MULTI-DIRECTIONAL RAPIDLY EXPLORING RANDOM GRAPH (MRRG) FOR MOTION PLANNING

A Thesis

by

SHUVRA KANTI NATH

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Nancy M. Amato
Committee Members,	Dezhen Song
	Suman Chakravorty
Head of Department,	Duncan M. Walker

December 2013

Major Subject: Computer Science

ABSTRACT

The motion planning problem in robotics is to find a valid sequence of motions taking some movable object from a start configuration to a goal configuration in an environment. Sampling-based path planners are very popular for high-dimensional motion planning in complex environments. These planners build a graph (roadmap) by generating robot configurations (vertices), and connecting nearby pairs of configurations according to their transition feasibility. Tree-based sampling-based planners (e.g., Rapidly-Exploring Random Tree, or RRT) start growing a tree outward from an initial configuration of the robot.

In this work, we propose a multi-directional Rapidly-Exploring Random Graph (mRRG) for robotic motion planning, a variant of the Rapidly-Exploring Random Graph (RRG). Instead of expanding a vertex in the tree in a single random direction during each iteration, mRRG expands in m random directions. Our results show that growing in multiple directions in this way produces roadmaps with more topologically distinct paths than previous methods. In an environment with dynamic obstacles, moving or new obstacles may invalidate a path from the start to the goal. Hence, roadmaps containing alternative pathways can be beneficial as they may avoid recalculation of new valid paths.

One of the important phases in sampling-based methods involves finding candidate nearest neighbors to attempt to connect to a node. Generally, the entire graph is considered to search for the nearest neighbors. In this thesis, we propose a heuristic method for finding nearest neighbors based on the hop limit, i.e., the maximum number of edges allowed in the path from a vertex to its neighbor. The candidate nearest neighbors are found by considering only those vertices within the hop limit. We experimentally show that our hop limit neighbor finder significantly reduces neighbor searching time over the standard brute force approach when constructing roadmaps.

To my Family and Friends

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Nancy M. Amato, for her guidance and support. She has helped me a lot to grow as a researcher; without her guidance and continuous encouragement, this work would not be possible.

I would like to thank my committee members Dr. Dezhen Song and Dr. Suman Chakravorty for their cooperation, feedback and support.

I would like to thank members of the Algorithms and Applications group in the Parasol Lab. I have truly enjoyed working with them. I am really grateful to Dr. Shawna Thomas for her continuous feedback, help and guidance. I would also like to thank Jory Denny, Chinwe Ekenna, Hsin-Yi (Cindy) Yeh and Aditya Mahadevan for their help and patience to answer all my questions.

Finally, I must thank my wife Rimi for her constant mental support and love for me. I am really honored to have her in my life. I would also like to thank my parents, Narayan and Karuna, who taught me to do hard work and provided support and encouragement throughout my studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In robotics, the motion planning problem involves finding a valid trajectory for a movable object from a start configuration to a goal configuration in a given environment. Usually, an environment consists of a robot and a set of obstacles. Motion planning has a wide range of applications in a variety of domains including robotics [24], gaming [29], computational biology [43, 45], virtual prototyping [8], computer animation [21] and computer aided design [8, 29].

The motion planning problem has been actively studied for decades and is known to be PSPACE-hard [38]. Several approaches have been proposed and recent attention has focused on randomized motion planning methods. One popular class of algorithms, sampling-based motion planning [20, 25], has been successful in solving problems with high dimensionality. Graph-based approaches explore the configuration space [30], the set of all possible robot configurations within the environment (valid or not). These planners build a graph data structure, called a roadmap, where valid configurations are encoded as vertices and edges represent the connectivity among them.

Tree-based sampling-based techniques [25][18] explore the space by growing one or more trees outward from an initial configuration in the configuration space. Incremental exploration of the space makes tree-based planners well-suited for problems with differential constraints. Tree-based approaches are largely used for single query planning. Current tree-based methods are mostly aimed at getting a single path from a start configuration to a goal configuration. However, in some situations, multiple paths from the start to the goal can be more desirable. For example, in an environment with dynamic obstacles, a moving or new obstacle may invalidate the current path from the start to the goal and an alternative path would need to be selected. We propose a motion planner, *multi-directional Rapidly*

Exploring Random Graph (mRRG), which can generate multiple topologically distinct pathways from the start to the goal. Informally, two paths are topologically distinct if they pass by different obstacles. For example, in Figure 1, all the three paths shown from the start to the goal are topologically distinct from each other. These paths can be used as alternative pathways instead of recalculating new ones if one becomes invalidated for some reason.

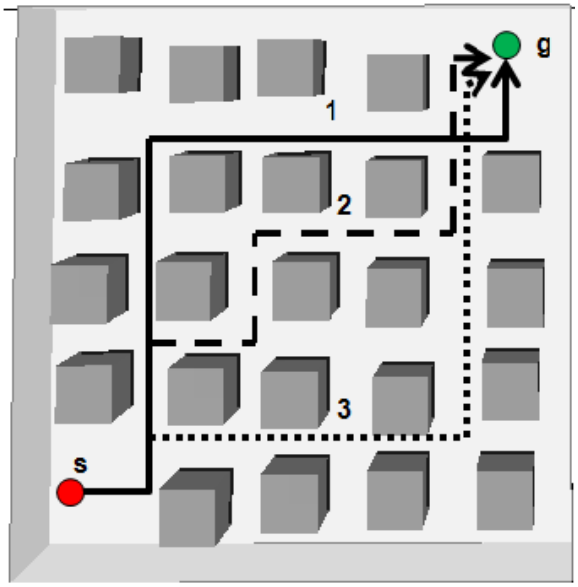


Figure 1: Topologically distinct pathways from s to g.

