(Preprint) AAS 23-246

ADAPTIVE ROADMAP GENERATION FOR TRAJECTORY DESIGN IN THE EARTH-MOON SYSTEM

Kristen L. Bruchko[*] and Natasha Bosanac[†]

In this paper, probabilistic roadmap generation is used to summarize a region of the solution space near the Moon in the Earth-Moon circular restricted three-body problem. This roadmap is a directed, weighted graph formed by adaptively selecting nodes that reflect states and edges that capture their connectivity. Then, the roadmap is rapidly searched using graph search algorithms to construct distinct initial guesses for transfers between periodic orbits. The goal of this approach is to reduce the reliance on a human-in-the-loop by constructing a single roadmap that can be repeatedly searched to construct initial guesses for various transfers.

1 INTRODUCTION

Designing spacecraft trajectories in chaotic multi-body systems currently relies heavily on preliminary analysis in low-fidelity models as well as the expertise and intuition of a trajectory designer. Current state-of-the-art approaches often leverage dynamical systems theory applied to simplified models such as the circular restricted three-body problem (CR3BP) to predict the types of natural motion in a system and their associated transport pathways. Then, a trajectory designer examines these solutions to select arcs that may satisfy a variety of constraints or possess desired geometries. These arcs are assembled to form an initial guess that is corrected to produce a continuous trajectory. However, as mission requirements, hardware capabilities, or path constraints become increasingly complex, an expert trajectory designer may struggle to both examine the space of feasible motions or identify a viable initial guess. Furthermore, with continuous control, the array of possible motions may deviate significantly from natural dynamical structures generated in low-fidelity models. As a result, there are scenarios where it may be difficult to use existing approaches for initial guess construction and to examine a broader solution space. Inspired by these challenges, path planning techniques may offer alternative approaches that reduce the reliance on a human-in-the-loop as well as the a priori knowledge required to rapidly design trajectories in a multi-body system.

Path planning techniques are used in various disciplines to construct collision free paths that adhere to geometric and dynamical constraints.[1] Path planning algorithms, also commonly referred to as motion planning, are often used in robotics, artificial intelligence, and control theory.[2] Sampling-based planning, specifically, relies on sampling to construct a graph of the environment. Probabilistic roadmap generation is one sampling-based technique that focuses on constructing a graph, called the roadmap, that is a summarized representation of the environment. The graph consists of nodes, which are valid configurations that an object may possess within its environment, and

[*]Graduate Researcher, Colorado Center for Astrodynamics Research, Smead Department of Aerospace Engineering Sciences, University of Colorado Boulder, Boulder, CO, 80303.
[†]Assistant Professor, Colorado Center for Astrodynamics Research, Smead Department of Aerospace Engineering Sciences, University of Colorado Boulder, Boulder, CO, 80303.

edges, which are valid directed paths between neighboring nodes and are weighted to reflect the difficulty of traversing the path.[3] Valid nodes or edges in the graph adhere to static and/or dynamic constraints. Once the roadmap is considered complete, it may be rapidly queried to construct multiple unique paths between specified start and goal configurations that have been projected onto the roadmap. Sampling-based planners have been demonstrated to solve high-dimensional problems in complex spaces with probabilistic completeness, i.e., the probability of finding a solution if one exists approaches unity as the number of nodes increases.[3]

In astrodynamics, path planning techniques have previously been applied to trajectory design problems. Rapid search methods have been implemented by Das-Stuart, Howell, and Folta to search the accessible regions of a spacecraft with a database of possible trajectories to autonomously construct initial guesses for trajectories.[4] Additionally, Trumbauer and Villac have constructed weighted and directed graph representations from precomputed databases of known dynamical structures in lower-fidelity models, and heuristically searched the graph for transfers during onboard trajectory redesign.[5] Sampling-based motion planning, in particular a tree algorithm, has been applied for real-time, propellant-optimized autonomous spacecraft trajectory generation by Starek et. al.[6] Our ongoing work has built upon the foundation supplied by these contributions by using probabilistic roadmap generation to construct initial guesses for natural and maneuver-enabled transfers between Lyapunov and halo orbits, incorporating information from their stable and unstable manifolds into the roadmap.[7,8]

In this paper, we construct a roadmap as a directed and weighted graph that summarizes part of the solution space near the Moon at a single energy level in the Earth-Moon CR3BP. In our previous papers, we generated these graphs using information predominantly sampled from precomputed stable or unstable manifolds and with fixed parameters that govern the number and length of edges generated. However, this paper focuses on an updated approach that reduces the reliance on a priori knowledge, fixed parameters, and natural dynamical structures that exist in low-fidelity models. The goal of these updates is increasing the generality of the approach and diversity of the solution space that can be represented. First, nodes are iteratively added to the roadmap using both random sampling and the centroids of tetrahedra constructed via Delaunay triangulation in the configuration space. Next, natural and maneuver-enabled edges are added between a subset of the nodes to sufficiently capture the connectivity of the graph and characteristics of the solution space. During this step, local Lyapunov exponents, which capture the sensitivity of a trajectory to variations in the initial conditions, are used to adjust the number and length of edges that connect two neighboring nodes.[9] Nodes and edges are iteratively added to the graph until the nodes sufficiently cover the desired region of the solution space and the edges sufficiently capture their connectivity.

Once the roadmap is considered complete, a graph search method is used to construct initial guesses for trajectories between specified start and goal configurations from the roadmap. Dijkstra's search algorithm is used to search for paths within the roadmap that are optimal with respect to a selected heuristic, such as minimizing the cumulative edge weight across the trajectory. In this paper, we apply Dijkstra's search algorithm multiple times to subgraphs of the roadmap, following the procedure for Yen's algorithm.[10] This enables the identification of multiple initial guesses for transfers with distinct geometries connecting a fixed combination of start and goal configurations. These initial guesses are then input to a corrections scheme that uses collocation to produce a set of continuous trajectories with distinct geometries. To demonstrate this process, we search a single roadmap, constructed at $C_J = 3.15$, to identify two initial guesses for geometrically distinct, natural, and planar transfers from an L_1 Lyapunov orbit to an L_2 Lyapunov orbit.

2 BACKGROUND: DYNAMICAL MODEL

The CR3BP is an autonomous dynamical model used to approximate the motion of a spacecraft, under the point mass gravitational influence of two larger primary bodies, P_1 and P_2. The primaries are assumed to have a constant mass and travel in circular orbits about their mutual barycenter, whereas the spacecraft is assumed to have negligible mass relative to P_1 and P_2.[11] Mass, length, and time quantities are nondimensionalized such that both the mean motion of the primaries and the assumed constant distance between the primaries is unity. Furthermore, the mass ratio of the system, μ, represents the ratio of the mass of the smaller primary to the total mass in the system and the nondimensional mass of P_2. Then, a P_1-P_2 rotating frame is defined with the origin at the barycenter of the system and axes $\hat{x}, \hat{y}, \hat{z}$: $\hat{x}$ is directed from P_1 to P_2, $\hat{z}$ is aligned with the orbital angular momentum vector of the primary system, and $\hat{y}$ completes the right-handed orthogonal triad.[11] The nondimensional spacecraft state in the rotating frame is $\boldsymbol{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T$ and the associated equations of motion are written as[11]

$$\ddot{x} = 2\dot{y} + \frac{\partial U}{\partial x}, \qquad \ddot{y} = -2\dot{x} + \frac{\partial U}{\partial y}, \qquad \ddot{z} = \frac{\partial U}{\partial z} \tag{1}$$

where $U = \frac{1}{2}(x^2 + y^2) + (1-\mu)/r_1 + \mu/r_2$ and the distances of the spacecraft from P_1 and P_2 are $r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2}$ and $r_2 = \sqrt{(x-1+\mu)^2 + y^2 + z^2}$, respectively. This dynamical model possesses an integral of motion in the rotating frame, the Jacobi constant, that equals[11]

$$C_J = 2U - \dot{x}^2 - \dot{y}^2 - \dot{z}^2 \tag{2}$$

At a single value of the Jacobi constant, a variety of fundamental solutions exist such as five equilibrium points, continuous families of periodic and quasi-periodic orbits, and their associated hyperbolic invariant manifolds.[12]

3 BACKGROUND: NUMERICALLY CORRECTING TRAJECTORIES

To compute a continuous transfer, a two-point boundary value problem is solved using a free variable and constraint vector formulation of collocation. This approach closely follows the generalized formulation used by Grebow and Pavlak and Pritchett to recover continuous trajectories in a multi-body system.[13,14]

3.1 Collocation

Collocation is a numerical technique for implicitly integrating differential equations of a dynamical system.[13] A solution is recovered by computing piecewise polynomials that approximate a trajectory and satisfy the dynamics at collocation points. First, an initial guess for a trajectory is discretized into a mesh of boundary nodes along the trajectory consisting of s segments, resulting in $s+1$ boundary nodes. Then, the i-th segment is discretized into p_i arcs, each sampled to produce several nodes. For clarity, the following notation will be used for the remainder of this paper: the state and time measured from the beginning of the trajectory at the k-th node on the i-th segment and j-th arc are denoted as $\boldsymbol{x}_i^{j,k}$ and $t_i^{j,k}$, respectively. The time τ along the j-th arc along the i-th segment is normalized to $[-1, 1]$ using the following transformation:

$$\tau = \frac{2}{\Delta t_i^j}(t - t_i^{j,n}) - 1 \tag{3}$$

where the integration time along the arc is $\Delta t_i^j = t_{i+1}^{j,1} - t_i^{j,n}$.

