AIAA 2012-4592

GPU Accelerated Genetic Algorithm for Multiple Gravity-Assist and Impulsive ΔV Maneuvers

Sam Wagner, * Brian Kaplinger, † and Bong Wie ‡
Asteroid Deflection Research Center, Iowa State University, Ames, IA, USA

A combination of multiple gravity-assist and impulsive ΔV maneuvers are often utilized for interplanetary missions to outer planets, such as NASA's Galileo and Cassini missions. Such complex interplanetary missions often require the optimization of more than 20 variables, making brute force and traditional NLP methods difficult, if not impossible. In this paper a genetic algorithm, which utilizes the computational power of modern GPUs, is developed to solve the problem of designing advanced mission. By using modern GPUs the computational burden for advanced mission designs can be drastically reduced over traditional CPU optimization programs, allowing the mission designer to explore multiple mission scenarios or other mission options.

I. Introduction

Nearly every outer-planetary and interstellar missions require gravity-assist and/or deep-space maneuvers to keep mission costs down. A few notable example missions include Voyager 1, Voyager 2, Galileo, and Cassini. The Voyager missions used gravity-assist to insert the spacecraft on solar system escape trajectories, while Galileo and Cassini exploited inner-planetary gravity-assists to reach Jupiter and Saturn without the need of heavy lift launch vehicles.[1,2]

These types of missions can easily be formulated as constrained optimization problems, making them ideal for evolutionary algorithms. Due to their high computational cost, evolutionary algorithms have not traditionally been used for space mission designs. Recently, evolutionary algorithms have been employed by mission designers. With modern desktop microprocessor and, more recently, graphics processing units (GPUs) it is now possible to use evolutionary algorithms for mission design problems.

The traditional method for solving mission design problems is to have the designer prune the decision space, in order to obtain acceptable solutions. This is not possible for anyone, but the most experienced mission designers. Using genetic algorithms a more autonomous mission design program can be developed, removing the "guess work" from the mission designer. While there is no guarantee the global optimal solution will be found, evolutionary algorithms have been shown to work extremely well for highly non-linear and discontinuous problems. This paper will show that that when used properly, genetic algorithms can be used to find optimal or near optimal solutions in a computationally efficient manner.

The purpose of this paper is therefore to develop and implement a genetic algorithm that is capable of complex mission designs similar to NASA's Cassini mission. The results will be compared

*Graduate Research Assistant, Department of Aerospace Engineering, 2271 Howe Hall, AIAA Student Member.
†Graduate Research Assistant, Department of Aerospace Engineering, 2271 Howe Hall, AIAA Student Member.
‡Vance Coffman Endowed Chair Professor, Dept. of Aerospace Engineering, 2271 Howe Hall, AIAA Associate Fellow.

with past missions, notably Voyager 2, Galileo, and Cassini, in order to validate both the problem formulation methodology and the genetic algorithm results. The ultimate goal for developing such advanced mission design software tools is to enable the mission designer to efficiently and quickly conduct complicated mission designs. Example missions include, but are not limited to, multiple asteroid missions or low cost (low ΔV) missions to asteroid employing an arbitrary number of gravity assists and/or deep space maneuvers.

Two classes of problems will be tested. The first are traditional multiple gravity-assist(MGA) missions, such as the Voyager or Galileo missions. The second mission type, such as Cassini, uses multiple gravity-assists with the option for a deep space maneuver during each leg of the mission. The second mission type is often referred to as the MGA-DSM model in the astrodynamics community.[2,3]

II. Problem Formulation

Depending on mission complexity, there are different ways to formulate multiple gravity-assist problems. Simpler missions, such as NASA's Voyager, Pioneer, and Galileo missions, which don't require large deep space maneuvers, sometimes referred to as mid-course correction maneuvers, can be formulated using the MGA model. This model requires fewer variables than the MGA-DSM model, meaning they have a much lower computational cost. On many occasions, small deep space maneuvers, when compared to the departure and arrival maneuvers, can significantly lower to total overall mission cost (in terms of total ΔV or arrival mass). The formulation for both types of missions are included in the following subsections. Most of the solution algorithms needed for both problem types will be included. However, solutions to Lambert's problem, solutions to Kepler's equation, date conversions, etc will not be discussed in this paper.

The purpose of both methods is to determine a final cost function that the genetic algorithm can optimize. That is, either minimize or maximize, depending on how the individual problem is formulated. Cost functions typically consists of all of the required mission ΔV's, the Earth departure V_∞, the final arrival V_∞, and any other mission constraints. Any permutation of these elements can be used for each problem. For instance, when re-designing the Cassini mission the departure V_∞ is limited to approximately 4.25 km/s but is not directly part of the cost function. This is often done because the launch vehicle can be leveraged for the Earth departure burn (typically, limited by C_3). It is worth noting that for the genetic algorithm developed in this paper, all mission constraints are applied directly in the cost function.

A. Multiple Gravity-Assist Model

For the multiple gravity-assist model, two Lambert solutions are essentially "patched" together using the standard patched conic method. This results in a powered hyperbolic orbit for each gravity assist in which a ΔV is allowed only at the perigee passage for each gravity assist. The ΔV at each gravity assist is part of the final cost function, but is usually driven to a near zero value.

For each gravity assist, the incoming and outgoing velocity vectors (in the heliocentric frame) are given from Lambert's problem solutions. The velocities relative to the planets are then found as:

$$\vec{v}_{\infty-in} = \vec{V}_{s/c-in} - \vec{V}_{\oplus} \tag{1}$$

$$\vec{v}_{\infty-out} = \vec{V}_{s/c-out} - \vec{V}_{\oplus} \tag{2}$$

From this point the goal is to determine a method to find the perigee radius that is required to

patch the two solutions together. The first step is to determine the semi-major axis of the incoming and outgoing hyperbolic trajectories.

$$a_{in} = -\frac{\mu_{\oplus}}{v_{\infty-in}^2} \tag{3}$$

$$a_{out} = -\frac{\mu_{\oplus}}{v_{\infty-out}^2} \tag{4}$$

where $\mu_{\oplus}$ is the target planet's gravitational parameter.
The required turning angle is:

$$\delta = \cos^{-1}\left(\frac{\vec{v}_{\infty-in} \cdot \vec{v}_{\infty-out}}{v_{\infty-in} \cdot v_{\infty-out}}\right) \tag{5}$$

The flyby perigee radius is equal for both legs of the hyperbolic orbit. That is, we have

$$r_p = a_{in}(1 - e_{in}) = a_{out}(1 - e_{out}) \tag{6}$$

Where e_{in} and e_{out} are the incoming and outgoing orbit eccentricities. It should be noted that the orbits will be hyperbolic, so both eccentricities will be greater than 1. The turning angle δ can also be represented as the sum of the transfer angles for the incoming and outgoing orbits.

$$\delta = \sin^{-1}\left(\frac{1}{e_{in}}\right) + \sin^{-1}\left(\frac{1}{e_{out}}\right) \tag{7}$$

Eqnation 6 can be then rewritten for e_{in}, as

$$e_{in} = \frac{a_{out}}{a_{in}}\left(e_{out} - 1\right) + 1 \tag{8}$$

Substituting Eq. 8 into Eq. 7 gives

$$\delta = \sin^{-1}\left(\frac{1}{\frac{a_{out}}{a_{in}}\left(e_{out} - 1\right) + 1}\right) + \sin^{-1}\left(\frac{1}{e_{out}}\right) \tag{9}$$

which can be rewritten as

$$f = \left(\frac{a_{out}}{a_{in}}(e_{out} - 1)\right)\sin\left(\delta - \sin^{-1}\left(\frac{1}{e_{out}}\right)\right) - 1 = 0 \tag{10}$$