1.1 Our Approach

mRRG is an extension of Rapidly Exploring Random Graphs (RRGs) [19]. RRG is a tree-based motion planning method which builds a graph incrementally by expanding a configuration in a single random direction. After a successful expansion, connections are attempted between the new configuration and existing vertices in the graph, either those identified within a ball of radius r or as the k-nearest neighbors to the new configuration.

Our mRRG algorithm is different from traditional RRG in that it expands in m different random directions instead of in a single random direction. Expansion in multiple directions allows a single iteration to explore more in the locality of the vertex. This yields roadmaps containing more topologically distinct paths. mRRG is a generalization of previous approaches including RRG (where $m = 1$) and RRT (Rapidly Exploring Random Tree) [25] (where $m = 1$ and $k = 0$). We initially explored this idea of expanding in multiple directions for protein folding applications [35]. We obtained promising results for several small proteins and were better able to model the folding space than a standard RRT.

In RRT, RRG, and mRRG, we found that computing the neighbor configurations to which connections will be attempted consumes a significant portion of the computation time. To address this, we developed a heuristic method for finding nearest neighbors that instead of considering all vertices in the graph, restricts the candidate neighbors to vertices that are within h edges (hops) from the query vertex. We show that in many cases restricting the search to the locality of the query vertex significantly reduces running time without sacrificing much in result quality.

Our experiments show that mRRG helps to explore the configuration space better than RRT using multi-directional search. It also helps to create a larger number of topologically different paths from a start to a goal. As more paths are created, shorter paths are also achieved. We also examined the effect of using the hop limit heuristic during neighbor finding and found that the hop limit approximation reduces the neighborhood searching time significantly.

Our Contribution. The main contributions of this work are:

- An extension of RRG, that can create multiple topologically distinct pathways.
- A new hop limit neighbor finder to reduce the computation time spent in selecting

candidate vertices for connection.

- Approximation of energy landscape of protein using mRRG.

Some of the protein folding results in this thesis were published in [35]. Another manuscript is currently under preparation that will include results from the robotics applications.

1.2 Organization of Thesis

The thesis is outlined as follows. In section 2 and section 3, we give an overview of related work and basics for robotics and protein folding. We then present the mRRG algorithm and our hop limit neighbor finder in section 4. Section 5 provides experimental results showing improved performance of mRRG and our hop limit neighbor finder over prior approaches. Finally, we conclude in section 6.

2. PRELIMINARIES AND RELATED WORK FOR ROBOTIC MOTION PLANNING

2.1 Configuration Space

In motion planning, the robot is a movable object whose specification (position and orientation) can be defined by a set of degrees of freedom (DOF) of size n, one for each parameter used to specify the object position. A configuration of a robot can be uniquely described as a point in an n-dimensional space, called *configuration space (C-space)* [30]. Configuration space is the space containing all possible configurations (feasible or not) of the robot. The number of degrees of freedom of a robot is the dimension of C-space. The subset of all feasible configurations (i.e., which do not collide with obstacles) is called free C-space (C_{free}), and the subset of unfeasible ones is called *blocked* C_{space} (C_{obs}). With this notation, the motion planning problem is to find a valid path in C_{free} from the start to the goal configuration.

2.2 Tree-based Planners

Sampling-based approaches have been successfully used in solving various motion planning problems. Both graph-based approaches and tree-based approaches have demonstrated their effectiveness for solving motion planning problems in high dimensional space.

Tree-based planners build a tree incrementally from an initial configuration outwards towards unexplored regions of space. One of the first such methods, Ariadne's Clew [10, 31], grows a tree by alternating between two different phases: "explore" and "search". In the "explore" phase, the algorithm generates a random configuration and attempts to expand the tree as far as possible towards it. In the "search" phase, the algorithm attempts to expand the tree towards the goal. While "explore" aims at building the representation of the space, "search" looks for the target.

The Rapidly Exploring Random Tree (RRT) [25], shown in Algorithm 1, is one of the most popular types of tree-based methods. It biases tree growth to unexplored regions of the space by iteratively generating a random sample (q_{rand}) and expanding the nearest configuration (q_{near}) in the tree toward that sample. Typically, iterations are made until the goal configuration is reached. RRTs are well-suited for motion planning problems with obstacles and differential constraints- non-holonomic or kinodynamic.

