(Preprint) AAS 20-686

IMPLEMENTATION OF DEEP SPACE MANEUVERS IN BROAD SEARCH TRAJECTORIES USING MONTE CARLO TREE SEARCH

Burton A. Yale[*], Jehosafat J. Cabrera[†], Rohan D. Patel[‡], and Navid Nakhjiri[§]

Multiple flybys of the inner planets and the application of V_∞ leveraging are essential trajectory design techniques to reduce the required launch energy for interplanetary missions. These trajectories are often difficult to formulate and require extensive computational resources. However, this problem can be classified as a sequential planning task which can be solved by a Monte Carlo Tree Search (MCTS) method. In this paper, a MCTS algorithm is developed and tested with a focus on reducing required ΔV from the spacecraft. This method balances exploration and exploitation of the search space, and the algorithm's performance is assessed by tweaking the selection policy parameter. Optimizations of several cases are studied to prove the feasibility of the tree search results. The algorithm will allow for inner-planetary flyby search planning for outer planetary missions.

INTRODUCTION

Broad trajectory searches are one fo the first steps in mission planning, allowing for the selection of candidates given a list of constraints and a search space. As is the nature with combinatorial problems, every additional flyby adds another dimension to the search space. Brute force methods lead to expensive but thorough results. By pruning possible choices before the each selection, the overall computation time can be decreased, at the expense of losing possible results. There have been various approaches to this method, from evolutionary/genetic algorithms, to particle swarm optimization, to enumerated searches, all with their own benefits and drawbacks. The method of interest for this algorithm was enumerated searches, a subset of grid searches, as recent work has proven their ability to quickly and accurately find multi-leg interplanetary trajectories.[1]

Monte **C**arlo **T**ree **S**earch (MCTS), is a heuristics-based adaptation of the enumerated search, has the ability to both explore and exploit its environment. This allows the algorithm to find the narrow bands where multiple planet combinations are possible and exploit them to complete its goal. As such, larger search spaces can be employed without exponential growth in computation time.

This algorithm will deliver a set of planetary sequences and the dates associated that meet mission criteria, such as destination, launch windows, total fuel budgets, etc. These results can then be passed on to the next phase of the design, optimization

[*]Undergraduate Student, Aerospace Engineering, Cal Poly Pomona, 3801 W Temple Ave, E-mail: bayale@cpp.edu
[†]Undergraduate Student, Aerospace Engineering, Cal Poly Pomona, 3801 W Temple Ave, E-mail: jehosafatc@cpp.edu
[‡]Undergraduate Student, Aerospace Engineering, Cal Poly Pomona, 3801 W Temple Ave, E-mail: rohanpatel@cpp.edu
[§]Assistant Professor, Aerospace Engineering, Cal Poly Pomona, 3801 W Temple Ave, E-mail: nnakhjiri@cpp.edu

BACKGROUND

Interplanetary mission design is an iterative process. First, a sequence of encounter bodies is selected and is followed by evaluating all possible trajectories for that specified sequence. In literature, this is often referred to as pathfinding and path solving respectively [3]. In preliminary design, due to the increasing complexity of gravity assists and combinatorial solutions, a low-fidelity tool is generally utilized first to find regions of interest. This is then followed by a high-fidelity tool which searches through prospective and highly valued regions. These low-fidelity tools make use of algorithms to decrease computation time while increasing the number of possible solutions.

Grid-search, for instance has been previously used as a search algorithm to find possible trajectories to KBOs [4]. Grid search is a type of search algorithm with the ability to map the entire search space so that there are no regions of interest missed. This method has been used to properly parametrize the time of flight between encounter bodies. In this manner, Earth having a period of 365 days will have the same number of possible encounters as Jupiter with a period of approximately 12 years [2]. Thus, the output grid will be that of an angular grid as opposed to a cartesian grid. This type of search algorithm, however, can be expensive in terms of computing power and time. To alleviate both of these constraints, heuristics can be implemented into the program.