The above equation, which is now only a function of e_{out}, can be iterated upon to solve for e_{out}. A simple Newton iteration scheme works well. For this the derivative of f with respect to e_{out} must first be found.

$$\frac{df}{de_{out}} = \left(\frac{a_{out}}{a_{in}}e_{out} - \frac{a_{out}}{a_{in}} + 1\right)\frac{\cos\left(\delta - sin^{-1}\frac{1}{e_{out}}\right)}{e_{out}^2\sqrt{1 - \frac{1}{e_{out}^2}}} + \frac{a_{out}}{a_{in}}\sin\left(\delta - \sin^{-1}\frac{1}{e_{out}}\right) \tag{11}$$

To start the Newton iteration an initial value for e_{out} of 1.5 works well. In a typical Newton iteration scheme a do while loop is used until e_{out} stops changing within a certain tolerance. The iteration number inside the loop should also be monitored, so the loop can be exited if convergence doesn't occur. Each new e_{out} is calculated as:

$$e_{new} = e_{old} - \frac{f}{\frac{df}{de_{out}}} \tag{12}$$

When a converged e_{out} is found the perigee radius is calculated from Eq. 6. Finally the ΔV that must be applied to the perigee is.

$$\Delta V_{GA} = \left| \sqrt{v_{\infty-in}^2 + \frac{2\mu_\oplus}{r_p}} - \sqrt{v_{\infty-out}^2 + \frac{2\mu_\oplus}{r_p}} \right| \tag{13}$$

The flyby perigee radius and ΔV are found directly from the spacecraft's incoming and outgoing velocities. These are found from solutions to Lambert's problem, which are a function of planetary positions and time of flight. The planetary positions are also a function of time, meaning that the decision variables for the MGA model are the time of Earth departure, time of each gravity assist, and final arrival time. Thus the final cost function C for the MGA optimization problem is represented as.

$$C = f(\mathbf{X}) + g(\mathbf{X}) \tag{14}$$

$$\mathbf{X} = [T_0, T_1 \ldots T_f]^T \tag{15}$$

In this case T_0 is the launch date, T_i is the time of flight for each leg of the mission, and T_f is time of flight for the final mission leg. The penalty function, $g(\mathbf{X})$ will be covered later. As shown above each element of the cost function, ΔV's, etc, are only a function of time. This MGA cost function can then be optimized by the genetic algorithm to find optimal or near optimal solutions.

B. Multiple Gravity-Assist Deep Space Maneuver Model

The MGA model, while simple and easy to implement, is not suitable for all missions. Often times allowing deep space maneuvers, where the ΔV is applied sometime during the interplanetary coast arc, rather than at the flyby perigee allows for a more optimal solutions to be found. This simply implies that the optimal location for mission burns may not be at the flyby perigees. This is why the MGA-DSM model was originally developed. This section will outline the basics necessary to implement this model.[2,3,9]

In the MGA-DSM model the mission is broken down into three phases, Earth departure, gravity assist, and the final terminal phase. For the launch phase an impulsive ΔV can be applied in any direction from Earth. This can be defined by three variables, the v_∞ magnitude (or possibly C_3), and the departure right ascension and declination angles, α and β respectively. Three additional variables are also needed to completely describe the model, namely the launch date, time of flight from launch to the first flyby, and the burn index (ϵ). The burn index is used to define the point along the trajectory in which a ΔV will be applied to target the next planet in the sequence. The burn index can vary anywhere between 0 and 1, although both 0 and 1 would represent a simple Lambert solution without a correction maneuver. Specifying the departure declination directly has the added benefit of ensuring that any trajectories found will be flyable by a given launch vehicle. Launch vehicles often have lowered C_3 performance, and often isn't specified in the appropriate payload planners guides, when the departure declination is outside of the launch vehicle's intended range. The spacecrafts initial state vector at the Earth departure is:

$$\vec{r}_{s/c} = \vec{r}_\oplus(T_0) \tag{16}$$

$$\mathbf{V}_{s/c} = \mathbf{V}_\oplus(T_{launch}) + v_\infty \begin{bmatrix} \cos\alpha\cos\beta \\ \sin\alpha\cos\beta \\ \sin\beta \end{bmatrix} \tag{17}$$

From this point on, the next step is the essentially same for both the departure phase and gravity assist phases. The spacecraft's state vector is propagated forward by a time $\epsilon_1 T_1$, at which point a deep space maneuver is applied to target the next planet. This targeting is performed by applying a solution to Lamberts problem, with the time of flight given by $(1-\epsilon_1)T_1$. The magnitude of the deep space maneuver is just the magnitude of the difference between the velocity vector after the orbit it propagated and the initial velocity vector from the Lambert's solution. The planetary arrival velocity, also from the Lambert solver, is then used for the next phase of the mission.

For each gravity assist phase of the trajectory, an additional four variables are necessary, the flyby radius ratio (r_{pi}), b-plane insertion angle (γ_i), time of flight (T_i), and burn index (ϵ_i). In this case i is the gravity assist number of the trajectory. The perigee radius ratio directly defines the perigee radius. In this formulation the perigee radius is a variable, rather than being defined by the problem geometry. In order for the incoming and outgoing hyperbolas to be defined the incoming and outgoing v_∞ magnitudes are set equal.

$$R_{pi} = r_{pi} R_\oplus \tag{18}$$

$$|\vec{v}_{\infty-in}| = |\vec{v}_{\infty-out}| \tag{19}$$

Similar to the the MGA model, the incoming velocity vector is given by:

$$\vec{v}_{\infty-in} = \vec{V}_{s/c-in} - \vec{V}_\oplus \tag{20}$$

From the patched conic model[10] the eccentricity must be found. In the MGA-DSM model the arrival and departure v_∞ magnitudes are defined to be the same, thus the semi-major axis and eccentricity are the same for both arrival and departure trajectories.

$$a = -\frac{\mu_\oplus}{v_\infty^2} \tag{21}$$

$$e = 1 - \frac{R_{pi}}{a} \tag{22}$$

The flyby turning angle is given by

$$\delta = 2 \arcsin \frac{1}{e} \tag{23}$$

The outgoing v_∞ is the obtained from the following four equations.

$$\vec{v}_{\infty-out} = v_\infty \left[\cos\delta \hat{i} + \cos\gamma_i \sin\delta \hat{j} + \sin\gamma_i \sin\delta \hat{k} \right] \tag{24}$$

$$\hat{i} = \frac{\vec{v}_{\infty-in}}{|\vec{v}_{\infty-in}|} \tag{25}$$

$$\hat{j} = \frac{\hat{i} \times \vec{V}_{\infty-in}}{\left| \hat{i} \times \vec{V}_{\infty-in} \right|} \tag{26}$$

$$\hat{k} = \hat{k} \times \hat{j} \tag{27}$$

The outgoing velocity is found the same as the MGA model, as follows.

$$\vec{V}_{s/c-out} = \vec{v}_{\infty-out} + \vec{V}_\oplus \tag{28}$$

Similar to the Earth departure phase, the spacecraft is propagated forward by $\epsilon_i T_i$ time. Then the next planet is targeted using a Lambert solver. The deep space maneuver is found just like in the departure phase of the trajectory.

The terminal phase of the mission can be formulated in multiple ways. The deep space maneuver which targets the final planet is part of the prior flyby cost function, so the only addition to the cost function is typically either the arrival v_∞ or a ΔV for a specific orbit insertion. Often the orbit about the final planet is defined by an insertion perigee radius and insertion eccentricity or orbit period.