There are many variants of the basic RRT algorithm. RRT-Connect [23] builds two trees, one rooted at the start and one rooted at the goal, until the two trees can be connected [23]. RRT-Connect combines RRT with a aggressive greedy heuristic to connect the trees. Rapidly Exploring Random Graph (RRG) [19] builds a graph instead of a tree by attempting additional connections from the expanded node at each expansion step. Usually a radius-based or k-nearest connection method is used. In the radius-based version, connections are attempted to all the configurations contained within a ball of a given radius r from the new vertex. In the k-nearest version ([19]), connections are sought to the k nearest neighbors where $k = k_{RRG} log(n_V))$. n_V denotes the number of vertices in the graph and $k_{RRG} > k^*_{RRG} = e(1 + 1/d)$ where d is the dimensionality of the configuration space.

RRT* is a variant of RRG that maintains the asymptotic optimality of the tree structure [19]. It removes the "redundant" edges (edges which are not part of the shortest path from the root) of RRG, ensuring minimum cost paths to reachable vertices.

Expansive-Space Trees (EST) [18] expands nodes based on the density of C-space. Nodes in sparsely sampled areas are more likely to be chosen for expansion. The selected node is expanded in multiple directions such that the directions cover the sparse areas of C-space. Sampling-based Roadmap of Trees (SRT) [37] integrates Probabilistic Roadmap Methods (PRMs) [20], a graph-based sampling method, with RRTs by building a roadmap, or graph, of trees that combines the global sampling properties of PRM and the local

sampling properties of RRT.

Algorithm 1 Rapidly Exploring Random Tree

Input. An initial placement q_{init}, step distance δ, and an evaluator E.

Output. A graph G rooted at q_{init} that satisfies E.

1: $G.ADD\_VERTEX(q_{init})$.
2: **while** G does not satisfy E **do**
3: Let q_{rand} be a random sample, valid or not.
4: Let q_{near} be the nearest sample $\in G$ to q_{rand}
5: $q_{new} = STEER(q_{near}, q_{rand}, \delta)$
6: $G.ADD\_VERTEX(q_{new})$
7: $G.ADD\_EDGE(q_{near}, q_{new})$
8: **end while**

2.3 Nearest Neighbor Finding Methods

Nearest neighbor finding is one of the most important operations in motion planning. There have been a number of methods, both exact and approximate, for nearest neighbor searching.

The k-closest method is a common exact neighbor finding strategy that finds the k nearest neighbors from the query vertex, where k is typically a small, fixed constant. Using a brute force approach to compute the k-nearest neighbors, the running time per node is usually $O(kn)$, totalling $O(kn^2)$ overall. Another similar approach is the r-closest method which calculates all neighbors within a radius r of the query vertex using some distance metric. The size of the neighbor set is dependent on the sampling density.

In [32], a randomized variant of these methods is proposed, which first selects closest

k_1 nodes and returns k_2 of them at random. A small amount of randomness was shown to produce more edges and better connected roadmap compared to the k-closest method.

2.3.1 Data Structures for Approximate Neighbor Finding

Another direction in nearest neighbor finding is to use different data structures to provide approximate and/or more efficient solutions. For higher dimensions, one suitable data structure is a KD-tree [9] which recursively partitions the data set. Each node in the KD-tree denotes a plane through one of the dimensions partitioning the set of points into left/right (up/down) sets. Each child set contains half the elements of the parent node set. Children are partitioned using different dimensions recursively. Partitioning stops when each point is in just one cell. So, after KD-tree construction, each cell contains a small portion of the input data. A KD-tree can be constructed in $O(knlogn)$ time by sorting n points in k dimensions independently.

Nearest neighbor search using KD-trees can eliminate half the points with a simple test. The tree is recursively searched starting from a root node until a minimum region containing the target vertex is found. Each parent node is checked to see if there are other regions to contain a point that is closer. Searching is terminated when the algorithm decides that there is no chance of finding a closer point. Finding the nearest neighbor is an $O(logn)$ operation using a KD-tree.

Different software libraries provide implementation of KD-trees. CGAL [1] finds an approximate nearest neighbor by constructing a single KD-tree [5] structure in the configuration space of the robot. CGAL approximates nearest neighbors by changing proximity to the query vertex. An approximation parameter ϵ can be used , where a k-nearest-neighbor search returns k neighbors that are guaranteed to be no more than $(1 + \epsilon)$ times farther away than the exact k-th neighbor.

The MPNN [6] library finds an approximate nearest neighbor by constructing multiple

KD-trees in the configuration space. MPNN extends the ANN [4] algorithm, developed by Arya and Mount for Euclidean spaces, to handle topologies arising in motion planning. Building multiple trees is particularly helpful as trees can grow from multiple difficult areas. However, additional trees also create complicated decision problems in selecting nearest neighbors.

The Metric Tree [46] organizes the data set in a spatial hierarchical manner. The tree is constructed by recursively subdividing the group of nodes based on the largest distance. The search in this tree is done by a guided depth-first search (a node is not explored if it is far away from the query vertex).

3. PRELIMINARIES AND RELATED WORK FOR MODELING PROTEIN MOTION

Before describing our mRRG method, we present some basics related to the protein model, energy function and energy landscape. We also discuss related work on protein folding, both experimental and computational, regarding the features and quality of solutions of different methods.

3.1 Protein Model

Proteins are biological molecules that are made up of a sequence of amino acids. A protein's structure can be represented at three different levels: primary structure, secondary structure (Figure 2) and tertiary structure [11, 40]. The amino acid sequence of a protein is referred to as primary structure. The secondary structure of a protein consists of regular sub-structures which link together to form the three-dimensional tertiary structure (native state). The major secondary structures are alpha helices and beta strands.