Each arc is discretized into boundary nodes and collocation nodes using an n-th order polynomial and a node spacing strategy. The accuracy of the final solution depends on the selection of the polynomial and node spacing strategy: lower order polynomials may be used in conjunction with a finer mesh, while higher order polynomials may achieve a higher order accuracy with a coarser mesh. A 7^{th} order polynomial is used in this paper, consistent with previously demonstrated trajectory design implementations in multi-body systems.[13,14] Therefore, 7 collocation nodes are placed along each segment and the location of these nodes is selected by the node spacing strategy. A Legendre-Gauss-Lobatto (LGL) node spacing strategy is employed in this paper due to its higher order of accuracy and simplified design problem due to placing collocation nodes at the boundary nodes of each segment.[13,14] For this method, collocation nodes are placed at the boundary nodes of each segment as well as at the roots of the derivative of the $(n-1)$th polynomial at time τ.

Along each arc, the collocation nodes are categorized into free nodes and defect nodes. The free nodes are the odd-numbered nodes for a given arc while the defect nodes are the even-numbered nodes. The free nodes, labeled $\boldsymbol{x}$ are used to construct the polynomial representation for each state component along an arc while the defect nodes, labeled $\boldsymbol{p}$, are the locations where the polynomials are evaluated and compared to the system dynamics. Figure 1 conceptually depicts a segment discretized into two arcs using 7 collocation nodes described by the 7th order polynomial representation.[14] Free nodes are depicted as blue triangles, defect nodes are depicted as red circles, and the arc boundaries are the free nodes outlined in black. Because the i-th segment is discretized into p_i arcs, the final state along arc j and the initial state along arc $j+1$ are shared, but consecutive segments do not share any nodes.

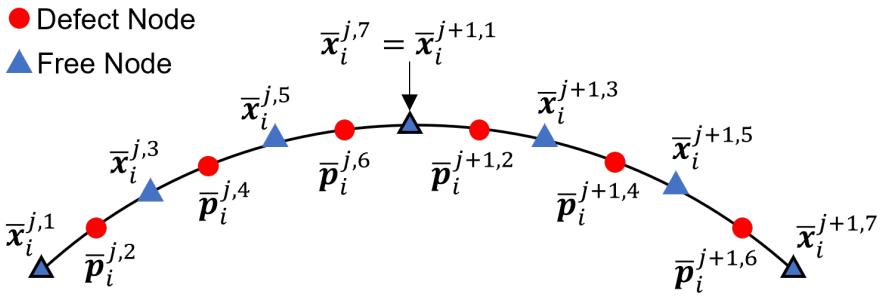

Figure 1. A conceptual example of the collocation nodes for arc j and $j+1$ along segment i using a 7th degree polynomial and a LGL node spacing strategy (Figure adapted from Pritchett, and Grebow and Pavlak.[13,14]

The design variables are the collection of all unique states at the free nodes along each arc as well as the nondimensional time for each arc so that the total time of flight may vary. Assuming each segment is discretized into p_i arcs, the free variable vector is the defined as:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{1,1} & \dots & \boldsymbol{x}_1^{1,n-2} & \Delta t_1^1 & \dots & \boldsymbol{x}_1^{p_1,1} & \dots & \boldsymbol{x}_1^{p_1,n} & \Delta t_1^{p_1} \\ \boldsymbol{x}_2^{1,1} & \dots & \boldsymbol{x}_2^{1,n-2} & \Delta t_2^1 & \dots & \boldsymbol{x}_2^{p_2,1} & \dots & \boldsymbol{x}_2^{p_2,n} & \Delta t_2^{p_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{x}_s^{1,1} & \dots & \boldsymbol{x}_s^{1,n-2} & \Delta t_s^1 & \dots & \boldsymbol{x}_s^{p_s,1} & \dots & \boldsymbol{x}_s^{p_s,n} & \Delta t_s^{p_s} \end{bmatrix} \tag{4}$$

and $\boldsymbol{X}$ is then reshaped into a single column vector.

The constraint vector captures both the continuity constraints between segments and the defect constraints for every defect node along the trajectory. First, the continuity constraint between the end of the i-th segment and the start of the $i+1$-th segment is defined as

$$\boldsymbol{F}_{c_i} = \begin{cases} \boldsymbol{x}_i^{p_i,7} - \boldsymbol{x}_{i+1}^{1,1}, \text{if no maneuver} \\ \boldsymbol{r}_i^{p_i,7} - \boldsymbol{r}_{i+1}^{1,1}, \text{if maneuver} \end{cases} \tag{5}$$

where $\boldsymbol{r} = [x, y, z]^T$. Then, the constraint vector that enforces continuity between neighboring segments along the entire trajectory is defined as

$$\boldsymbol{F}_c = \begin{bmatrix} \boldsymbol{F}_{c_1} & \boldsymbol{F}_{c_2} & \dots & \boldsymbol{F}_{c_{s-1}} \end{bmatrix}^T \tag{6}$$

Next, the defect constraints evaluate how well the polynomial satisfies the system dynamics at the defect nodes. The defect constraints for the j-th arc along the i-th segment for a 7th order polynomial are defined as

$$\boldsymbol{F}_{d_i}^j = \begin{bmatrix} \boldsymbol{\Delta}_i^{j,1} \\ \boldsymbol{\Delta}_i^{j,2} \\ \boldsymbol{\Delta}_i^{j,3} \end{bmatrix} = \begin{bmatrix} (\dot{\boldsymbol{p}}_i^{j,2} - \dot{\boldsymbol{x}}_i^{j,2}) w_2 \\ (\dot{\boldsymbol{p}}_i^{j,4} - \dot{\boldsymbol{x}}_i^{j,4}) w_4 \\ (\dot{\boldsymbol{p}}_i^{j,6} - \dot{\boldsymbol{x}}_i^{j,6}) w_6 \end{bmatrix} \tag{7}$$

where w_i are the quadrature LGL node weights and $\dot{\boldsymbol{p}}$ is the derivative of the polynomial state with respect to normalized time. Then, the defect constraints for the i-th segment are written as $\boldsymbol{F}_{d_i} = [\boldsymbol{F}_{d_i}^1, \boldsymbol{F}_{d_i}^2, \dots, \boldsymbol{F}_{d_i}^p]^T$. Finally, the complete constraint vector is defined as

$$\boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{F}_c & \boldsymbol{F}_{d_1} & \boldsymbol{F}_{d_2} & \dots & \boldsymbol{F}_{d_s} \end{bmatrix}^T \tag{8}$$

To recover a solution, the free variable vector is iteratively updated from an initial guess by applying Newton's method. These iterations continue until the norm of the constraint vector equals zero within a tolerance of 10^{-12}.