Beam Search (BS), a heuristic tree search algorithm, can be used as a searching criterion to find a sequence of planets from which gravity-assist from [4]. BS uses the method of Breadth-First Search (BFS) to search the tree of possibilities. BS builds each layer of the tree and orders the nodes in accordance to their heuristic cost. However, it only chooses those nodes with a maximum value to build from. Depth-First Search (DPS), alternatively, is used as a searching criterion to traverse down the tree [5]. Opposite to BFS, DPS does not search the tree at every level but rather explores a branch until the termination criteria is met. After which, moving in reverse updates the branch and moves towards the root node to start the process again. Both the BFS and DPS, search and prune the search-space consecutively. Using the Lazy Race Tree Search, the search space is pruned, and nodes ranked through the time of flight [5]. The use of heuristics avoids finding redundant solutions and increases the efficiency of the method used. Izzo et. al. [2] proposed the use of the Monte Carlo Tree Search to find fast solutions to interplanetary trajectories.

The search sensitivity criteria of DPS, BS, and BFS are explored in conjunction with that of the MCTS and compared. These searches, however, did not take into account Deep Space Maneuvers (DSMs) and led to trivial solutions when trying to validate the Rosetta Trajectory. The validation and correct recreation of this mission will be the basis of the algorithm taking into account DSMs to fully model Leveraging Orbits.

MONTE CARLO TREE SEARCH IMPLEMENTATION

This tree is built upon a set of individual nodes, that are all connected to each other through their parent and children, much like a family tree. Each node has an associated state to represent a leg in the tour of a spacecraft, with a planetary body, and the time at which it is encountered. The process of building the tree throughout a run of the algorithm can be characterized by four essential steps: selection, expansion, simulation, and backpropagation. Figure 1 depicts how the tree changes with respect to each phase of the MCTS loop.

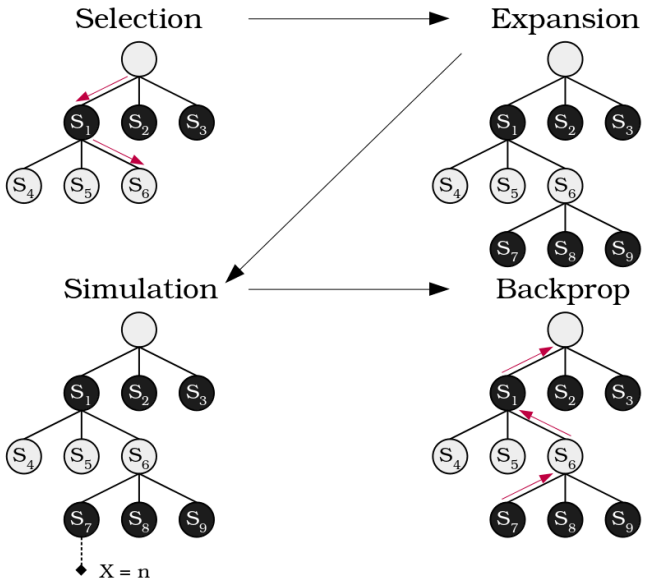

Figure 1. 4 Main Steps for the Creation of a Monte Carlo Tree

Selection

At the start of each iteration, the program begins at the root of the tree $id = 0$, and from its children, select the child with the highest associated UCB1 value. With X being the future reward from exploring the node, N being the number of visits the node has recieved, and n being the number of visits the parent node has recieved.

$$\text{UCB1 Node Value: } X + C_p\sqrt{\frac{\ln n}{N}}$$

In the case of C_p, the exploration-exploitation parameter, a value of $C_p = \frac{1}{\sqrt{2}}$ was selected due to its performance in [Hennes and Izzo, 2015].