Each amino acid has two major flexible bond angles, ϕ and ψ, which determine a protein's flexibility. The ϕ angle represents the rotation along the $N - C_\alpha$ bond, and the ψ angle represents the rotation along the $C_\alpha - C$ bond of an amino acid. For each amino acid of the protein, we model the ϕ and ψ backbone torsional angles as flexible and keep all other bond lengths and angles fixed. This is a standard modeling assumption [44].

The protein's degrees of freedom (DOF) can be expressed in terms of a set of ϕ and ψ angles. If N is the number of amino acids of a protein, its DOF will be $2N$ (number of ϕ and ψ angles). Thus, the protein is modeled as an articulated linkage robot with joints ranging between $[0, 2\pi)$.

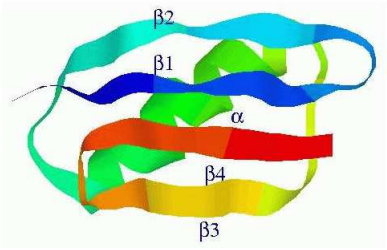


Figure 2: The native state of protein G. It consists of a central alpha helix and a four strand beta sheet. This figure is taken from [42].

3.2 Potential Functions

The validity of a configuration is determined by the energy/potential value of that configuration. Atoms inside a protein interact with the surrounding solvent and each other. These interactions, which cause a protein to fold or unfold, can be represented by potential functions. A general form of the potential function can be expressed as a summation of potentials associated with bond length, bond angle, dihedral angle, electrostatic interactions [26] and van der Waals potentials.

In this work, we consider a coarse potential function as described in [3]. Side chains are modelled as spheres with the same radii. If the side chain spheres are too close (less than 2.4Å during sampling and 1.0Å during connection), the conformation is rejected to prevent clash of atoms. Otherwise, the energy is calculated as:

$$U_{tot} = \sum_{constraints} K_d\{[(d_i - d_0)^2 + d_c^2]^{1/2} - d_c\} + E_{hp}$$

where K_d is 100 kJ/mol, d_i is the length on the ith constraint, and $d_0 = d_c = 2$Å as shown in [26].

3.3 Energy Landscape

The energy landscape of a protein represents all the protein configurations and their associated energies. In Figure 3, the xy plane gives simplified representation of ϕ and ψ angles of amino acids and the z axis represents associated potential/energy. It is usually thought that native state of a protein is at the bottom of the funnel like energy landscape.

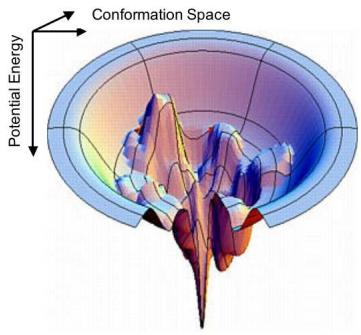


Figure 3: Visualization of the protein's energy landscape [13]. The xy-plane represents the protein's configuration space, and the z-axis is the potential energy.

3.4 Experimental Methods

Numerous experimental methods have been applied to study protein motion including circular dichroism [39], fluorescence experiments [39], hydrogen exchange [47], pulse labeling, and NMR spectroscopy [33]. Circular dichroism (CD spectra) use UV light to inspect absorption of polarized light by protein configurations. Fluorescence experiments measure change in fluorescence as a function of denaturant. Hydrogen exchange mass spectrometry and pulse labeling experiments are used to identify most exposed/protected parts of protein structures. NMR spectroscopy is another useful tool to study side-chain motion and backbone motion.

3.5 Computational Methods

Experimental methods are not only complex and costly, but it is also hard to inspect the fast moving folding process using these methods. Thus, computational simulation techniques are needed to study these process by providing a realistic model of the folding process. Table 3.1 provides a summary of different computational methods, their computational requirements, solution quality, etc.

Traditional simulation methods such as molecular dynamics [26, 15, 16], Monte Carlo methods [14, 22], and simulated annealing [27] can provide a single, detailed, high-quality folding pathway but at a high computational expense. Statistical mechanical models [34, 2, 7] provide statistics about the global energy landscape but cannot provide individual pathways. Lattice models [12] are theoretical models and not used on real proteins in practice. Robotic motion planning methods such as the Probabilistic Roadmap Method (PRM) [20] and rapidly Exploring Random Tree (RRT) [25] have been adapted to model the folding process. These robotics-based methods are quite promising as they can produce multiple folding pathways using a short amount of time (few hours). This helps to study both individual folding trajectories and global properties of the protein energy landscape.