The cost function is represented in the same way as the MGA model. However, the number of total state variables is increased significantly when compared to the MGA model. As before, $f(\mathbf{X})$ is the actual cost function (real mission ΔV's, etc) and $g(\mathbf{X})$ is a penalty function used to enforce and problem constraints. N is defined to be the number of gravity assist required by the mission.

$$C = f(\mathbf{X}) + g(\mathbf{X}) \tag{29}$$

$$\mathbf{X} = [T_0, v_{\infty-0}, \alpha_0, \beta_0, \epsilon_0, T_l, T_1 \dots T_{n+1}, \epsilon_1 \dots \epsilon_n, \gamma_1 \dots \gamma_n, r_{p1} \dots r_{pn}]^T \tag{30}$$

In Eq. 30 n is the number of gravity-assists to be performed. An example final form of a typical cost function, without any constraint penalties, is shown below.

$$C = v_{\infty-0} + \Delta V_0 + \Delta V_1 \dots \Delta V_n + v_{\infty-f} \tag{31}$$

Where ΔV_i is the magnitude of the deep space maneuver corresponding an accompanying planetary flyby.

C. Problem Constraints

In optimization problems, constraints are often used to help shape the final solution. When implementing genetic algorithms, constraints are used, rather than hard limits, in order to keep to solution space open. If the solution space isn't sufficiently large the genetic algorithm will be unable to start, due to the random initialization process.

Common constraints for mission design problems include, perigee radius during a gravity assist, mission time/leg length constraints, and guarding against low velocity flybys. During a low-velocity flyby the spacecraft is captured by the planet, rather than gaining velocity in the heliocentric frame. Any other constraint the user wishes impose to help shape the solution can be added, as long as the constraint values are approximately the same order of magnitude at the actual cost function values. It should be noted that when using the MGA-DSM model in conjunction with a genetic algorithm all of the variables can be explicitly constrained. However, low-velocity flybys can still occur.

Often time, the MGA model results in a perigee radius that passes through the planet or too close to the planets atmosphere. In this situation a constraint-handling method is necessary to move the solution toward more feasible solutions. The method below is from Englander et al.[3]

$$g_i(\mathbf{X}) = -2 \log \frac{R_{pi}}{k R_\oplus} \tag{32}$$

Here k is a multiplier used to define how close the spacecraft is allowed to flyby a target planet. For all of the examples in this paper a value of 1.1 was used. The only exception was when reproducing the Galileo mission, which had an Earth close approach altitude of 300 km for the second gravity assist (k value of approximately 1.047).

The second constraint penalty method penalizes low velocity flybys. The method has also been adapted from Englander et al.[3] Low velocity flybys are very rare, but solutions should still be protected from converging on them. The orbital energy, about the flyby planet, can be calculated as.

$$E = \frac{|\vec{v}_{\infty-in}|^2}{2} - \frac{\mu_\oplus}{R_{soi}} \tag{33}$$

For the flyby orbit to be hyperbolic about the planet E must be greater than zero. However, the sphere of influence model is an approximation, so an additional 10% margin on the incoming velocity needs to be added. A simple penalty that is scaled inversely to the flyby arrival velocity scales well with. For relatively large v_∞ the penalty is zero or very small compared to the overall cost function value. Alternatively for very low v_∞ values the penalty is large enough to severely influence the final shape of the solution. The flyby orbital energy, adjusted for the 10% margin, and final constraint are calculated using the following two equations:

$$E = \frac{|0.9\vec{v}_{\infty-in}|^2}{2} - \frac{\mu_\oplus}{R_{soi}} \tag{34}$$

$$g_i(\mathbf{X}) = \begin{cases} 0 & E \geq 0 \\ \frac{1}{|\vec{v}_{\infty-in}|} & E < 0 \end{cases} \tag{35}$$

Additionally, any other penalty constraint functions can be added in to shape the solution as the user desires. The final constraint function is the sum of the individual constraint functions given by.

$$g(\mathbf{X}) = \sum g_i(\mathbf{X}) \tag{36}$$

With the cost function for each method finalized then next step is to develop the genetic algorithm capable of optimizing these types of advance mission design problems.

III. Overview of Genetic Algorithms

The heart of a genetic algorithm is the simulation of natural selection, reproduction, and mutations found in nature. Genetic operators are used to 'evolve' an initial population, through genetic operators, in order to determine a best fitness design.[11] The purpose of this section is to develop and implement a genetic algorithm for the MGA, MGA-DSM, and other complex mission design problems.

A genetic algorithm(GA) is a stochastic optimization method based on the principles of evolution. Genetic algorithms perform a probabilistic search by evolving a randomly chosen initial population. The population is just a series of sets of variables that are evaluated by a cost function, in this case the cost functions previously covered. The advantage of using evolutionary methods over traditional optimization methods is that no initial solution is necessary. This helps ensure, but does not guarantee, that solutions are not confined to locally optimal solutions. Genetic algorithms also perform well in very complex nonlinear design spaces. Despite all the advantages, evolutionary algorithms do have their downside. They almost always require a greater number of cost function evaluations, increasing the computational requirements. Additionally evolutionary algorithms do not make use of gradients, so there is no proof of convergence. It should also be noted that genetic algorithms were developed for constrained minimization problems and may not perform well for unconstrained minimization.

The basic genetic operators are, selection, reproduction/crossover, mutation, and elitism. The genetic algorithm developed in this section can utilize a number of different selection, reproduction, crossover, and mutation methods. Each of the methods will be discussed in detail in later sections.

In a simple genetic algorithm (SGA), each variable is represented by a binary string where individual bits are represented by 1's and 0's. This binary string is referred as a gene. The separate genes for each variable are then concatenated to form a complete chromosome. One chromosome correspond to one member of the entire population, which can number in the hundreds of thousands for some problems. This binary representation of variables means that each variable is discretized to a certain resolution. In our case the desired resolution for each variable is one of the inputs to the genetic algorithm, along with the individual variable upper and lower bounds, size of the population, and number of generations the population is to be run out. For an individual variable the resolution is defined as

$$r = \frac{x^U - x^L}{2^b - 1} \tag{37}$$

where x^U and x^L are the user specified variable upper and lower bounds, while b is the minimum number of bits required to obtain the desired variable accuracy. Eqn. 37 can then be solver for b for each individual variable. With the variable b known for each variable the size of each chromosome can be determined.

Then next step in the process is to randomly, via a "digital coin flip", assign a value (1 or 0) to each position in the chromosome. The coin flip is performed using a uniform random number generator, that generates random numbers between 0 and 1. If the random number is greater than 0.5, the a value of 1 is assigned, otherwise a value of 0 is assigned. This process is run out to generate an entire initial population, which is randomly distributed throughout the design space.

From this point each member of the population is assigned a fitness value, via a user supplied cost function. The genetic operators are then used to generate a new population from the initial parent population. This is the crux of how genetic algorithms are able to evolve to more fit populations. This process continues until the genetic algorithm is told to exit and output the final population. Common stopping conditions include stopping after a certain number of iterations, monitoring when the best fit solution stop changing, or monitoring when the average cost function value approaches the population minimum. For this study the stopping condition is always running the genetic algorithm out by a user specified number of generations. The sequence of operations of the core operations of the genetic algorithm is illustrated in Fig. 1. One secondary operator, elitism, was used in this genetic algorithm as well. The elitism operator ensure that the best fit solution(s) are not lost from generation to generations. A simple elitism

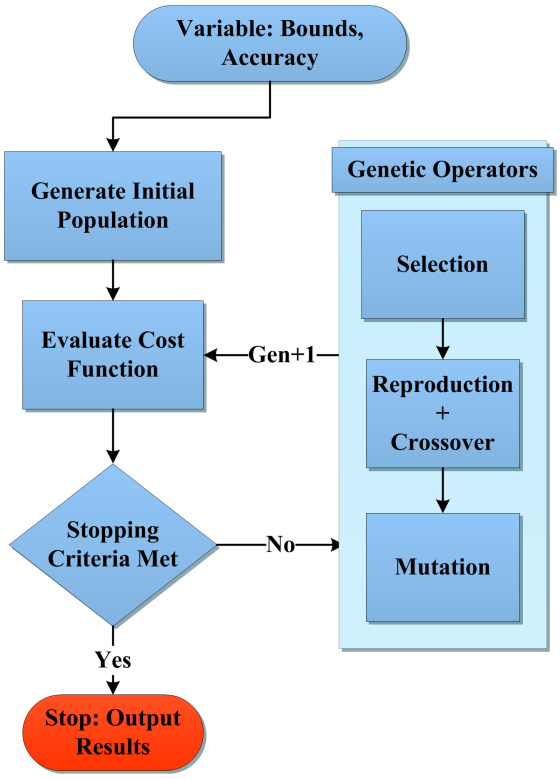

Figure 1. Simple genetic algorithm flow chart.

operator is used, in which the top two solutions from the parent population are directly inserted into the next generation.