3.2 Mesh Refinement

Once the initial collocation problem is solved with a pre-specified discretization, the dynamics at each defect node are sufficiently satisfied by the polynomial constructed, but the trajectory may not be accurately approximated at other locations. Therefore, to improve the accuracy of the final solution, a mesh refinement process is used to remove and/or add arcs and adjust the location of the boundary points in the initial mesh. Following the approach presented by Pritchett, Control with Explicit Propagation (CEP) is used in this paper to update the number of arcs in the mesh.[13]

The CEP mesh refinement method involves two phases: arc removal and arc addition. After the initial collocation problem is solved, consecutive arcs are examined to determine if they can be combined using explicit propagation. First, the initial boundary node on the i-th arc is propagated for $t = \Delta t_i^j + \Delta t_i^{j+1}$. The error is then defined as the difference between the state at the end of this propagation time and the final boundary node on the subsequent arc. If the norm of this error is below 10^{-12}, then the shared boundary point between the two arcs is removed and the two arcs are combined into a single arc. Then, the free collocation nodes are recomputed using the LGL node spacing strategy using numerical propagation. If any arcs were merged during this first phase of mesh refinement, the collocation problem is solved again using the updated mesh. This process is repeated until the number of arcs does not change. Next, arcs are added where the accuracy of the

solution is not met to within some tolerance. The initial boundary node of each arc is numerically propagated for the integration time associated with the arc. The error for a specific arc is then computed as the difference between the final state from the numerical propagation and the final boundary node of the arc. If the norm of the error is above 10^{-12}, then the arc is split into two separate arcs of equal integration time. The collocation problem is then solved with the new mesh, repeating until no new arcs are added. Using mesh refinement in collocation results in a higher level of accuracy along the entire solution, verified via explicit propagation, and the accuracy of the final solution is no longer dependent on a user-defined initial mesh.[13]

4 BACKGROUND: PROBABILISTIC ROADMAP GENERATION

Probabilistic roadmap generation (PRM) is a sampling-based path planning method that constructs a summarized representation of an environment by creating a directed, weighted graph containing nodes and edges; this graph is referred to as the roadmap. PRM is a common path planning technique because it is a probabilistically complete planner: the planner will eventually compute a solution path as the number of nodes and edges in the graph grow.[3] The path planning process for PRM is divided into two main phases: the learning phase and the query phase. The learning phase is where the roadmap is constructed by sampling nodes and edges until the environment is sufficiently summarized. Once the roadmap is constructed, the graph can be queried rapidly and repeatedly for valid paths between start node(s) and goal node(s) using a graph search algorithm.

The two key elements within the roadmap are nodes and edges. Nodes are valid configurations within the environment that adhere to all static or dynamic constraints and do not intersect with any obstacles. Edges are paths between neighboring nodes that represent valid motion in a specific environment. Nodes added to the graph are determined by the selected sampling scheme, such as uniform, random, or informed sampling, or a combination of techniques.[3] Valid configurations are added as nodes until a sufficient number of nodes are contained within the graph. While this number is often one of the most difficult parameters to select, especially in unknown or complex environments, common methods include sampling a fixed number of nodes or sampling until a coverage or density metric is successfully evaluated. When a node evaluation metric is combined with incremental sampling, the resulting roadmap produced is often a sufficiently summarized representation of an environment. These two methods combined determine when to stop sampling nodes and may limit unnecessary or redundant information being added to the roadmap.[15–17]

Once the desired number of nodes are added to the graph, determined by the node evaluation metric, edges are then constructed as valid paths between neighboring nodes. For each node, a local planner is used to determine which neighboring nodes to connect through edges. Choosing an appropriate technique for determining the best neighbors for a given node may help increase the connectivity within the graph, ideally capturing all sensitive or narrow regions within an environment. Common techniques to identify neighbors include attempting to connect all nodes in simple environments, connecting nodes within a specified radius, or connecting nodes to only their k-nearest neighbors where k is either pre-defined or determined by local environment parameters.[3] Edges are either directed, meaning the node pair can only be traversed in one direction, or undirected. Edges may also have an edge weight that reflects either a desired characteristic of the environment or the likelihood of a specific edge being traversed. Selecting a heuristic for a given environment may aid the search algorithm in finding optimal paths with respect to this heuristic, e.g. if time along an edge is selected as the edge weight, a path with the shortest total time from the start node to the goal node may be found.

Given a completed roadmap, a graph search method is then used to query the roadmap for various paths from a given start node(s) to a goal node(s). A conceptual example of path planning using PRM appears in Figure 2, where nodes are indicated by circles, obstacles are indicated by squares, and directed edges are indicated with a dashed arrow. Each edge in the conceptual example also has an edge weight that the graph search algorithm aims to minimize across the final path from the start node, indicated by the blue circle, and the goal node, indicated by the red circle. A completed roadmap may then be queried additional times to produce a variety of paths between the same start and goal node(s) after the optimal path has been identified.

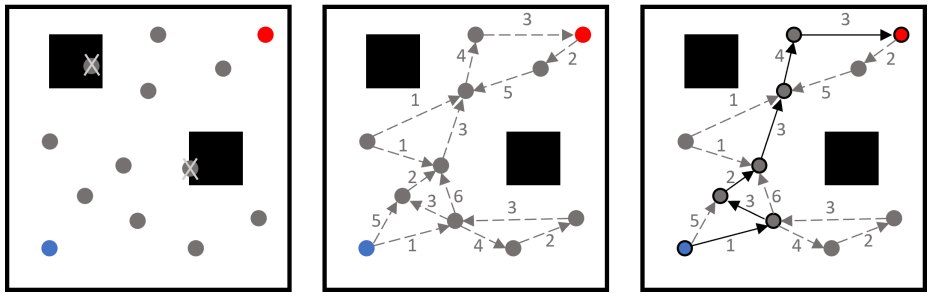

Figure 2. Example of probabilistic roadmap generation: a) sample valid nodes, b) connect nodes to create directed, weighted edges, and c) search the completed roadmap for an optimal solution path with respect to the sum of edge weights. Image reproduced from Bruchko and Bosanac 2022.[8]

5 BACKGROUND: GRAPH SEARCH METHODS

A graph search algorithm is used to solve a path planning query using a completed roadmap. Many graph search methods are commonly used in path planning problems, such as classical depth first searches or breadth first searches. However, the implementation depends on the desired characteristics of the algorithm for a given problem, such as computational efficiency or how many nodes need to be explored. For instance, Dijkstra's algorithm is a search algorithm used to find the shortest paths with respect to a selected heuristic between a source and destination within a graph. This algorithm is a best-first search algorithm that explores a graph by expanding in 'promising' directions; 'promising' nodes and edges have smaller weights or costs. Exploring the roadmap in this manner ensures Dijkstra's algorithm is computationally efficient because it does not need to explore every node in the graph and still guarantees it will find the optimal path with respect to a given heuristic.[2] It is also a complete algorithm, meaning it will terminate in finite time, even if no solution exists.

5.1 Dijkstra's search algorithm

Dijkstra's search algorithm creates a tree of the roadmap and uses both a priority list and a closed list to keep track of which nodes are currently being explored and which nodes have already been visited.[3] Given a start node(s), all neighboring nodes that are connected by an edge from the start node are added to the priority list. Then, the priority list is sorted based on the cumulative cost from the start node to the current node. The start node is then added to the closed list, because it has been explored and all of its neighbors have already been added to the priority list. The most 'promising' node at the top of the priority list is explored next. The 'promising' node is removed from the priority list and added to the closed list, and all neighbors of the 'promising' node are added to the priority list. All unexplored neighbors from the start node are also kept in the priority list. The

priority list is sorted again based on cumulative cost from the start node to the current node. This process is repeated until the goal node(s) is reached or all nodes appear in the closed list.

5.2 k-Shortest Unique Paths

Once a graph is completed and an optimal path is found, if one exists, it may be useful for additional paths to be explored. In a variety of path planning problems, using a repeating or modified graph search algorithm to search for a specific number of paths is common. For example, in a driving application, a second-optimal path that avoids traffic present on the time-optimal route might be of interest to a driver if the second route is only one minute longer. Searching for these additional paths is commonly denoted as the k-shortest paths problem. Developing algorithms for k-shortest paths has been of great interest over the past few decades with most algorithms improving upon Yen's algorithm; one of the first algorithms to search a directed and weighted graph for additional simple paths.[10] In this approach, $k-1$ additional paths of increasing sum of the desired heuristic, such as length or edge weight, are identified using Dijkstra's search algorithm to repeatedly search subgraphs of the completed roadmap. A subgraph is a graph with nodes and edges that span a subset of the completed graph.

Given a roadmap, a start node, and a goal node, k paths are identified from the same graph, or until no additional paths are found. A conceptual example of the k-shortest paths algorithm for a directed and weighted graph is depicted in Figure 3, where the start node is labeled 'A'; and the goal node is labeled 'E'. First, the shortest path is identified using Dijkstra's algorithm on the completed graph. This path is highlighted in blue in Figure 3a. To identify the second shortest path, three subgraphs are created because the first best path has three edges. These subgraphs are created by removing one edge from the first shortest path at a time. The edges that are removed from the graph have their edge weight set to infinity and are depicted in Figure 3b with dashed red arrows. Then, Dijkstra's graph search algorithm is used to search each subgraph for three candidates for the second shortest path. The second shortest path is then selected as the path with the lowest weight from the three candidate paths. The three subgraphs are displayed in Figure 3b; the second shortest path is drawn as the bold orange path in the black graph, while the two other candidate paths appear as light orange paths in the grey subgraphs. These two grey subgraphs and candidate paths are then discarded; they do not correspond to the third and fourth best paths. To identify the third shortest path (and subsequent $k > 3$ paths), the same process of creating subgraphs and producing candidate shortest paths is repeated with one additional step: each candidate path identified in the subgraph cannot be a previously identified shortest path. In this example, when the orange edge 'C-E' is removed from the second shortest path to create a new subgraph, the shortest path found would be 'A-C-D-E', which is already the first shortest path found. Therefore, there is only one candidate path for $k = 3$, resulting in the third best path shown in green in Figure 3c. This additional check ensures the top two paths are not repeated as the $k > 2$ shortest paths. This entire process is repeated until k paths are computed or until no more paths exist.