Approach	Landscape Properties	# Paths Produced	Path Quality	Computational Time Required	Native Needed
Trajectory-Based	Poor Coverage	1	Good	Long	No
Statistical Mechanical Model	Good Coverage	0	N/A	Short	Yes
Robotics-Based					
PRM	Good Coverage	Many	Approx	Short	Yes
T-RRT	Poor Coverage	Many	Approx	Very Short	Yes
Lattice Model	*Not used on real proteins*				

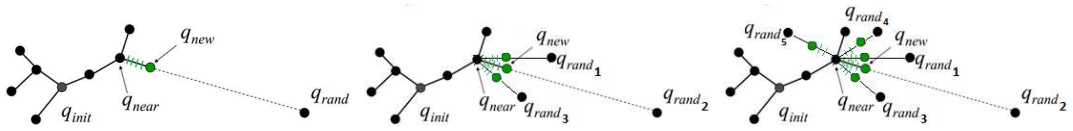
Table 3.1: Comparison of computation models for modeling protein motion.

4. MULTI-DIRECTIONAL RAPIDLY EXPLORING RANDOM GRAPH mRRG

We begin this chapter by introducing the mRRG method that explores the configuration space in multiple directions. Then we discuss the hop limit neighbor finder which can be used to reduce nearest neighbor searching time.

4.1 mRRG

The multi-directional Rapidly Exploring Random Graph (mRRG) extends the RRG algorithm by steering the parent vertex in multiple dispersed directions at each expansion step instead of with a single bias towards q_{rand}. Expansion in multiple directions allows the planner to search more densely near the locality of q_{near}. Figure 4 compares RRT with mRRG growth with m values 3 and 5.



(a) RRT(m=1) (b) m=3 (c) m=5

Figure 4: An overview of mRRG with different m values. Instead of a single q_{rand} for expansion, m random samples are generated to guide expansion from q_{near}. Along each direction q_{rand_i}, a q_{new} is identified (shown in green) and connected to its nearest neighbors.

In each iteration of mRRG (see Algorithm 2), a random configuration q_{rand} is generated. The nearest vertex q_{near} to q_{rand} is selected for expansion, and q_{near} is steered towards q_{rand} to get q_{new}. A connection is attempted from q_{new} to q_{near} and added to the graph if valid. If the graph type is not required to be a tree (a connected graph without cycles), CONN_NEIGHBORS (see Algorithm 4) is then called with the query configuration

q_{near} to select a set of candidate neighbors. For each candidate, a local planning method is used to evaluate if there is a feasible trajectory connecting them. An edge is added if it is feasible.

Unlike traditional RRG, $m-1$ additional directions (see Algorithm 3) are then selected towards which to expand q_{near}. Just as before, multiple connections are attempted from each expanded vertex. The directions are chosen such that they are dispersed, i.e., not near, from already expanded directions. Multiple random directions are chosen and the direction with the most dispersion is chosen. The mRRG graph incrementally grows until a set of evaluation criteria is met.

Lemma 1. *mRRG with $m = 1$ is the same as RRT and RRG when all other aspects (connection method, local planner, distance metric, etc.) are same.*

Proof. When $m = 1$, mRRG performs all the operations in Algorithm 2 except line 12-22. RRG algorithm also performs the same operations. It uses k-nearest or radius based connection method. So, mRRG ($m = 1$) with all other aspects the same as RRG will produce the same graph. If the graph type is a tree and $m = 1$, then mRRG performs all the operations in Algorithm 2 except line 12-22. RRT also perform the same operations. So, RRT and mRRG will produce the same tree if all aspects are same. □

4.1.1 mRRG Complexity

The mRRG algorithm consists of the following basic operations: sampling a point in C-space and determining feasibility of a configurations. If the C-space is d dimensional, choosing a random point (line 3, 13) in C-space will take $O(d)$ time. Let the time to check the feasibility of a sample be $O(f)$ time. The feasibility checking depends on the dimensionality of the robot and the geometric model of the robot and obstacles in the environment. Let us assume that nearest vertex calculation takes $O(t_{nf})$ time (depends on neighbor finder). Let the average number of feasibility checks for edge connection be

Algorithm 2 mRRG

Input. An initial configuration q_{init}, a path step Δq, a number of expansion directions m, a neighbor finder nf, a local planner lp, and an evaluator E

Output. A graph G rooted at q_{init} that satisfies E

1: $G.V = q_{init}$
2: **while** G does not satisfy E **do**
3: q_{rand} = RAND_CONF()
4: q_{near} = NEAREST_VERTEX(q_{rand}, G)
5: q_{new} = STEER($q_{near}, q_{rand}, \Delta q$)
6: **if** lp.IS_CONNECTABLE(q_{near}, q_{new}) **then**
7: G.ADD_VERTEX(q_{new})
8: G.ADD_EDGE(q_{near}, q_{new})
9: **if** $G.type \neq$tree **then**
10: CONN_NEIGHBORS(q_{new}, nf, lp, G)
11: **end if**
12: **for** $i = 2 \ldots m$ **do**
13: q_{disp} = SELECT_DIRECTION(q_{near}, G)
14: q_{new} = STEER($q_{near}, q_{disp}, \Delta q$)
15: **if** lp.IS_CONNECTABLE(q_{near}, q_{new}) **then**
16: G.ADD_VERTEX(q_{new})
17: G.ADD_EDGE(q_{near}, q_{new})
18: **if** $G.type \neq$ tree **then**
19: CONN_NEIGHBORS(q_{new}, nf, lp, G)
20: **end if**
21: **end if**
22: **end for**
23: **end if**
24: **end while**
25: **return** G

$O(l)$. Let the number of nearest neighbors be k. Let the maximum number of trials in Algorithm 3 be c. Let n_E be the number of edges in graph G.