As the function selects down the tree, it evaluates the all children of each node for is they lead to a terminal condition, such as running out of the fuel budget. If all child nodes are considered terminal, then the associated parent node is also considered terminal, and the function restarts at the top of the tree. This process was implemented in order to prune the tree of any branches that showed results initially, but lead to all terminal states. Once the function has reached a leaf node, it will return the id associated and move onto the next step in building the tree, expansion.

Expansion

Before the creation of the new set of nodes, the function will first look at the state of the current node being expanded from as a base. As planetary motion is a problem involving conic sections, cartesian grid position sampling is not viable, thus angular divisions are required to efficiently sample a planets position. When selecting states for new nodes, the following policy is implemented.

$$E = \begin{pmatrix} n*(\tau_1+\tau_0)+t_0 \\ \vdots \\ m*(\tau_1+\tau_0)+t_0 \end{pmatrix} \text{ for } \begin{cases} n = 0.1, & m = 1.0 & \text{if } a_1 < 2 \text{ AU} \\ n = 0.05, & m = 0.25 & \text{if } a_1 \geq 2 \text{ AU} \end{cases}$$

For the ephemeris time array E, the time of flight of the Lambert arc is dictated by the period of the planet state of the child node, τ_1, and the period of the planet state of the parent node, τ_0. This array is bounded using the parameters n and m, which are defined by the semi-major axis of the child node's planet, a_1. The values for n and m were determined based on the performance of the tree, for outer planet destinations, early, faster, arrivals were prefered, so their parameters have been reduced to reflect as such.

With the upper and lower bounds of the array set, an additional parameter, defined by the user, d specifies the length of the array, also defining the resolution of the time steps. For the purpose of this paper, a value of $d = 16$ was found to be sufficient for all calculations as a balance between computation time and detail.

Once the list of states has been established, a new set of children nodes are created to match all combinations of ephemeris times and planetary id pairs. In order to further prune the tree to reduce unnecessary calculations, Lambert arcs are calculated to each state pair to evauluate their associated Δv cost. Any node that exceeds the mission budget is considered terminal and not used in any further exploration.

In order to calculate mission Δv usage, the simulation uses powered flybys, as characterized by [Wei] **MORE HERE LATER**

$$a_{\text{in/out}} = -\frac{\mu_p}{v^2_{\infty-\text{in/out}}}$$

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

$$f =$$

$$\frac{\mathrm{d}f}{\mathrm{d}e_{out}} =$$

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

$$\Delta V_{fb} = \left|\sqrt{v^2_{\infty-\text{in}} + \frac{2\mu_p}{r_p}} - \sqrt{v^2_{\infty-\text{out}} + \frac{2\mu_p}{r_p}}\right|$$

Simulation

Backpropagation

DEEP SPACE MANEUVERS IMPLEMENTATION

- Rohan how dsm's work
- I talk about how its implemented

PERFORMANCE EVALUATION

Leveraging Maneuver to Saturn

- Figures
- States (Seq vs Vinf)
- Tables
- MCTS Inputs

Europa Clipper Trajectory Recreation

- Figures
- States (tof vs vinf)
- MALTO
- Tables
- MCTS Inputs
- MCTS Outputs

Triton Trajectory Recreation

Exploratory

- Figures
- States
- MCTS Trajectory Families
- Tables
- MCTS Inputs

```
def mcts.run():
    for _ in maxIterations:
        if all(launchNodes is terminal):
            break

        # Select Most Valuable Leaf Starting from Root
        id = select()

        # Expand If Prevously Visited and Pick Most Valuable Child
        if node.children is None and node.visits is not 0:
            id = expand(id)

        # Randomly Explores From Selected Node To Generate Cost
        X = simulate(id)

        # Propagates Results Up Branch
        backprop(id, X)
```

REFERENCES

[1] D. Hennes and D. Izzo, "Interplanetary trajectory planning with Monte Carlo tree search," IJCAI International Joint Conference on Artificial Intelligence, Vol. 2015-Janua, No. Ijcai, 2015, pp. 769–775.