With largely connected graphs and the k-shortest paths algorithm, the k shortest paths may share a large number of edges that are the same. These returned paths may not be useful for certain path planning problems, such as a driver interested in comparing two distinct routes that take separate highways due to common traffic patterns rather than two routes that take the same highway and only vary by the selected exit. To overcome this challenge, it is useful to find the k-shortest unique paths. While the criteria that determine a unique, or sufficiently different, path may be different for each path planning problem, common approaches focus on identifying paths with limited overlapping

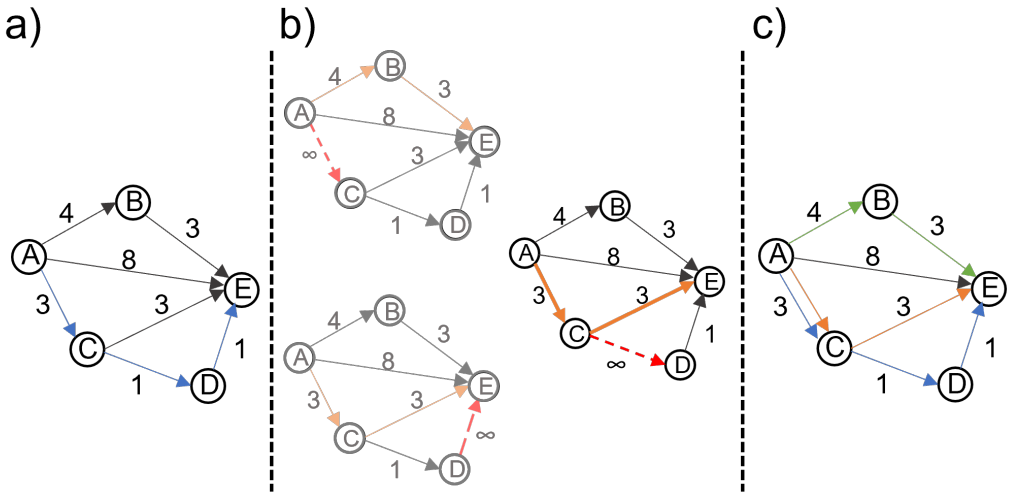

Figure 3. A conceptual depiction of the k-shortest paths algorithm for a directed, weighted graph with $k = 3$.

edges. These approaches often use similarity measures that compare the number of shared edges to the number of unique edges for each new path:[18] paths that share no edges have a similarity measure of 0, while paths that share every edge have a similarity of 1. Then, if the goal is to search for geometrically distinct paths, a designer may require that the k-th shortest distinct path must not share, for example, 50% of its edges with a previous shortest distinct path. For the example presented in this section, the orange path in Figure 3b would not be the second shortest distinct path because it shares one of its two edges (edge 'A-C') with the first shortest path. Therefore, the second shortest distinct path would be the green path in Figure 3c. Alternative approaches incorporate the geometric or topological differences between paths based on partitioning the graph into cells to determine which region the path lies in or which obstacles have been intersected.[19]

6 TECHNICAL APPROACH

In this paper, probabilistic roadmap generation is used to construct transfers between periodic orbits in the Earth-Moon CR3BP with minimal input from a trajectory designer. Following traditional roadmap generation methods, the approach in this paper is divided into two key phases: the learning phase, where the roadmap is constructed, and the query phase, where the roadmap is searched for valid paths. During the learning phase, the roadmap is constructed by: 1) sampling valid spacecraft states, within the desired solution space, that are saved as the nodes of the graph and 2) connecting nodes together to form the edges of the graph that represent valid motion between neighboring nodes. Nodes and edges are added to the graph until coverage and connectivity evaluation methods are successful, facilitating an efficient representation of the solution space without relying on extensive a priori knowledge about the environment. During the query phase, the optimal initial guess for a transfer between periodic orbits is identified by using Dijkstra's algorithm to search the roadmap. Then, additional unique initial guesses for transfers are constructed by repeatedly searching subgraphs of the roadmap using a modified Yen's algorithm along with a specified similarity measure. A conceptual overview of this approach is depicted in Figure 4. The learning phase and query phase are each described in more detail in the remainder of this section.

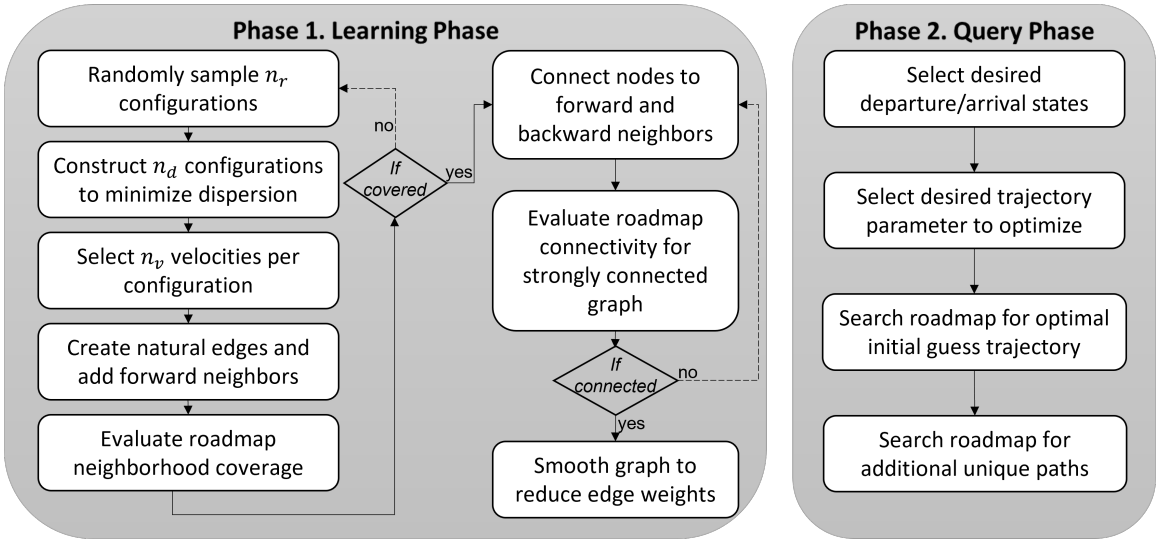

Figure 4. An overview of the key phases in roadmap generation used to construct initial guesses for transfers between two periodic orbits in the CR3BP.

6.1 Phase 1: The Learning Phase

The learning phase for this application of PRM to transfer design in the Earth-Moon CR3BP involves constructing a roadmap as a graph that sufficiently summarizes the solution space. During this phase, the roadmap is constructed incrementally with the following two steps: 1) adding nodes until the graph is sufficiently covered, and 2) constructing edges until the graph is strongly connected. Nodes are defined as spacecraft states expressed in the rotating frame while edges represent valid natural arc or arcs with impulsive maneuvers that connect existing nodes. Nodes are added to the graph until the desired solution space is sufficiently covered. Then, directed and weighted edges are added to connect selected nodes. A directed edge can only be traversed in one direction representing motion forward in time, while the edge weight reflects the ease or difficulty of traversing that edge with respect to a given heuristic. For this analysis, the heuristic used is the total maneuver magnitude required to traverse an edge, resulting in an edge weight of zero for natural trajectory arcs and a positive edge weight for maneuver-enabled arcs. Once the roadmap evaluation is successful, meaning the graph sufficiently covers the desired solution space and is strongly connected, the roadmap is considered completed. This iterative process helps the trajectory designer construct the roadmap without the need to specify how large the graph should be in order to sufficiently represent the solution space.

Nodes are defined by first randomly sampling n_r position vectors within the desired region of the solution space from a uniform distribution. These position vectors are then checked for validity; if any configuration intersects with any obstacles within the environment, another sample is drawn. For this paper, position vectors must lie within the allowable regions of motion for a given Jacobi constant and satisfy a 500km altitude constraint around the Moon. Then, the speed associated with each position vector for a fixed Jacobi constant is computed using Eq. 2. If the roadmap spans multiple energy levels, then the Jacobi constant is selected randomly. Once the speed is computed for each position vector, a velocity direction must be selected to define a complete state vector that serves as a node.