In the $STEER$ function in line 5 and $IS\_CONNECTABLE$ function in line 6, feasibility is checked for intermediate nodes between two nodes at a fixed resolution. The expected cost of total feasibility checks is $O(lf)$. $CONN\_NEIGHBORS$ is called in line 10. Line 1 of Algorithm 4 takes $O(t_{nf})$ time. Line 2-6 of Algorithm 4 takes $O(klf)$

Algorithm 3 SELECT_DIRECTION

Input. A vertex q and a graph G

1: Let $maxTrial$ be a data member denoting number of trials to make
2: Let MAX_ANGLE denotes the maximum angle
3: **for** $i = 1 \ldots maxTrial$ **do**
4: Let MIN_ANGLE denote the minimum angle
5: q_{rand} = RAND_CONF()
6: **for** each v adjacent to q in G **do**
7: $angle$ =CALCULATE_ANGLE($\overrightarrow{qv}$, $\overrightarrow{qq_{rand}}$)
8: **if** $MIN\_ANGLE > angle$ **then**
9: $MIN\_ANGLE = angle$
10: **end if**
11: **end for**
12: **if** $MAX\_ANGLE < MIN\_ANGLE$ **then**
13: $MAX\_ANGLE = MIN\_ANGLE$
14: $q_{best} = q_{rand}$
15: **end if**
16: **end for**
17: **return** q_{best}

time. Algorithm 3 takes $O(cE))$ time.

The operations in line 3-11 of Algorithm 2 are repeated for $m - 1$ times in line 12-22.

If the algorithm runs for I number of iterations, total running time becomes $O(Im(lf + t_{nf} + klf + d + cE))$.

4.2 Hop Limit Neighbor Finder

The cost of finding the nearest neighbors of a given configuration is one of the most expensive operations for sampling based motion planning. As described in Section 2.3, the standard approach is a brute force method that computes the distances from the configuration in question to every vertex in the graph and sorts them accordingly, see Algorithm 5. It takes $O(kn)$ time where n is the number of vertices in the graph and k is the number of neighbors to find.

We propose a heuristic method we call *hop limit*. Instead of considering the entire

Algorithm 4 CONN_NEIGHBORS

Input. A connecting vertex q, a neighbor finder nf, a local planner lp and a graph G

```
1: N = nf.FIND_NEIGHBORS(q, G)
2: for each n ≠ q ∈ N do
3:    if lp.IS_CONNECTABLE(q, n) then
4:       G.ADD_EDGE(q, n)
5:    end if
6: end for
```

Algorithm 5 BruteForce.FIND_NEIGHBORS

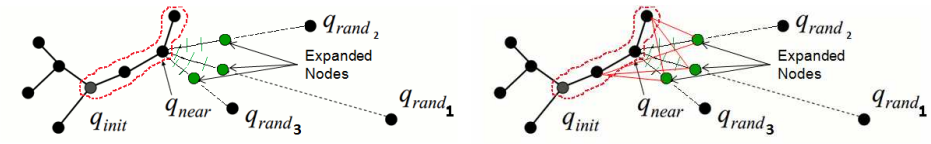
Input. A connecting vertex q and a graph G

Output. A set of neighboring vertices

```
1: Let k be a data member denoting the number of neighbors to find
2: X = ∅ with maximum allowed size k
3: for each v ∈ G.V do
4:    D=PAIR(v, distance(q, v))
5:    Insert D into X in sorted order
6: end for
7: return X
```

graph in the nearest neighbor search, we only examine those vertices within h edges (hops) of the parent node of query configuration. By only looking at a subgraph of the original graph, the candidate set for distance calculations is dramatically reduced while still preserving locality. Other neighbor finders, either exact or approximate, can be used in combination to search the neighbors from this reduced set. Figure 5(a) shows the subgraph G' that is within $h = 2$ hops of r outlined in dash. Then, for each of the expanded configurations, connections are attempted to the k nearest neighbors in the subgraph. Here, the brute force neighbor finder is called to identify the k nearest neighbors from G'. Figure 5(b) displays the connection attempts when $k = 2$. Algorithm 6 sketches the approach.

The user has control over the size of the candidate set by changing the hop limit h. There are many ways to compute the subgraph induced by the hop limit. Here we use a



(a) (b)

Figure 5: Hop limit neighbor finding with $k = 2$ and $h = 2$. (a) The subgraph G' from q_{near} where $h = 2$ is outlined in dash. (b) The resulting $k = 2$ connections attempted are shown.

Algorithm 6 HopLimit.FIND_NEIGHBORS

Input. A connecting vertex q a graph G
Output. A set of neighboring vertices

1: Let h be a data member denoting the hop limit
2: Let nf be a member denoting another neighbor finder
3: r = PARENT_VERTEX(q, G)
4: $G' =$ SUBGRAPH_WITHIN_HOP_LIMIT(r, G, h)
5: **return** nf.FIND_NEIGHBORS(q, G')

simple breadth first search (BFS) with search depth equal to h. Calculating the parent in line 1 takes $O(1)$ time. In our case, $r = q_{near}$ is the parent of $q = q_{new}$ for mRRG.

