circularizable trajectory was found, and the training scheme switched to time optimization, the neural network would never be able to break out of the basic trajectory it was using, meaning only minor optimizations could be made.

4.4.3 General Variant – Time and Fuel-Optimized

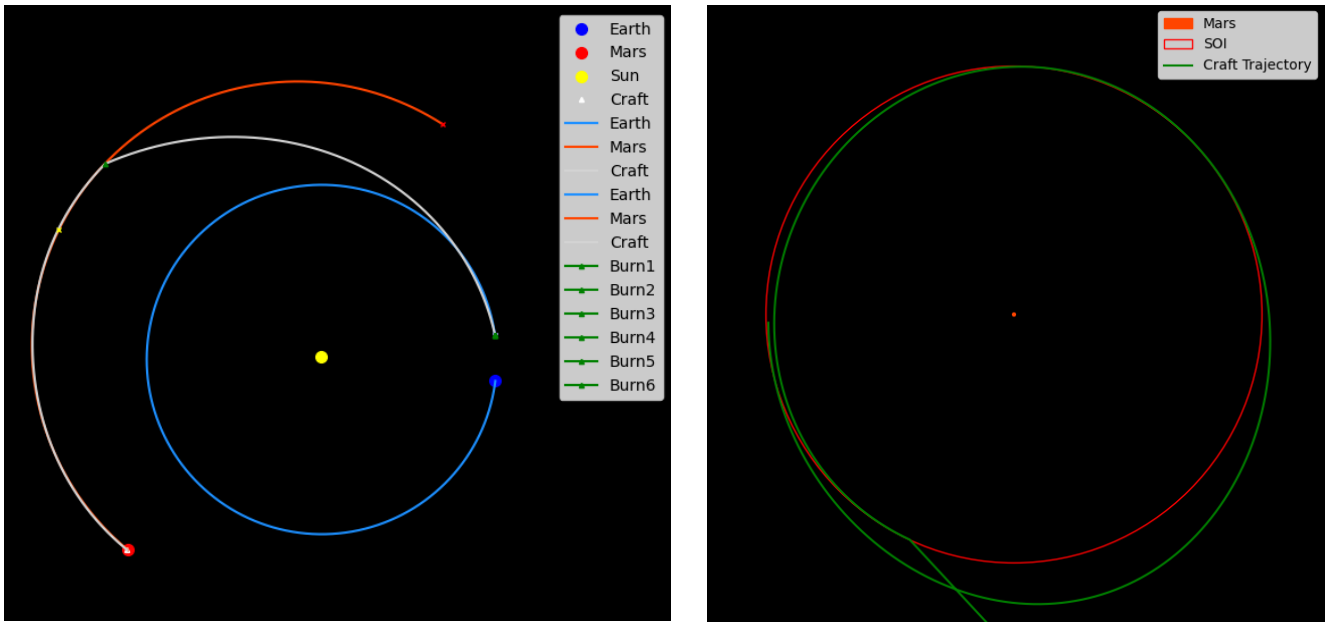

Figure 10 - General Variant best trajectory and Mars circularization path

The batch of tests referred to as the General Variant centered around a scenario where time and delta-v were considered equally important. This variant also established a framework for working with scenarios where both parameters must be considered but in different weights. Out of the variety of tests outlined in Table 4 and Table 7, the best performing neural network is displayed visually above in Figure 10. This figure shows that the craft reached Mars in a much shorter time than in the Hohmann transfer and approached at a less tangential trajectory to the SOI. This indicates that the program optimized more for transit time than the

delta-v used. The best performing neural network scores confirm this visual indication as the time score achieved was around 1.77 while the delta-v score was only 0.36. It appears that this individual optimized for the variable that was simpler to optimize for, as indicated by the program's performance in the Human Variant and Cargo Variant analyses. This trend holds for other individuals produced by the General Variant tests and has a number of possible solutions to enable a more even optimization.

The simplest of these solutions would be to increase the delta-v weight relative to the time weight until both are consistently optimized in a more balanced way. This would require another set of parametric studies where the ratio of the delta-v weight to the time weight was varied, and the time and delta-v scores were recorded. Another possible solution would be to set caps on the scores for one or more of the parameters. For example, the time score could be capped at 1.5, meaning any improvements in the transit time would not be rewarded but improvements in the delta-v while maintaining a sufficiently high transit time would be. This would likely lead to more balanced solutions but to longer transit times than necessary for a given delta-v score. The delta-v available to the program could also be reduced using the craft properties section of the UserVariables.py file, forcing the craft to indirectly optimize for delta-v while still working to improve its time score.

Overall, despite the repeated instances of the normalized fitness scores being greater than one, the neural networks chose to optimize for a single parameter while excluding the other, leading to unbalanced solutions. However, these solutions were still with the craft's capabilities and therefore present viable orbital trajectories given the mission profile.

5 Future Improvements

Considering the limited scope of this project, several avenues of study remain available. Using the robust framework developed throughout this project, numerous further improvements could be made to enable these areas of study and explore the feasibility of using neural networks to solve orbital transfer problems.