The velocity directions associated with each position vector are selected using the k-means clustering algorithm along with local sensitivity information. The k-means algorithm is a partition-based clustering method that partitions a dataset, described by some observations, into k clusters.[20] The k centroids then supply k representatives of the dataset. Given the input parameter k, the algorithm initially selects k random members of the dataset and assigns all remaining members to the nearest cluster defined by the least squared Euclidean distance. Then, the centroids are recomputed and members of a dataset are reassigned to clusters using the new centroids. This process is iteratively repeated until the centroids no longer change or a maximum number of updates have occurred.

For each position vector, k velocity directions are selected using k-means clustering, resulting in k unique nodes that share the same position coordinates. First, for each configuration, u unit velocity directions that span a sphere are sampled. Each of these unit vectors, $\hat{\boldsymbol{v}}_{\mathbf{0}}$, is multiplied by the speed and the resulting state vector is propagated for a finite horizon time, T_h, to produce an associated arc. This arc is characterized using a local Lyapunov exponent (LLE), which indicates the effect of perturbations or maneuvers on a nominal trajectory over the selected horizon time.[9] The LLE is computed along this arc, generated from an initial state $\boldsymbol{x}(t_0)$ for a horizon time T_h, as

$$\lambda|_{\boldsymbol{x}(t_0),T_h} = \frac{1}{T_h} \ln \left(maxeigenvalue \left(\sqrt{\Phi(T_h + t_0, t_0)^T \Phi(T_h + t_0, t_0)} \right) \right) \tag{9}$$

where $\Phi(T_h + t_0, t_0)$ is the corresponding state transition matrix. A feature vector, f_i, that characterizes the arc associated with the i-th velocity direction is then defined as

$$\boldsymbol{f}_i = [\hat{\boldsymbol{v}}_{\mathbf{0}}, \lambda|_{\bar{\boldsymbol{x}}(t_0),T_h}, v(T_h), \lambda|_{\bar{\boldsymbol{x}}(T_h),T_h}]^T \tag{10}$$

where $\lambda|_{\bar{\boldsymbol{x}}(T_h),T_h}$ is a second LLE computed from the state $\boldsymbol{x}(t_0 + T_h)$ propagated for a horizon time T_h. The dataset that describes all u velocity unit vector directions and is input to the k-means clustering algorithm is formed as $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_u]$. Then, the desired number of clusters k is calculated using the following function

$$k = \lceil \ln \left(max\{\lambda_1, \ldots, \lambda_u\} \right) \rceil \tag{11}$$

such that the number of selected velocity directions used to define k nodes is proportional to the estimated maximum sensitivity of any arc generated from that position vector propagated forward for time T_h. While the value of k depends on the horizon time selected, it typically ranges between $k = [2 - 6]$ for this example when $T_h = 2$ days. The k-means clustering algorithm is then applied to $\boldsymbol{F}$ to identify k distinct centroids and, therefore, k distinct velocity directions that are used to define k nodes with the same position components. These centroids help select velocity directions for each node that represent the sensitivity and natural motion of each cluster for the propagated horizon time.

Next, each configuration's neighborhood is defined by naturally propagating the states at all unique nodes forward in time to create a natural edge. For each natural edge, its propagation time is selected to be inversely proportional to its LLE as $t = 1/\lambda$, ensuring more sensitive regions have natural edges generated over shorter time intervals. The final state along each of the natural edges is then also saved as an additional node, if it is valid, and these nodes become children of the initial node. This information is also used to define a neighborhood in the vicinity of the k nodes with the same position components. This neighborhood is defined in the configuration space with a radius equal to the maximum distance to all of the child nodes, i.e., the states at the end of each

of the k natural edges. As an example, 50 neighborhoods are depicted in Figure 5, where nodes are indicated by the blue circles and the neighborhood of each computed configuration is indicated by the red circle. These neighborhoods do not cover the desired solution space yet because there are still neighborhoods that do not intersect any other neighborhoods, indicating they have no close neighbors.

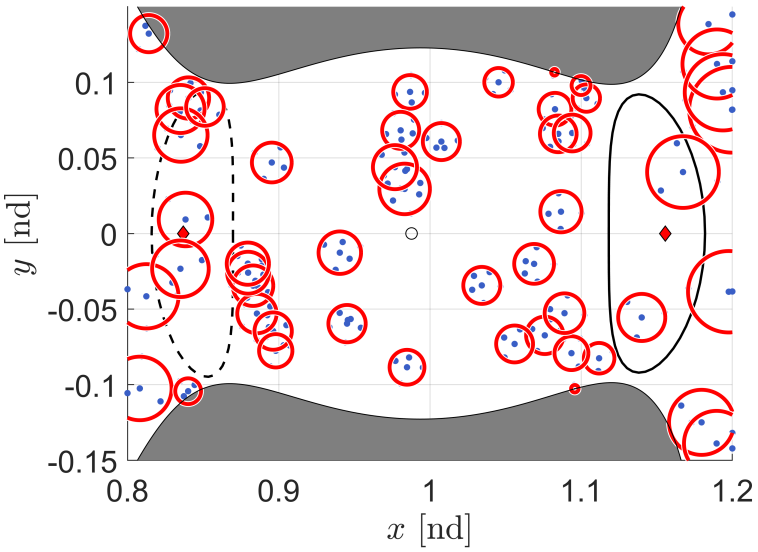

Figure 5. 50 neighborhoods for a roadmap that does not cover the desired solution space (depicted in white).

Once each randomly sampled configuration gains natural edges and children that lie within its neighborhood, additional nodes are added to the graph to minimize dispersion. The dispersion of the graph is defined by the largest empty sphere in the configuration space.[21] Under sampled regions in the configuration space are identified using a Delaunay triangulation (DT) between the current nodes in the graph, states along the zero velocity curves calculated at a specific Jacobi constant, and any additional desired solution space boundaries such as the L_1 or L_2 gateways. A Delaunay triangulation is constructed in the configuration space of the roadmap by creating a set of tetrahedra such that no nodes are within the circumcircle of any tetrahedra within the set. The Delaunay triangulation is a method commonly used in roadmap generation to aid in graph construction by identifying areas to add additional nodes or edges within the solution space since every tetrahedra contains no information already present in the graph.[22,23] Once the set of tetrahedra is constructed, the volume of each tetrahedron provides information about the volume of the encompassed region of the configuration space that does not contain any nodes. The largest tetrahedra are then used to identify regions within the configuration space that indicate under-sampled areas by the roadmap. For this implementation, the centroid of any tetrahedron with a volume that is greater than the top 5% of the volumes that all tetrahedra produce seeds the configuration of a potential node to add to the graph. This results in n_d potential configurations. If a centroid selected as a potential configuration is valid, it gains k velocity directions, k natural edges, and k neighboring children that all make up its neighborhood that are constructed in the same manner as described previously for the randomly sampled configurations.

Once $n_r + n_d$ neighborhoods are added to the graph, the coverage of the roadmap is evaluated. The coverage is defined by the number of neighborhoods that do not intersect any other neighborhood in the graph. These two methods for constructing neighborhoods by sampling configurations and spanning the velocity space using k-means is repeated until all neighborhoods are overlapping with at least one other neighborhood. This coverage definition ensures each neighborhood has a higher chance of being connected to other neighborhoods and increases the probability of creating a well-connected graph, while also facilitating an adaptive selection of the number of nodes and natural edges within the graph.[15]

Once the graph is covered, edges that represent maneuver-enabled arcs are added to the graph by connecting selected neighboring nodes. Each node gains up to k maneuver-enabled edges by identifying neighboring children forward in time and neighboring parents backward in time. The number of edges in each direction, k, is once again computed as a logarithmic function of the node's local sensitivity based on its LLE value using Eq. 11 with only one LLE value as the input. In asymptotic motion planning, k is selected as a logarithmic function of the number of nodes in the graph such that the number of edges increases as the number of nodes grows.[24] This function for selecting k is well-defined for linear simple spaces more relevant to robotic path planning, but does not account for various dynamics in an environment well. Inspired by this approach, this paper uses local sensitivity information instead to increase the number of edges in sensitive regions. Neighbors are then selected using a distance measure that captures the distance between two nodes in the configuration space as well as the angle between their velocity vectors and is defined as

$$dist = w_d \|\mathbf{d}_i(t) - \mathbf{d}_n(t_0)\| + w_t \theta \tag{12}$$

where $\mathbf{d}_i(t)$ is the position of the initial node propagated for time $t = 1/\lambda_i$, $\mathbf{d}_n(t_0)$ is the position of the neighboring node, and θ is the angle between the velocity vectors of the initial node after propagation time t and the position of the neighboring node. The weights, w_d and w_t are selected such that the two components of the distance metric are on the same order of magnitude. For this implementation, they are selected as $w_d = 10$ and $w_t = 1$. This approach captures the connectivity between nodes that would result in a lower edge weight while also discouraging edges that connect nodes that naturally flow in opposite directions.