Calculating the subgraph takes $O(b^h)$ time where b denotes the maximum branching factor (the out-degree) of the graph. As the h value is usually small and the average branching factor of the graph is small, this time is much smaller as compared to a BFS of the entire graph ($O(n_V + n_E)$ time, where n_V and n_E denote number of vertices and edges respectively). Computing the k-nearest neighbors (brute force) in this subgraph G' takes $O(kn_{V'})$ time where $n_{V'}$ is the number of vertices in the subgraph. Usually $n_{V'}$ is much less than n_V.

So, the overall running time of the hop limit neighbor finder with the brute force neighbor finder is $O(b^h + kn_{V'})$, which is much smaller than the $O(kn)$ running time of the brute force neighbor finder alone.

Notice that the same subgraph is induced during each iteration of the loop in line 12 of Algorithm 2.

5. EXPERIMENTAL ANALYSIS

In this chapter, we study the performance of mRRG under different input parameters and compare against RRT and RRG. We run experiments for both robots and proteins to study the effect of m and h. We study how m value effects on number of pathways. We show that growing in multiple directions help to find a path faster compared to RRT in different environments. We also study the computational cost and accuracy of hop limit neighbor finder and compare with brute force neighbor finder. We show that a hop limit value of 2 results in a large savings in terms of time without losing much accuracy.

5.1 Implementation and Platform

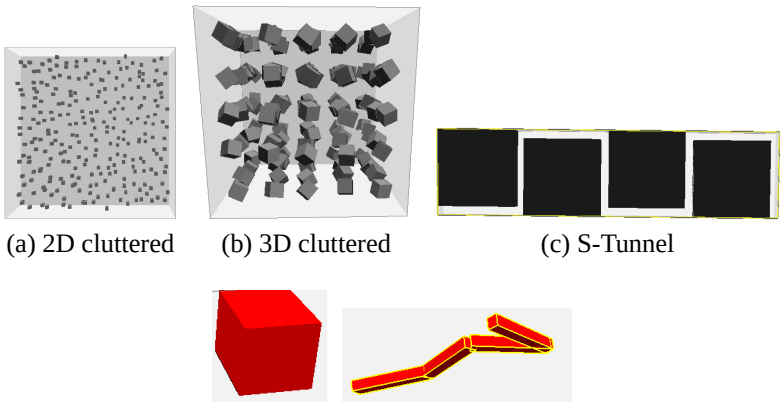
The algorithm was implemented and tested using the C++ motion planning library (PMPL) developed in the Parasol Lab at Texas A&M University and results are averaged over a series of 10 runs. We used the collision detection library PQP [17] for our experiments. For RRG, k is automatically tuned as in [19] (see Section 2.2).

5.2 Experimental Study for Robots

In this section, we describe the different environments and robot types that were used for the experiments with robots. These were selected to allow us to study the effect of m and hop limit performance.

5.2.1 Environments

- **2D cluttered (2DOF cube/6 DOF articulated linkage).** The 2D cluttered environment in Figure 6(a) has 256 randomly positioned obstacles ($0.2 \times 0.2 \times 0.2$ cube) in a bounding box that is 10×10.
- **3D cluttered (9 DOF articulated linkage).** The 3D cluttered environment in Figure 6(b) has 125 randomly positioned obstacles ($0.2 \times 0.2 \times 0.2$ cube) in a bound-



(a) 2D cluttered (b) 3D cluttered (c) S-Tunnel

(d) Rigid body cube (e) 4-link articulated

Figure 6: Environments and robots used for robot experiment.

ing box that is $10 \times 10 \times 10$. We consider two different robots: a $0.2 \times 0.2 \times 0.2$ cube (Figure 6(d)) and a 4-link articulated linkage (Figure 6(e)) where each link is $0.2 \times 0.03 \times 0.03$. The cube robot has 2 DOF (Planar Translational) and the articulated linkage robot has 6 DOF (Planar Rotational).

- **S-Tunnel (6DOF cube/9 DOF articulated linkage).** The S-Tunnel environment in Figure 6(c) has a long tunnel (width 1) created by 4 big obstacles in a bounding box that is $40 \times 1 \times 10$. We consider the same two different robots but in a 3D environment: a $0.5 \times 0.5 \times 0.5$ cube (Figure 6(d)) and a 4-link articulated linkage where each link is $0.2 \times 0.03 \times 0.03$ (Figure 6(e)). The cube robot has 6 DOF , and the articulated linkage robot has 9 DOF.

5.2.2 Study of m Value

We compare the effect of changing the m value on finding alternative pathways. Our goal is to get more topologically distinct pathways. We study a range of values of m that we determined resulted in qualitatively different pathways. These were determined for each environment/scenario separately.