A. Selection Types

In genetic algorithms, selection operator, also the first genetic operator, is used to ensure that the best fit solutions are chosen to pass on their genes through the reproduction and crossover operators. The selection operator is essentially the operator that represents the survival of the fittest principle. A total of two selection types have been implemented in this genetic algorithm one based solely on a roulette selection method, and another that combines roulette and tournament selection. These two operators are by no means the only possible selection methods, but that are some of the simplest to use and prove to work well for the mission design problems.

Roulette selection is a fitness proportionate selection method in which better fit individuals are more likely to be chosen for reproduction and crossover. There are four fitness types that are used in the roulette selection method described in this section. The ultimate goal is to get a normalized fitness that ranges from 0 to 1. More fit members of the population will have a higher normalized fitness value, thus they will be more likely to be chosen for further genetic operations.

The first fitness type is the raw fitness, $r(i)$, which is the fitness values provided directly from the cost function for each population member. In this case i represents the chromosome/population number. The raw fitness is then standardized, depending on whether the problem is being minimized and maximized. If the problem is a minimization problem, the standardized fitness is the same as the raw fitness as

$$s(i) = r(i) \tag{38}$$

However, if the problem is a maximization problem, the standardized fitness becomes

$$s(i) = r_{max} - r(i) \tag{39}$$

Now the fitness is adjusted so that individual fitnesses lie between 0 and 1. The adjusted fitness is larger for better individuals. The advantage of this optional step is that small differences in the most fit individuals, those that approach an adjusted of 0, are exaggerated. This has a larger benefit when the population improves by emphasizing small differences in individuals. This effect is most exaggerated in problems where the best solution has a cost function near 0,[12,13] as follows

$$a(i) = \frac{1}{1 + s(i)} \tag{40}$$

Now that the adjusted fitness values lie between 0 and 1 the next step is to normalize the whole population, so the sum of each normalized fitness value is 1. Ultimately, this is necessary because individuals are chosen through the use of random numbers, that also in 0 and 1 range, as

$$n(i) = \frac{a(i)}{\sum_{j}^{n} a(j)} \tag{41}$$

The actual selection of individuals, based on normalized fitness, is done in the following manner. Firstly the normalized fitnesses are sorted from largest to smallest. Next a random number is generated. This random number will decide which individual is then selected. The actual selection of an individual is done by a summation of the normalized fitness values. The individual that causes the summation value to be greater than the random number is chosen to move along through the rest of the genetic operators. This process is repeated and both of the individuals then go through

the reproduction or crossover operators. Thus, more fit individuals are more likely to be passed on to the next generation.

Tournament is a greedy over selection method that also utilized the roulette selection method. Two individual are chosen using roulette selection and then compete to see which individual is allowed to pass on its genetic material. This is done through the direct comparison of their normalized fitness function. The individual with the best fitness is chosen as the one that gets to pass through to the reproduction and crossover operations. The process is repeated twice, resulting in two individuals who's genetic material can be passed on.

This selection method has a distinct advantage in that it drives down the raw fitness and average of the fitness function in fewer generations than pure roulette selection. With all greedy over selection methods there is a risk that genetic diversity, which could help out the population in later generations, will be lost. Therefore tournament selection should be used with care by the user. In the orbital problems discussed in this paper, tournament selection does appear to work well.

B. Reproduction and Crossover Operator Types

While the selection operator determines which population members get the privilege of reproducing and passing on their genetic material to future generations, the crossover and reproduction operators decided what to do with that genetic material. In the algorithm utilized in this paper the user must supply the probability in which an individual will under go either reproduction or crossover. These probabilities must total up to 1.0 (100%), with values typically being close to 0.9 and 0.1 for crossover and reproduction respectively. It should also noted that both operations require two parent and produce two offspring.

The simplest of these two operators is reproduction. In traditional genetic algorithm terms this means that the operation is a fitness-proportionate process in which individuals are allowed to directly pass to the next generation with a probability proportionate to their fitness values. This essentially means that when reproduction occurs the two parents are passed directly from the parent generation into the next generation. As with every other critical procedure in the genetic algorithm, whether the parents undergoes reproduction or crossover is chosen through the use of random number. In the algorithm a random number is generated. If the random number is greater than the user given crossover probability the parents undergoes reproduction. If not, the a crossover operation is applied. This, of course, is the reason that the two probabilities must total up to 100%.

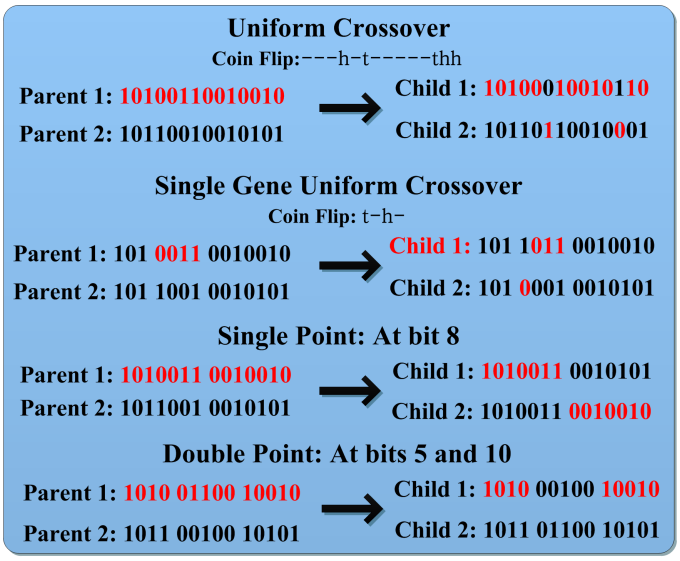

Figure 2. Simple illustration of the four implemented crossover types.

The crossover process is a process in which biological reproduction is used to allow new individ-

uals, with new and unique genetics, to be created. Unlike the reproduction process, crossovers allow new points in the design space to be searched. Without reproduction (and also mutations) genetic algorithms would be no more useful and a completely random search. The crossover operation produces to offspring from two parents that contain genetic material from both parents. A total of 4 distinct crossover types have been implemented in for use with this genetic algorithm. The user supplies which crossover type should be used, making the final genetic algorithm suitable for many different types of optimization problems. For the mission design problems discussed in this paper the double point crossover proved to be the best crossover type. However for other types of problems this may not necessarily be the cause.

The purpose of each crossover type is to promote genetic diversity and expand, in a controlled way, the search of the design space. The first type considered is uniform crossover. It has been shown to be a very effective method for promoting genetic diversity, and in turn the discovery of new useful chromosomes.[11,14] In uniform crossover each bit in the chromosome is a crossover point. The two offspring are generated by the same virtual coin flip that was used to initially generate the population for the first generation. If two bit are the same for both parents no coin flip is necessary. However, if two bits are not identical in the parents the coin flip is used to decide which offspring gets the genetic material from the separate parent individuals. This process is graphically shown in the first entry in Fig. 2. In this case if the coin flip is heads, i.e. the random number is greater than or equal to 0.5, the bit from the second parent is inserted into the chromosome for the first child and the bit from the first parent is inserted into the chromosome of the second child. The exact opposite is true if the coin flip is tails, i.e. the random number is less than 0.5.