Once k neighbors are selected for a node, k maneuver-enabled arcs are constructed to form the edges of the graph. Each neighbor is connected to the node using a single shooting corrections algorithm to construct a maneuver-enabled edge that allows the velocities at each node as well as the integration time to vary. To construct the initial guess, the initial node is propagated naturally forward in time. The time associated with the state along this arc that is closest in position space to the neighboring node seeds the integration time initial guess, while the velocity initial guess assumes no maneuver is required. The velocity differences between the states at the selected nodes and the beginning and end of the transfer arc then correspond to two impulsive maneuvers, and the sum of their magnitudes supplies the edge weight. Each edge is also subject to a 500km altitude constraint around the Moon; if a periapsis exists along the arc and falls within the desired altitude, the edge is not saved and a new neighbor is identified. If a maneuver-enabled edge is successfully connected, sequential quadratic programming within the fmincon tool in MATLAB® is used to adjust the edge to minimize the sum of the squares of the maneuver magnitudes.

After k neighbors forward in time and k neighbors backward in time are added for each node, the graph connectivity is checked by determining if the graph is strongly connected. A strongly connected graph guarantees that each node can reach every other node within the graph.[25] A digraph

is strongly connected if it satisfies two depth-first searches: 1) if a selected vertex can reach every other node in the original graph and 2) if a selected vertex can still reach every other node if the direction of all the edges were reversed. If the graph is not strongly connected, one additional forward and backward edge is added for each node until the graph is strongly connected.

Once the graph is completed, an additional post-processing step called path smoothing occurs. While this step can have a significant computational expense, it is included to improve the quality of the edges in the graph and the resulting solutions.[3] For this application, a sequence of edges are tested to determine if they can be combined to produce a smaller edge weight than the sum of the two individual edges. Smoothing the edges for a specific node could remove unnecessary maneuvers that may oppose each other. For a given node, one of its parents and one of its children are tested to see if a maneuver-enabled edge can be constructed using the previously described single shooting approach, bypassing the given node. If the resulting edge weight is less than the sum of the edge weights from the neighboring parent to the node and the node to the neighboring child, then the new edge is added to the graph. For each node, new connections are attempted between all neighboring parents and children. Including this post-processing step may produce a graph containing smoother edges with smaller edge weights.

6.2 Phase 2: The Query Phase

After the roadmap is constructed, the query phase searches the graph for valid initial guesses for transfers between periodic orbits. Given a state(s) on the initial periodic orbit used as the start node(s), a state(s) on the final periodic orbit used as the goal node(s), and the heuristic to optimize across the final path (in this paper, total maneuver magnitude on all edges), an initial guess for the optimal path is formed by applying Dijkstra's search algorithm to the completed roadmap. If no path exists between the selected start and goal node(s), then the search algorithm will terminate with no solution.

If an optimal path exists, additional unique paths between the same start and goal node(s) are identified by applying Dijkstra's search algorithm in conjunction with the modified similarity measure to multiple subgraphs of the roadmap. In this paper, unique paths are identified by modifying a common similarity measure that is a ratio of the sum of the edge weights of the shared edges between two paths, P_i and P_j, and the path with the maximum cumulative edge weight. The similarity measure in this paper is defined as:

$$Sim(P_i, P_j) = \frac{L(P_i \cap P_j)}{max\{L(P_i), L(P_j)\} + \sum_{n=1}^{d} \|\delta \bar{\boldsymbol{r}}\|} \tag{13}$$

where $L(\cdot)$ represents the cumulative weight of the path and the summation term in the denominator represents the position difference between the d unique nodes in each path. This term is added to reflect the geometric differences between two paths; the farther in distance the unique nodes are, the smaller the value of the similarity measure. If the unique nodes are close for both paths P_i and P_j, the similarity measure will not be affected as drastically. By using this modified similarity measure in conjunction with Yen's algorithm, multiple unique paths that are geometrically dissimilar may be identified by searching one completed graph.

All the initial guesses identified from the graph or subgraph consist of a sequence of nodes and edges that minimize the cumulative sum of edge weights across the path, where path $k-1$ always has a lower total edge weight than path k. The initial guesses selected for potential transfers are then corrected using collocation with a CEP mesh refinement process. The state of each node and

the integration time along the subsequent edge form the arcs used in the initial mesh, so the total time of flight may vary during corrections. If impulsive maneuvers are allowed on the transfer, or necessary if natural solutions do not exist, only position constraints are included in between segments. The locations of these maneuvers must be user-selected prior to corrections, although the initial guess may help the user select where to allow maneuvers by analyzing the edge weights: higher edge weights may indicate locations where a maneuver may be beneficial in recovering a nearby transfer.

7 RESULTS

7.1 Natural Transfers Between L_1 and L_2 Lyapunov Orbits at $C_J = 3.15$

To demonstrate the approach presented in this paper, a roadmap is constructed at $C_J = 3.15$ in the lunar vicinity in the Earth-Moon CR3BP. The solution space of interest lies within the zero velocity curves at $C_J = 3.15$ and possesses an x-component within the range $x = [0.8, 1.2]$, i.e., just outside the L_1 and L_2 gateways. All nodes and edges added to the graph are only considered valid if they are located within these bounds; any edge that leaves this region will not be saved to the graph. Two initial guesses for transfers between an L_1 Lyapunov orbit and an L_2 Lyapunov orbit are constructed from one roadmap by searching the graph and subgraphs repeatedly. Each initial guess is then corrected using the collocation corrections algorithm presented in Section 3 to recover multiple natural transfers. At this energy level, two direct heteroclinic connections exist, thus they will be used as a straightforward verification of the results.[26]

To begin the roadmap construction process, neighborhoods are constructed until the desired region of the solution space is sufficiently covered. During each iteration of sampling nodes and adding natural edges, 10 position vectors are randomly sampled and the centroids of the top 5% of tetrahedra, identified via Delaunay triangulation, are saved as potential position vectors for the nodes of the graph. If a larger number of position vectors are added at each iteration, a smaller number of iterations may be required to fully cover the solution space but also may oversample the environment. Accordingly, a smaller number of position vectors are added at each iteration to create a more efficient representation of the solution space. Each valid position vector is then used to define k velocity directions and, therefore, k neighboring child nodes. To determine k, the LLE is calculated using a horizon time of $T_h = 2$ days. This horizon time is selected because the local immediate sensitivity of each state is of greater interest than the sensitivity of the state much further away from the node and the edges constructed for this graph are often shorter arcs. While the number of natural edges and neighboring child nodes is unique to each position vector, a large majority of the nodes produce values in the range $k = [3, 7]$, depending on if the node is closer to the Moon or the x boundaries of the roadmap.

For this example, the roadmap requires 5 iterations of adding nodes and natural edges until the solution space is sufficiently covered. The distribution of nodes in the configuration space after the first iteration and after the final iteration are depicted in Figure 6a) and 6b), respectively. After only the first iteration, the nodes are not well-distributed within the desired region. However, the nodes sufficiently cover the solution space after the last iteration, increasing the likelihood of connectivity to all regions of the solution space. Once the desired solution space is covered, states along the initial and final orbits are also projected onto the roadmap. Each of these periodic orbits is computed using collocation and discretized evenly in time into 50 states along each orbit. These states along each orbit are used as the set of start nodes and final nodes used during the graph search.

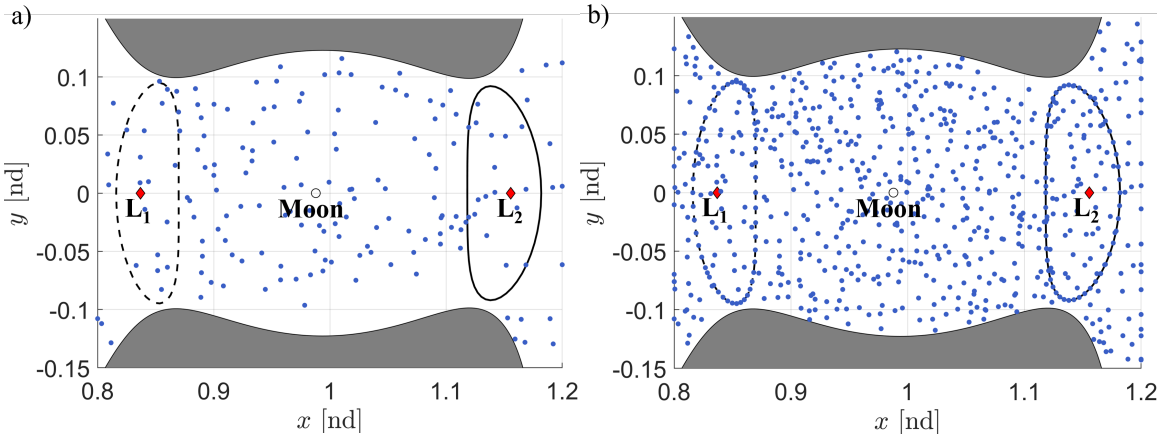

Figure 6. a) Distribution of nodes after the first iteration of the coverage evaluation b) Distribution of nodes after the roadmap is completed.