5.1 Array Training

In order to ensure that the neural network is not just learning the nuances of a specific initial start date, further training should be carried out over a variety of dates. A function is already built into the code that accomplishes this by running "array training". This training has each member of each generation trained on several specified training dates. For example, to see if the neural networks can develop the Hohmann transfer given the initial planetary ephemerides, it could be fed different Hohmann transfer dates ranging in time from March 2021 to June 2025. The network

would be trained over this date range, and its final fitness score for comparison to the Hohmann transfer would be taken from a simulation run at the same start date as the Hohmann transfer in September of 2024. This avenue of training was explored, but the training time for a fixed number of generations linearly increases with an increasing number of dates in the training array. Furthermore, the complexity of this problem from the perspective of the neural network means that this type of training would likely necessitate much larger populations and many more generations to train.

One possible solution to this problem would be to utilize a non-local compute cluster such as the High-Performance Computing (HPC) cluster at UConn. Such clusters allow for a large throughput of parallelized data to be processed simultaneously. This would significantly decrease the training time and allow many more individuals to be simulated for many more generations in the same physical time span.

5.2 Craft and Planet Modification

A single case study of the Earth-Mars transfer and a single craft, the Dawn spacecraft, was used to validate the concept of using NEAT to develop neural networks to solve orbital transfer problems. In future iterations of this project, each of these should be modified and expanded to determine if the current program can develop effective solutions for transfers between different celestial bodies using different crafts. For example, an Earth-Jupiter or Mars-Venus transfer, or even a multi-planet fly-by trajectory, could be tested by adding new bodies into the simulation framework. Since the program uses an approach similar to patched conics in that gravitational interactions are only simulated when in the body's SOI, this would not markedly increase the runtime of the simulation. One possible complication arising from this setup would be the increased number of inputs to the neural network. Rather than simply the positions of the craft, Earth, and Mars, the components of each body's position vector would have to be input into the neural network, increasing the complexity of the network and, therefore, the training time as well.

For craft modifications, the program includes a framework for changing the craft's dry and wet mass and the specific impulse of the engines, which all contribute to the delta-v available to the system. Reducing the delta-v available to the system by using a craft with less efficient engines and a heavier dry mass would force the neural networks to converge to a more delta-v optimized trajectory, though it could also result in a lack of convergence altogether.

5.3 Gravitational Assist Trajectories

Another interesting avenue for further exploration is the introduction of more massive bodies into the simulation. This would allow for the expansion of the possible starting

and ending points of transfer, as discussed above, in addition to opening up gravitational assist trajectories. In practice, gravity assist trajectories have been used on missions such as Voyager 1 and 2 to alter the craft's trajectory without the need for additional delta-v. If the craft were given the option to use gravity assists off of other bodies, such as the Moon for an Earth-Mars transfer, it would open up many novel pathways for exploration.

6 Summary

Overall, the goal of this project was to develop a program to test the claim that neural networks could be used as effective tools for optimizing orbital transfer maneuvers given different desired transfer characteristics. This program was created in Python and used NeuroEvolution of Augmenting Topologies and a custom fitness function to evolve the neural networks over time given the weights of several different training parameters. The simulation environment took the form of an N-body simulation that made use of an RK4 integrator also written within Python. A testing matrix was developed and executed for each parameter weight combination of interest and divided into the three categories of time-optimized (Human Variant), delta-v-optimized (Cargo Variant), and a combination of both (General Variant). Each simulation was run at the same orbital radius and starting date as the control Hohmann transfer and trained with 72 individuals for 100 generations.

The program was ultimately found to consistently develop solutions that outperformed the control Hohmann transfer for time-optimized transfers given high time to RSSOI input weights. Given an equal weighing of delta-v and time scores, the neural networks consistently developed solutions with similar performance characteristics to the Hohmann transfer but typically selected a single parameter to optimize for while leaving the other without optimization. For delta-v optimized transfers, the networks could not find solutions that were more efficient or even as efficient as the Hohmann transfer.

In future iterations of this project, several improvements could be made in expanding the simulation space for the network to be trained. Namely, the craft properties, number of celestial bodies, and desired starting and ending locations could be modified. These modifications would also expand the number of possible gravitational assist trajectories that the neural network could develop. Training the networks over a variety of dates rather than a single Hohmann transfer window date would also allow for neural networks to be developed that were less specialized to a single date and effectively "understood" the problem and data given rather than blindly working towards a solution.

7 References

[1] F. Cacciatore and C. Toglia, “Optimization of orbital trajectories using genetic algorithms,” *Journal of Aerospace Engineering, Sciences and Applications*, vol. 1, no. 1, pp. 58–69, 2008, doi: 10.7446/jaesa.0101.06.

[2] D. P. S. dos Santos and J. K. da Silva Formiga, “Application of a genetic algorithm in orbital maneuvers,” *Computational and Applied Mathematics*, vol. 34, no. 2, pp. 437–450, 2015, doi: 10.1007/s40314-014-0151-x.

[3] O. Abdelkhalik and D. Mortari, “N-Impulse Orbit Transfer Using Genetic Algorithms,” *Journal of Spacecraft and Rockets*, vol. 44, no. 2, pp. 456–460, Mar. 2007, doi: 10.2514/1.24701.

[4] Y. H. Kim and D. B. Spencer, “Optimal Spacecraft Rendezvous Using Genetic Algorithms,” *Journal of Spacecraft and Rockets*, vol. 39, no. 6, pp. 859–865, Nov. 2002, doi: 10.2514/2.3908.

[5] T. Paschalis, “What is RK4? - Explaination with Figure,” Jan. 22, 2018.

[6] Leafnode, “File:Hohmann transfer orbit.svg,” Mar. 04, 2007.