A second crossover operator has also been developed that is similar to uniform crossover but only operates on an individual gene. This process only changes the value of one gene (i.e. variable). In some situations changes to an individual variable can greatly improve (or alternatively worsen) the individuals fitness. For this process a gene is randomly selected. The same gene then undergoes uniform crossover. An illustration of this method can be seen in Fig. 2. In this example the second gene (bits 4-7) undergoes uniform crossover through the same virtual coin flip method described above.

In single point crossover one bit is randomly chosen to be the crossover point. All bits after the random crossover point are swapped between the two points. Double point crossover is the same thing with two randomly chosen crossover points. While these two methods are the simplest of the tested crossover methods they have proven extremely useful in practical applications.

C. Mutation Types

The last genetic operator, which the newly generated population must pass through, is the mutation operator. The probability that a mutation will occur and the desired type of mutation are the last two user inputs. The probability that a mutation should occur should be small, typically less than 0.05 (5%). When the mutation probability is set too high the genetic algorithm will start to resemble a simple random search. If the user doesn't wish to make use of the mutation operator, a probability of 0 can be entered as well.

Allowing the genetic algorithm to utilize a mutation operator has several advantages. Mutations are used to help maintain genetic diversity in a population as it ages. Often times after many generations the population can lose genetic diversity and stagnate at a local minimum. Mutations help to prevent this from happening. By introducing (or in some cases reintroducing) changes in a chromosome or individual gene it is possible for better more fit individuals to appear.

As with the crossover operator four separate types of mutations have been implemented. Unlike reproduction and crossover mutations operate on a single individual in the population. Different mutation types may work well for individual problems. The four types introduced here are rep-

resentative of some of the most common types of mutations used in genetic programming. Each mutation type is illustrated in Fig. 3.

The simplest type of mutation is the flip bit mutation. In this type of mutation the program randomly chooses a single bit to be flipped. When a bit is flipped that value is either changed from 1 to 0 or from 0 to 1. The next three mutation types are only allowed to operate on a single randomly chosen gene.

The boundary mutation type is as simple as it sounds. An individual gene is set either to corresponding variables user supplied minimum or maximum. The digital coin flip is used to decided whether the minimum or maximum values will be used. To set the gene to the maximum every bit is set to 1. Alternative is the gene is set to the minimum every bit is changed to 0.

Uniform mutation is similar to uniform crossover and the method used to initialize the population. A digital coin flip is used to completely redefine an individual gene. The last mutation type studied, inversion, is a simple mutation operator that has proven to work extremely well for mission design problems. As the name implies, when inversion is applied the bits that make up the gene are simply inverted.

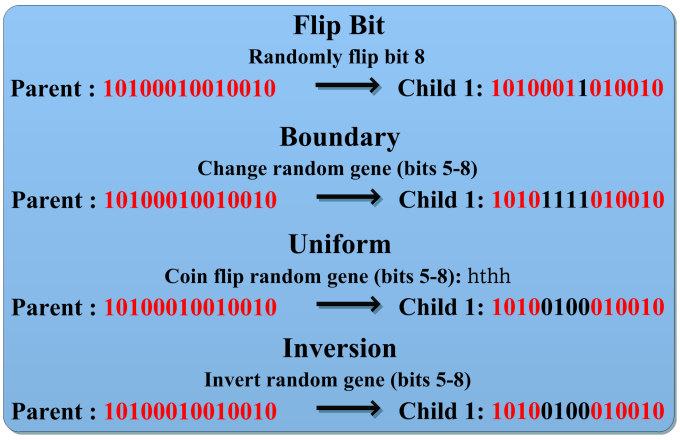

Figure 3. Simple illustration of the four mutation types implemented.

IV. Comparison of Genetic Operator Performances

When using genetic algorithms, it is beneficial to perform a study to determine which combination or selection, crossover, and mutation operators work best with the type of problem being solved. A much more detailed study than that illustrated in Table 1 was performed when designing and implementing the genetic algorithm. None the less, Table 1 is representative of the results. In general, the best crossover type for the MGA and MGA-DSM problems is double point. It is less clear which mutation type is the best. Both the uniform and inversion operators perform well. For the following example problems, the selection, crossover, and mutation types used are tournament, double point, and inversion respectively.

Table 1. Comparison of mutation and crossover types for a Cassini-type MGA-DSM mission or a population size of 20,000 run for 200 generations. The numbers represent the final cost function values in km/s.

Crossover Type	Mutation Type			
	Flip Bit	**Uniform**	**Boundary**	**Inversion**
Uniform	6.510	6.938	6.805	6.536
Uniform Gene	4.903	5.050	5.626	5.368
Single Point	5.182	4.148	4.636	4.687
Double Point	4.389	3.731	5.137	3.363

V. MGA and MGA-DSM Mission Design Examples

With the development of the genetic algorithm and problem formulation completed, a few example missions will be tested in ordered to benchmark the performance of the final program. This series of benchmarks helps to ensure the final program works correctly and is capable of finding optimal or near optimal solutions. To date, the final program has been able to reproduce (within a reasonable tolerance) every past mission that has been attempted (for obvious reasons low-thrust missions are not included in this list). Three past missions are shown in this section that are representative of the some of the most common types of missions that mission designers are often asked to develop. The 3 missions, Voyager 1, Galileo, and Cassini, progressively increase mission complexities.

A. Simple Voyager Type Missions

Voyager 1 was launched on September 5th 1977 and is still in operation today, over 34 years later. The spacecraft is now the most distant man-made object from Earth. It was developed to study the outer planets and interstellar medium. This was accomplished through gravity assists at Jupiter and Saturn. After the Saturn gravity assist, the probe continued on an interstellar course.

In this section, the mission will only be modeled up to the Saturn gravity assist. To ensure a trajectory was found that would ultimately result in approximately the same interstellar trajectory, the cost function was modified to penalize solutions that didn't arrive with a V_∞ value at Saturn that was approximately the same value as the actual Voyager 1 mission. For the departure leg of the trajectory, the Earth departure v_∞ was used. The final cost function is defined as

$$C = v_{\infty-0} + \Delta V_{GA-Jupiter} + G(\mathbf{X}) + |v_{\infty-saturn-in} - 15.286| + G(\mathbf{X}) \tag{42}$$

The genetic algorithm inputs are shown in Table 2(a). For this problem the MGA model was used, resulting in 3 design variables. As the table shows, the mission time ranges were are very open compared to actual mission lengths shown in Table 2(b). However, the launch date ranges had to be limited to less than a year prior to the actual launch. If this was not done the optimal solutions found by the algorithm have launch dates in the 1975-1976 range. For this mission a relativly small population of 15,000 was run out 50 generations to obtain this solution, although the algorithm did appear to converge after roughly 20 generations. With a relatively small population and low number of design variables and generations, the algorithm took approximately 2 minutes to run on the CPU. Given the low run times, this problem was not run on the GPU version of the program. None the less this example mission helps to validate the proposed approach.

As Table 2(b) shows, the genetic algorithm results are nearly identical to the actual mission flown. In the optimal solution, launch occurs approximately 2 days prior to what the launch actually

Table 2. Genetic algorithm inputs and outputs for Voyager 1 mission.

(a) Genetic Algorithm Inputs for Voyager 1 mission.