Each edge gains k maneuver-enabled edges both forward and backward in time and then the graph's connectivity is evaluated. For this example and with the selection of the k parameter, the graph is strongly connected after connecting k neighbors for each node. While only one iteration of constructing edges could indicate redundant edges, allowing some redundant edges has been observed to improve the quality of the original path without significant overhead caused during post processing of the roadmap.[3] Future investigations will include additional analysis on an efficient number of edges to construct a more simplified, but still summarized, representation. The completed roadmap for this scenario, including both nodes and edges, appears in Figure 7. In the roadmap, nodes are indicated by a blue circle, natural edges are indicated by a dark blue curve, and maneuver-enabled edges are indicated by a light blue curve. The desired region of the solution space is well-covered and well-connected by the nodes and edges of roadmap. Using the k-means clustering algorithm with an adaptive selection of k, there is a larger number of natural edges near the Moon where maneuvers may be larger, promoting natural motion in these regions during graph searches.

Once the roadmap is completed, the graph is searched using Dijkstra's search algorithm to produce the path with the minimum cumulative maneuver magnitude. This path may depart from any of the states along the L_1 Lyapunov orbit that were projected onto the roadmap and arrive at any of the states along the L_2 Lyapunov orbit that were projected onto the roadmap. This path corresponds to a sequence of nodes and edges that together form an initial guess for a transfer; along this initial guess, each arc that is represented by an edge may possess velocity discontinuities between its neighboring arcs. The initial guess, constructed from the completed roadmap in Figure 7, is displayed in Figure 8a). The initial and final Lyapunov orbits are plotted with dashed and solid black lines, respectively. Each edge color corresponds to its edge weight indicated by the colorbar, where lower cost edges appear blue and higher cost edges appear green. Next, the graph is smoothed to produce a slightly smoother initial guess. During graph smoothing, 2308 edges with lower weights were added to the graph that originally only contained 1253 edges. The initial guess constructed after the completed roadmap has been smoothed in displayed in Figure 8b), using the same color scheme for edge weights with a different colorbar. After smoothing the roadmap, the total cost, i.e. the sum of the edge weights, of the optimal initial guess decreased from 2.44 km/s to 1.21 km/s.

Once the optimal path has been found, the graph is searched for an additional path using Yen's algorithm with the modified similarity measure presented in Section 6.2 with the same initial start

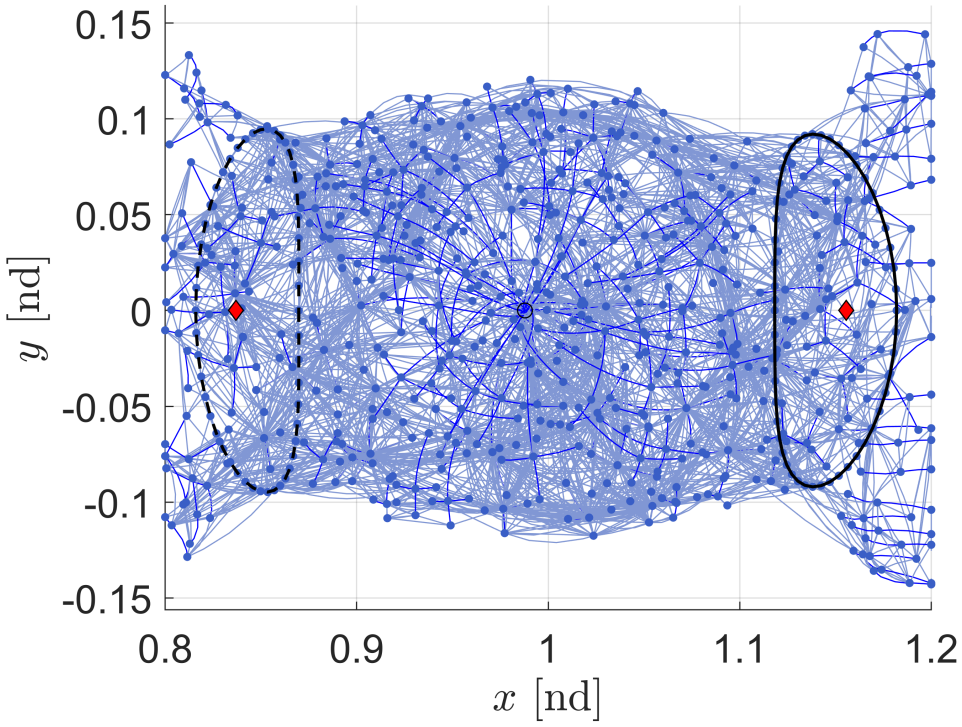

Figure 7. A completed roadmap generated to summarize the solution space in the lunar vicinity at $C_J = 3.15$.

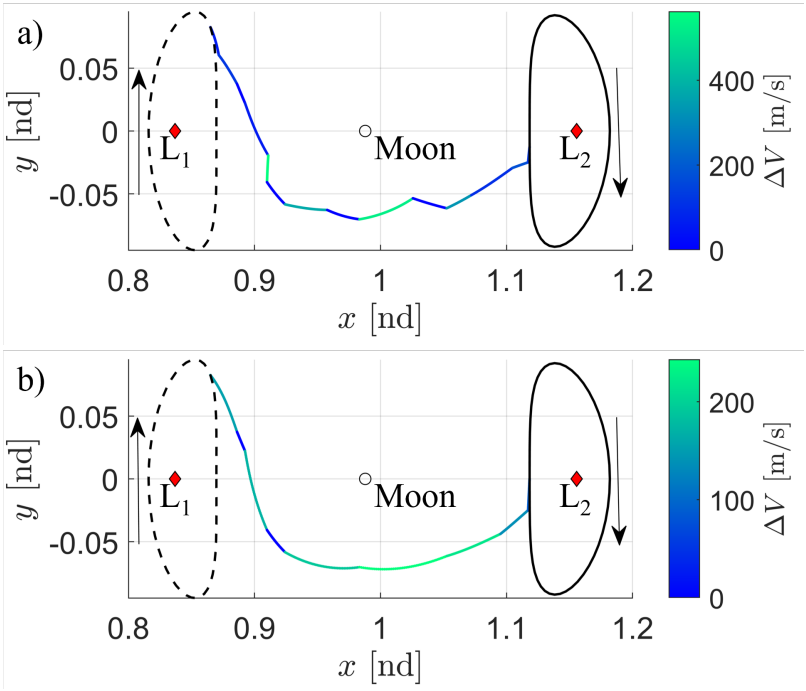

Figure 8. a) The initial guess constructed from the roadmap in Figure 7 and b) the initial guess constructed from the roadmap after it has been smoothed.

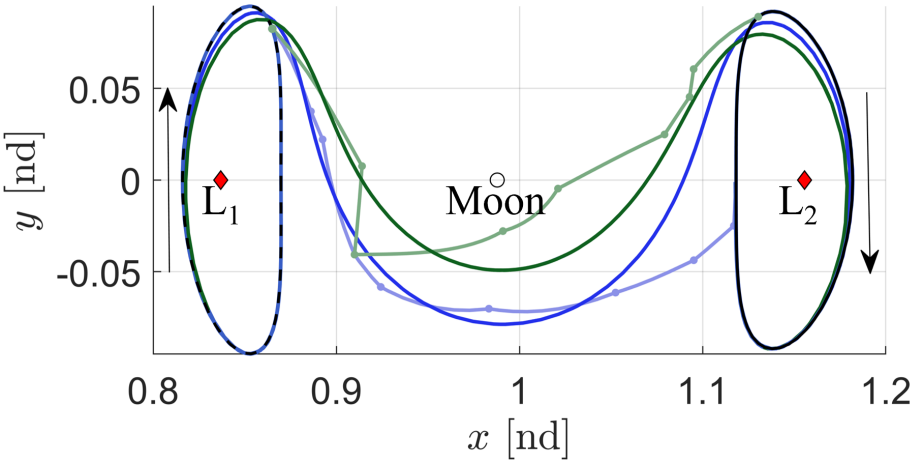

Figure 9. Two unique initial guesses (light blue, light green) for transfers and the corresponding two geometrically distinct continuous transfers (dark blue, dark green).

nodes along the L_1 Lyapunov orbit and the same goal nodes along the L_2 Lyapunov orbit. To distinguish between unique paths, a similarity threshold must be determined. For this example, a threshold of $Sim(P_i, P_j) < 0.60$ is selected to eliminate solutions that may vary by the number of shared edges but still look geometrically similar. The initial guesses are plotted as transparent curves in Figure 9. The second initial guess depicted in light green has a cumulative maneuver magnitude of 7.8 km/s, largely because of the second edge in the initial guess having an edge weight of 4.6 km/s. Due to this large overall path cost, the second initial guess is discovered as the 57th distinct path produced by Yen's algorithm.

Collocation is applied to each of the two initial guesses to produce two natural, planar transfers with no revolutions around the Moon. The resulting transfers are depicted in Figure 9 using solid curves with the same coloring scheme as the initial guesses. While the corrected natural transfers do not exactly resemble the initial guesses, this example demonstrates the feasibility of searching for various initial guesses from a single roadmap. These results also reveal useful avenues for further work. For instance, the deviations between the initial guesses and the corrected transfers indicate the need for further improvement of the roadmap: potentially increasing the number and distribution of nodes as well as improving the diversity of connections between nodes. The roadmap could also benefit from efficiently representing a wider array of natural motions, without biasing the roadmap too heavily, to guide the graph searches towards solutions with lower cumulative maneuver costs. Future improvements also include the computational efficiency of the search algorithms used to recover k shortest paths.

8 CONCLUSION

In this paper, PRM is used to summarize the solution space in the lunar vicinity in the Earth-Moon CR3BP. The roadmap is constructed in two key phases: the learning phase and the query phase. The learning phase consists of adding nodes, which represent valid spacecraft states, and edges, which represent valid trajectory motion, to a graph called the roadmap. Nodes and natural edges are added to the graph using a combination of randomly sampled motion and informed local region information to fully cover the solution space. The natural edges capture various sensitivities of distinct regions in the solution space while the nodes ensure each area within the graph is reachable.

Nodes and natural edges are iteratively added to the graph until the desired solution space is well-covered. Then, maneuver-enabled edges are added by connecting neighboring nodes that represent trajectory arcs with two impulsive maneuvers. The sum of the impulsive maneuvers is then saved as the associated edge weight, while the natural edges have an edge weight of zero. The number of edges each node gains is adaptively selected to reflect a higher number of edges in sensitive regions of the graph and promote better connectivity. Maneuver-enabled edges are added to the graph until the graph is strongly connected; each node can reach every other node in the graph by at least one path. Lastly, a post processing phase is applied to smooth edges within the completed graph resulting in smaller edge weights that promote better connectivity of the solution space and smoother solution paths.

Once the roadmap is completed, the graph is searched using Dijkstra's search algorithm for the optimal path that corresponds to a sequence of nodes and edges that minimizes the cumulative sum of the total edge weights across the path. After the optimal path is found, additional paths are searched for using Yen's algorithm and a modified similarity measure on various subgraphs of the roadmap to find unique paths. All paths found represent initial guesses that are then input to a collocation corrections method with a CEP mesh refinement process to recover natural solutions. This approach was employed to recover geometrically distinct natural solutions between two Lyapunov orbits at $C_J = 3.15$ in the Earth-Moon CR3BP. These results for this application motivate further exploration of PRM in the CR3BP to more efficiently summarize the solution space, increase the likelihood of producing a nearby continuous transfer, and decrease the computational time required to find feasible solutions.

9 ACKNOWLEDGMENT

This research is being completed at the University of Colorado Boulder. It is supported by a NASA Space Technology Graduate Research Opportunity.

REFERENCES

[1] P. Svestka and M. H. Overmars, *Robot Motion Planning and Control*. Berlin: Springer, Berlin, Heidelberg, 1st ed., 1998.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 1st ed., 2006.

[3] K. Lynch, G. Kantor, H. Choset, L. Kavraki, S. A. Hutchinson, W. Burgard, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 1st ed., 2005.

[4] A. Das-Stuart, K. Howell, and D. Folta, "A Rapid Trajectory Design Strategy for Complex Environments Leveraging Attainable Regions and Low-Thrust Capabilities," Adelaide, Australia, 68th International Astronautical Congress, September 2017.

[5] E. Trumbauer and B. Villac, "Heuristic Search-Based Framework for Onboard Trajectory Redesign," *Journal of Guidance, Control and Dynamics*, Vol. 37, No. 1, January-February 2014, 10.2514/1.61236.

[6] J. A. Starek, E. Schmerling, G. D. Maher, B. W. Barbee, and M. Pavone, "Real-Time, Propellant-Optimized Spacecraft Motion Planning under Clohessy-Whilshire-Hill Dynamics," Big Sky, MT, IEEE Aerospace Conference, March 2016.

[7] K. Bruchko and N. Bosanac, "A Preliminary Exploration of Path Planning for Initial Guess Construction in Multi-Body Systems," AAS/AIAA Astrodynamics Specialist Conference, August 2021.

[8] K. Bruchko and N. Bosanac, "Designing Spatial Transfers in Multi-Body Systems Using Roadmap Generation," Charlotte, NC, AAS/AIAA Astrodynamics Specialist Conference, August 2022.

[9] R. Anderson, M. Lo, and G. Born, "Application of Local Lyapunov Exponents to Maneuver Design and Navigation in the Three-Body Problem," Big Sky, MT, AAS/AIAA Astrodynamics Specialist Conference, August 2003.

[10] J. Y. Yen., "Finding the K shortest loopless paths in a network.," *Management Science*, Vol. 17, July 1971.

[11] V. Szebehely, *Theory of Orbits: The Restricted Problem of Three Bodies*. New York, NY: Academic Press, 1967.

[12] W. S. Koon, M. W. Lo, J. E. Marsden, and S. D. Ross, *Dynamical Systems, The Three-Body Problem and Space Mission Design*. Marsden Books, 3rd ed., 2006.

[13] R. Pritchett, *Strategies for Low-Thrust Transfer Design Based on Direct Collocation Techniques*. PhD thesis, Purdue University, IN, 2020.

[14] D. J. Grebow and T. A. Pavlak, "MCOLL: MONTE Collocation Trajectory Design Tool," Stevenson, WA, AAS/AIAA Astrodynamics Specialist Conference, August 2017.

[15] J.-H. Park and T.-W. Yoon, "Maximizing the Coverage of Roadmap Graph for Optimal Motion Planning," *Hindawi*, Vol. 2018, November 2018, 10.1155/2018/9104720.

[16] B. Park and W. K. Chung, "Adaptive Node Sampling Method for Probabilistic Roadmap Planners," St. Louis, USA, IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2009.

[17] S. T. Marco Morales and N. M. Amato, "Incremental Map Generation," Advanced Robotics, January 2006.

[18] B. Y. Huiping Liu, Cheqing Jin and A. Zhou, "Finding Top-k Shortest Paths with Diversity," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 30, No. 3, March 2018, 10.1109/TKDE.2017.2773492.

[19] S. Bique, J. Hershberger, V. Polishchuk, B. Speckii, S. Suri, T. Talvitie, K. Verbeek, and H. Yıldız, "Geometric k Shortest Paths," *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, Vol. 2015, 10 2015, pp. 1616–1625, 10.1137/1.9781611973730.107.

[20] J. Han, M. Kamber, and J. Pei, *Data Mining Concepts and Techniques*. Waltham, MA: Morgan Kaufmann Publishers, 3 ed., 2012.

[21] L. E. Kavraki and S. M. LaValle, *Motion Planning*. Springer Handbook of Robotics. Springer, Berlin, Heidelberg, 2008, 10.1007/978-3-540-30301-5_6.

[22] Y. Huang and K. Gupta, "A Delaunay Triangulation Based Node Connection Strategy for Probabilistic Roadmap Planners," New Orleans, LA, IEEE International Conference on Robotics and Automation, April 2004.

[23] W. C. T. Gene Eu Jan, Chi-Chia Sun and T.-H. Lin, "An O(nlogn) Shortest Path Algorithm Based on Delaunay Triangulation," *IEEE/ASME Transactions on Mechatronics*, Vol. 19, No. 2, April 2014, 10.1109/TMECH.2013.2252076.

[24] J. D. Marble and K. E. Bekris, "Asymptotically Near Optimal Planning with Probabilistic Roadmap Spanners," *IEEE Transactions on Robotics*, Vol. 29, No. 2, 2013, pp. 432 – 444, 10.1109/TRO.2012.2234312.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2 ed., 2001.

[26] E. Canalias and J. J. Masdemont, "Homoclinic and Heteroclinic Transfer Trajectories between Lyapunov Orbits in the Sun-Earth and Earth-Moon Systems," 10 2007.