Minimums	
Date	1-Jan-1977 12:00:00
J_l (JD)	2443145
T_1 (days)	50
T_2 (days)	50
Maximums	
Date	1-Jan-1980 12:00:00
J_l (JD)	2444240
T_1 (days)	2000
T_2 (days)	2000
Variable Accuracy	
R_1	0.0001
R_2	0.0001
R_3	0.0001
Operators	
Selection Type	Tournament
Crossover Type	Double Point
Mutation Type	Inversion
Probabilities	
P_{rep}	0.1
P_{cross}	0.9
P_{mut}	0.01
Sizes	
N_{pop}	15000
N_{gen}	50

(b) Genetic Algorithm Results for Voyager 1 mission compared to actual mission values.

Item	**Actual**	**Results**
Earth Departure		
Date	5-Sep-77	3-Sep-77
J_l **(JD)**	2443391.5	2443390.1
T_1 **(days)**	546	546.631
$v_{\infty-l}$ **(km/s)**	10.315	10.311
Jupiter Gravity Assist		
Date	5-Mar-79	4-Mar-79
T_2 **(days)**	618	618.293
R_{per} **(km)**	337242.866	336517.521
δ **(deg)**	98.506	98.546
$v_{\infty-arr}$ **(km/s)**	10.955	10.967
$v_{\infty-dep}$ **(km/s)**	10.972	10.969
Δ **V (km/s)**	0.0061	0.0008
Saturn Arrival		
Date	12-Nov-80	11-Nov-80
J_f **(JD)**	2444555.5	2444555.026
$V_{\infty-arr}$ **(km/s)**	15.286	15.286
Total Cost (km/s)	**10.321**	**10.312**

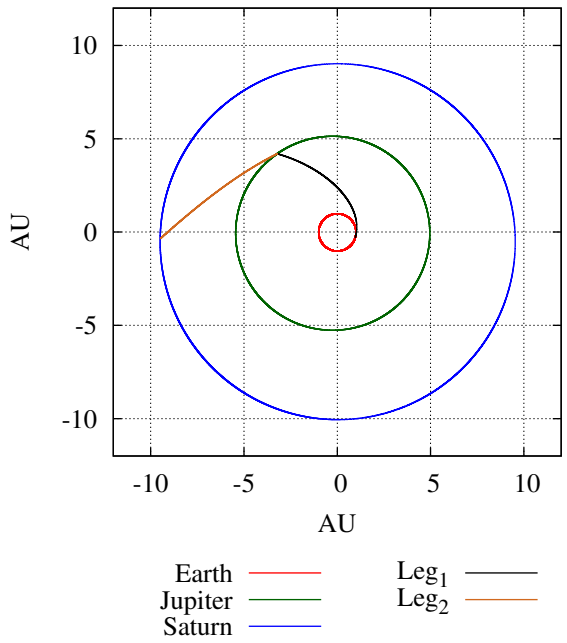

Figure 4. Trajectory plot for Voyager 1 mission.

was. Arrival is 1 day prior, with the approximate extra made up primarily on the Earth to Jupiter leg of the mission. The actual results represent output from the cost function using the actual launch date and mission lengths. The results are within a reasonable margin of error to conclude that the genetic algorithm output is highly accurate. The calculated trajectory is shown in Fig. 4

B. The Galileo Mission

The second mission used to benchmark the genetic algorithm was the Galileo mission. Galileo was launched on October 18th, 1989 on a mission to explore Jupiter. The spacecraft was launch from the Space Shuttle Atlantis using a Centaur upper stage. The combination of the spacecraft's high mass (over 2000 kg) and the Centaur upper stage placed tight constraint on the achievable Earth escape energy, or C_3, with is just $v_{\infty-0}^2$. With this launch vehicle limitation, Galileo was forced to make use of interplanetary gravity assists in order to achieve the energy necessary to reach Jupiter. Three gravity assists were necessary, the first at Venus and the second two at Earth. The mission was further complicated by the use of a 2-year resonant orbit between the two Earth gravity assists. The mission sequence flown by the Galileo spacecraft is often referred to as VEEGA (Venus-Earth-Earth Gravity Assist).

To reproduce the Galileo mission, the MGA model, with a total of 5 variables, was invoked. An algorithm had to be implemented that could determine solutions for resonant orbits (most Lambert solutions are not accurate when the two radii are co-linear). Thus an addition was made to the MGA model to check for resonant orbits. If this happened, the Lambert solutions were simply replaced. Therefore, a resonant orbit in the Earth-Earth leg of the journey was possible, but not strictly enforced. The Galileo problem certainly helps show the effectiveness of genetic algorithms, as the final solution did result in a very nearly resonant orbit.

The cost function for this mission was developed in a similar manner to the Cassini mission (both have tight constraint on the Earth departure energy). Any solutions that required a v_∞

Table 3. Genetic algorithm inputs and outputs for Galileo mission.

(a) Genetic Algorithm Inputs for Galileo mission.

Minimums	
Date	1-Jan-88
T_0 (JD)	2443145
$T_{1,2}$ (days)	50
$T_{3,4}$ (days)	250
Maximums	
Date	1-Jun-90
T_0 (JD)	2448044
$T_{1,2}$ (days)	750
$T_{3,4}$ (days)	1500
Variable Accuracy	
R_i	0.0001
Operators	
Selection Type	Tournament
Crossover Type	Double Point
Mutation Type	Inversion
Probabilities	
P_{rep}	0.1
P_{cross}	0.9
P_{mut}	0.01
Sizes	
N_{pop}	50,000
N_{gen}	500

(b) Genetic Algorithm Results for Galileo mission compared to actual mission values.

Item	**Actual**	**Results**
Earth Departure		
Date	18-Oct-89	18-Nov-89
T_1 **(days)**	114.545	97.490
$v_{\infty-0}$ **(km/s)**	3.936	3.935
Venus Gravity Assist		
Date	10-Feb-90	23-Feb-90
T_2 **(days)**	301.608	273.057
R_p **(km)**	22061.131	13239.258
δ **(deg)**	32.864	67.795
$v_{\infty-arr}$ **(km/s)**	6.181	4.412
$v_{\infty-dep}$ **(km/s)**	6.041	4.411
Δ **V (km/s)**	0.104	0.000
Earth-1 Gravity Assist		
Date	8-Dec-90	23-Nov-90
T_3 **(days)**	730.792	725.865
R_p **(km)**	10755.863	10658.497
δ **(deg)**	37.402	43.835
$v_{\infty-arr}$ **(km/s)**	8.861	7.924
$v_{\infty-dep}$ **(km/s)**	8.861	7.924
Δ **V (km/s)**	0.000	0.000
Earth-2 Gravity Assist		
Date	8-Dec-92	18-Nov-92
T_4 **(days)**	1185.369	1089.800736
R_p **(km)**	7498.785	10097.751
δ **(deg)**	47.648	31.828
$v_{\infty-arr}$ **(km/s)**	8.733	10.090
$v_{\infty-dep}$ **(km/s)**	8.984	10.358
Δ **V (km/s)**	0.164	0.202
Jupiter Arrival		
Date	8-Mar-96	Nov 13,1995
$V_{\infty-arr}$ **(km/s)**	5.798	5.696
Δ **V (km/s)**	0.758	0.738
Total Cost (km/s)	1.026	0.941

higher than 4.24 km/s was penalized. Additionally, the arrival cost function was the ΔV necessary to obtain an orbit with a perigee radius of 286,000 km an and apogee 10,776,000 km about Jupiter. All of the genetic program inputs are shown in Table 3(a) and the results are shown in Table 3(b). The final optimized trajectory is shown in Fig. 5.

The results of the genetic algorithm are extremely close to the actual mission shown with a couple of notable differences. The launch date found is nearly 1 month later than the actual mission flown. The first four legs of the mission are a few days shorter than the actual mission. The most notable difference is the Earth to Jupiter leg of the mission, which is nearly 100 days shorter. In total the mission is slightly over 146 days shorter than the actual flown mission. Further examination of the results reveals that after the second Earth gravity assist the trajectory has a speed relative to the Earth of 10.358 km/s, compared the the actual trajectory speed of 8.984 km/s. This additional velocity appears to be the reason for the mission length discrepancy. The mission found by the genetic algorithm also has a slightly lower final cost function value, equivalent to a cost savings of 85 m/s. It should also be noted that the numbers for the actual mission in Table 3(b), other than the dates, are obtained from our own cost function. Small differences in our"actual" and the real mission likely come from the time fit ephemeris model used for the planets, instead of more accurate ephemerides.

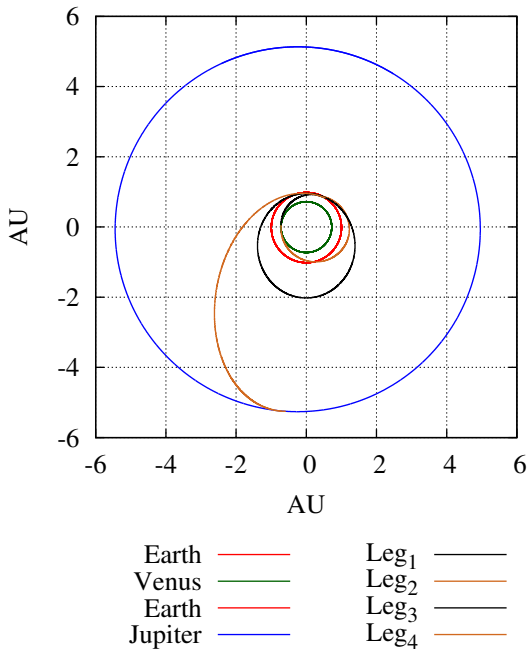

Figure 5. Trajectory plot for the Galileo mission.

C. Cassini Mission

The last benchmark mission examined is the Cassini trajectory. This is the only trajectory that makes use of the MGA-DSM model with 4 gravity assists. Cassini was launched on October 15th, 1997 and entered a Saturn orbit on July 1st, 2004. Like Galileo, the Cassini mission utilized gravity assists to reach its intended target. The mission is further complicated by the use of two large deep space maneuvers. Without these maneuver a feasible mission trajectory was not possible. Because the flown trajectory incorporates two large deep space maneuvers, the MGA-DSM model is required

to include them. Reproducing the Cassini trajectory, or one relatively close to it, using the proposed approach, has proven to be a difficult task.

The genetic algorithm was unable to reproduce the trajectory in any single run. However, a method has been devised, in which a trajectory can be found through multiple consecutive runs of the genetic algorithm where the variable limits for the launch date and mission length minimums and maximums are slowly trimmed. The process is able to converge to a trajectory that is very close that the one flown for the actual mission after four separate runs with a population size of 40,000 and 500 generations. Like the Galileo mission, the maximum launch v_∞ is limited, in the case to the actual flown value of 4.25 km/s. In the MGA-DSM model the launch v_∞ is explicitly constrained, so no additional penalties were added in. In this case the departure v_∞ is not added into the final cost function. The ΔV necessary to insert the spacecraft into a Saturn orbit with an eccentricity of 0.98 and a perigee radius of 80,330 km is also part of the cost function.

Table 4. Results for the Cassini mission compared to the actual results.

Mission Parameter	Cassini	Calculated
Launch Date	22-Oct-97	14-Oct-97
$v_{\infty-0}$	4.250	4.132
DSM (km/s)	0	0.006
E-V TOF (days)	194.000	194.521
DSM (km/s)	0.471	0.423
V-V TOF (days)	425.000	424.004
DSM (km/s)	0.000	0.193
V-E TOF (days)	54.000	54.729
DSM (km/s)	0.000	0.004
E-J TOF (days)	499.000	500.000
DSM (km/s)	0.376	0.113
J-S TOF (days)	1267.000	1266.000
Δ $\mathbf{V}_{arr}$	0.6	0.626
Total Cost (km/s)	1.4	1.364

A comparison of the results in Table 4 shows that the trajectories are similar, with a couple of minor discrepancies in when the deep space maneuver would be performed and how large it would be. In the computed trajectory the first large deep space maneuver occurs on the same mission leg, between Venus gravity assists, and is within 50 m/s of the actual trajectory. However, the computed trajectory has an additional deep space maneuver between the Venus and Earth gravity assists of nearly 200 m/s. The last deep space maneuver in the computed trajectory is 263 m/s less than the actual mission. In the end both trajectory have the same approximate cost. The Saturn insertion ΔV for the actual mission could not be determined, but was assumed to be approximately 600 m/s, putting the two trajectories well with a reasonable tolerance. The final trajectory is shown in Figs. 6 and 7.

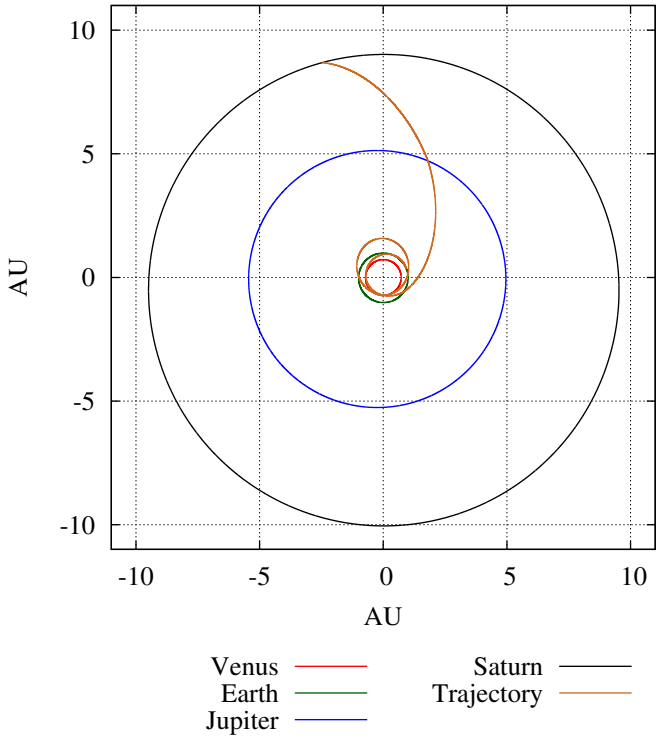

Figure 6. Trajectory for the optimal Earth to Saturn Mission.

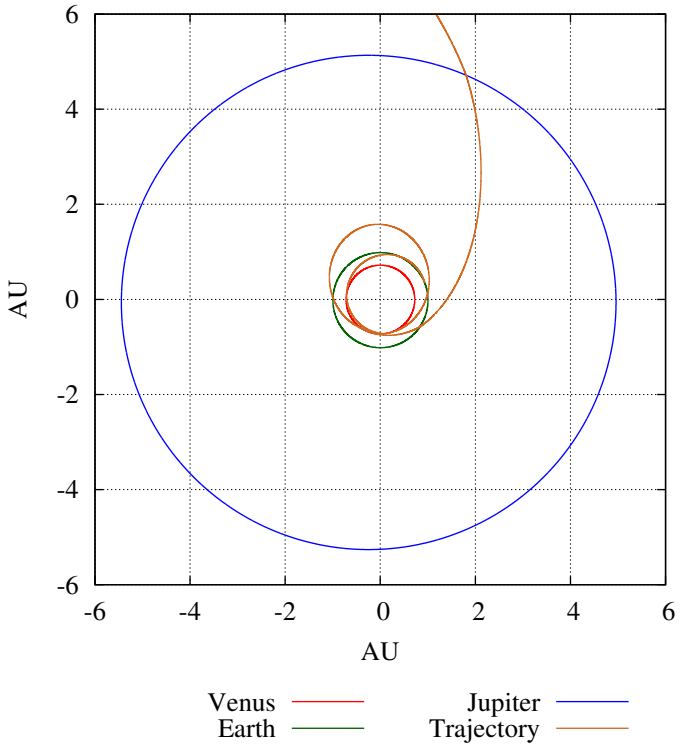

Figure 7. Trajectory showing inner planet flybys.

VI. Performance of Genetic Algorithm on GPU

With the benchmark performance testing of the genetic algorithm and mission design programs completed, it is worthwhile to compare the performances of the CPU and GPU version of the program. Both the CPU and GPU versions of the program use FORTRAN and CUDA FORTRAN respectively. FORTRAN was chosen due the performance increase over other languages and the relative easy of use of the PGI FORTRAN compiler. The CPU version was run using the GNU FORTRAN compiler using a standard workstation computer, see Table 5 for details. The CUDA FORTRAN version was run on a GPU compute cluster, Table 6, containing 4 NVIDIA Tesla c2060 graphics cards. As of the time of publication of this paper, the program can only use 1 GPU at a time.

Table 5. Information on the workstation used for the computational efficiency study.

Model	Dell T3500 Workstation
Operating System	Windows Vista Enterprise 64 bit
Processor	Intel(R) Xeon(R) W3520 2.67 GHz -4 core
Memory	6.00 GB 1066 MHz DDR3
Fortran	GCC v. 4.5.3

Table 6. Information on the GPU workstation used for the computational efficiency study.

Model	Amax ServMax Tesla GPU HPC
Operating System	RHEL v5.4
Processor	2x Intel Xon LGA1366 2.66 GHz -6 core
Memory	32 GB 1333 MHz DDR3
GPU	4x Tesla C2050

Performance improvement of the GPU algorithm is only possible for the complex Cassini type mission. On the CPU the Voyager 1 mission took about 2 minutes to run and the Galileo mission took approximately 20 minutes. The speed up with the GPU implementation of the genetic algorithm that the majority of the time is spent on initializing the GPU. Thus comparing the performance of the GPU genetic algorithm is inaccurate for these two missions. However, for the Cassini mission a speed up of over 9700% was realized. The total run time went from 10 hours (36,000 seconds) on the CPU to just 370 seconds (just over 6 minutes) utilizing a single Tesla C2050 GPU.

Such a performance increase will allow many (likely in the thousands) of complex mission to asteroid and outer planets to be analyzed by the mission designer. These mission types include, but are not limited to, multiple asteroid flybys, low-cost planetary defense demonstration missions, and possible multiple asteroid rendezvous mission (detailed surface and gravitational mapping or to more accurate pinpoint PHO positions).

VII. Conclusions

It has been shown that when used correctly, genetic algorithms can be powerful tools for the mission designer. By utilizing modern GPUs in combination with the genetic algorithm, it has also been shown that mission designs similar to Cassini (one of the most complex deep space mission ever flown) can be optimized in only a few minutes. In the specific case of reproducing the Cassini

Table 7. Performance of CPU and GPU versions of the program for the Cassini mission.

	CPU Version	GPU Version
Total Run Time (sec)	36004.544	370
Total Run Time (min)	600.076	6.167
Speed Up over CPU	1.000	97.310

mission, the algorithm took just over 6 minutes to produce near optimal solutions.

These new mission design capabilities will allow the mission designer to perform extremely complex mission analysis and design in an extremely short amount of time. Future versions of the program will continue to be further optimized and adapted for increasingly complex mission types.

Appendix

A few of the noteworthy functions required for the genetic algorithm are outlined in this appendix. Although crucial to implementing a genetic algorithm, random number generators and sorting functions, are beyond the scope of this paper. They are not included in this appendix or anywhere else in this paper. Further information can be found in Refs. [15–17]

Number of Bits per Gene

$$b = INT\left(\frac{\ln\left(\frac{X^U - X^L}{r} - 1\right)}{\ln(2)} + 1\right) \tag{43}$$

This equation is derived by solving Eq. 37 for b, the number of bits required for the desired variable resolution. The 1 is added to ensure that the minimum accuracy is preserved when the integer function rounds down.

Converting Binary Gene to Real Number Value

The conversion from a single gene to the gene's corresponding value is outlined below. A is the gene's binary string, b is the number of bits in the gene, and X^U and X^L are the user supplied upper and lower bounds the the variable represented by the gene.

$$sum = \sum_{i=1}^{b} A(i)2^{i-1} \tag{44}$$

$$scale = \frac{X^U - X^L}{2^{b-1}} \tag{45}$$

The final variable value X is then:

$$X = X^L + scale \cdot sum \tag{46}$$

Acknowledgments

This work was supported by the Iowa Space Grant Consortium (ISGC) through a research grant to the Asteroid Defection Research Center at Iowa State University. The authors would like to thank Dr. Ramanathan Sugumaran (Director, ISGC) for his interest and support of this research

work. In addition the author would like to thank Jacob Englander at the University of Illinois for providing data and support to help debug the MGA-DSM part of the program.

References

[1]D'Amario, L., Bright, L., and Wolf, A., "Galileo Trjectory Design," *Space Science Review*, 1992.

[2]Vasile, M. and De Pascale, P., "On the Preliminary Design of Multiple Gravity-Assist Trajectories," *Journal of Spacecraft and Rockets*, Vol. 43(4), 2009, pp. 794–805.

[3]Englander, J., Conway, B., and Williams, T., "Optimal Autonomous Mission Planning Via Evolutionary Algorithms," *21th AAS/AIAA Space Flight Mechanics Meeting*, Vol. AAS-11-159, Feb. 2011.

[4]Bate, R., Mueller, D., and White, J., *Fundamentals of Astrodynamics*, Dover Publications, Inc., 180 Varick Street, New York, NY, 1st ed., 1971.

[5]Battin, R., *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*, AIAA Educational Series, 1801 Alexadner Bell Drive, Reston, VA, revised edition ed., 1999.

[6]Vallado, D., *Fundamentals of Astrodynamics and Applications*, Microcosm Press, 401 Coral Circle, El Segundo, CA, 2nd ed., 2004.

[7]Wie, B., *Space Vehicle Dynamics and Control*, American Institute of Aeronatics and Astronautics, Inc., Reston, VA 20191, 2nd ed., 2008.

[8]Gooding, R., "On the Solution of Lambert's Orbital Boundary-Value Problem," *Royal Aerospace Establishment*, Apr. 1988.

[9]Conway, B., *Spacecraft Trajectory Optimization*, Cambridge University Press, 32 Avenue of the Americas, New York, NY 10013-2473, USA, 1st ed., 2010.

[10]Qadir, K., *Mult-gravity Assist Design Tool for Interplanetary Trajectory Optimisation*, Master's thesis, Lulea University of Technology, 2009.

[11]Vavrina, M., *A Hybrid Genetic Algorithm Approach to Global Low-thrust Trajectory Optimization*, Ph.D. thesis, Purdue University, 2010.

[12]Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1st ed., 1989.

[13]Koza, J., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, Cambridge, Massachusetts, 1st ed., 1992.

[14]Syswerda, G., "Uniform Crossover in Genetic Algorithms," *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989, pp. 2–9.

[15]Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., *Numerical Recipes in Fortran: The Art of Scientific Computing*, Cambridge University Press, 2nd ed., Sept. 1992.

[16]Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical recipes in Fortran 90 (2nd ed.): the art of parallel scientific computing*, Cambridge University Press, New York, NY, USA, 1996.

[17]Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E., *Introduction to Algorithms*, McGraw-Hill Higher Education, 2nd ed., 2001.