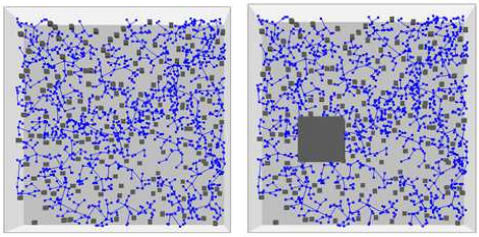
5.2.2.1 Experimental Setup and Evaluation Metrics

Below is the description of the experiments we conducted for RRT and mRRG:

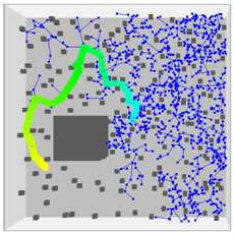
- **Roadmap Generation.** For RRT/mRRG, we first generate roadmap (Figure 7(a)) for a fixed amount of time.
- **Arrival of New Obstacle and Revalidation.** A new obstacle is added to the environment (Figure 7(b)). The roadmap is revalidated to remove the vertices and edges that are invalidated by the new obstacle. Next, all the connected components that do not contain the start are removed.
- **Continue Roadmap Generation.** After revalidation of the roadmap, RRT/mRRG starts growing (Figure 7(c)) the current roadmap.
- **Query for a Path.** A valid path from the start to the goal is searched. If not found, roadmap generation is continued. The metrics that are collected for the experiments are listed below.
- **Revalidation Time.** Time to remove newly invalid nodes/edges after the new obstacle is inserted.
- **Roadmap Generation Time (after new obstacle).** Time to generate the roadmap after newly invalid nodes/edges are removed.
- **Query Time.** Time to find a valid path from the start to the goal.
- **Total Time.** The sum of the time spent in roadmap generation (before new obstacle), revalidation time of roadmap, roadmap generation time (after new obstacle) and the query time.

5.2.2.2 Results for m Study

In our experiments, we used 2d-cluttered/3d-cluttered environments with a 2 DOF cube and a 4-link (6 DOF, 9DOF) robot. As described in Section 5.2.2.1, both methods (RRT,



(a) Initial roadmap (b) New obstacle



(c) Valid path

Figure 7: Experimental setup for m study.

mRRG) are first run for the same amount of time (see Table 5.1). Then the new obstacle ($2 \times 2 \times 2$) is inserted and the roadmap is revalidated. The roadmap is generated again until a valid path from the start to the goal is found.

For all the timing figures, we also show the standard deviation. In Figure 8, the value $m = 3$ helps to find a path faster compared to RRT and other m values of mRRG. The query time (see Figure 9), revalidation time (see Figure 10), generation time after new obstacle (see Figure 11) are also lowest for $m = 3$.

For $m = 5$, the roadmap was bushier and all the expanded directions did not disperse much from the start position to find a valid path. For $m = 2$, in most of the cases, all the valid paths became invalid after the new obstacle was inserted. In a few cases, there was still a valid path. That is why the standard deviation is very high for this value of m. For $m = 3$, in most cases there was still a valid path after the new obstacle was inserted into the environment.

Environment	robot	#Generation Time(s) before new obstacle
2D_cluttered	cube	20
2D_cluttered	4-link	60
3D_cluttered	4-link	40

Table 5.1: Roadmap generation time (before new obstacle) for different environments.

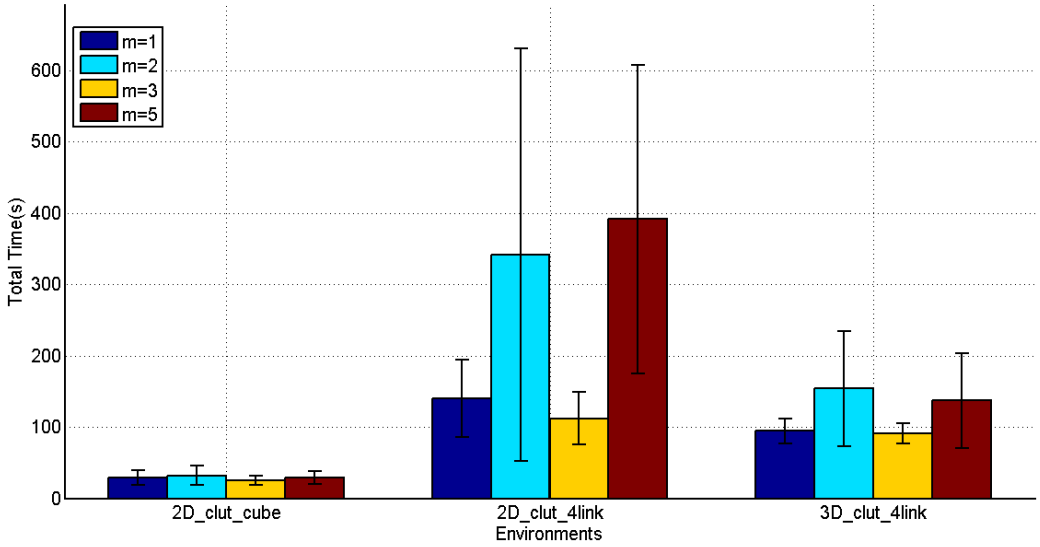


Figure 8: Comparison of total time needed for different m values.

Thus, overall, for the environments shown in the experiments, $m = 3$ is the best value of m based on these timing statistics.

5.2.3 Study of h Value

In this section, we compare the performance of our hop limit neighbor finder with standard brute force neighbor finding for the RRG algorithm.

5.2.3.1 Experimental Setup and Evaluation Metrics

Below is the description of experimental setup for RRT/mRRG:

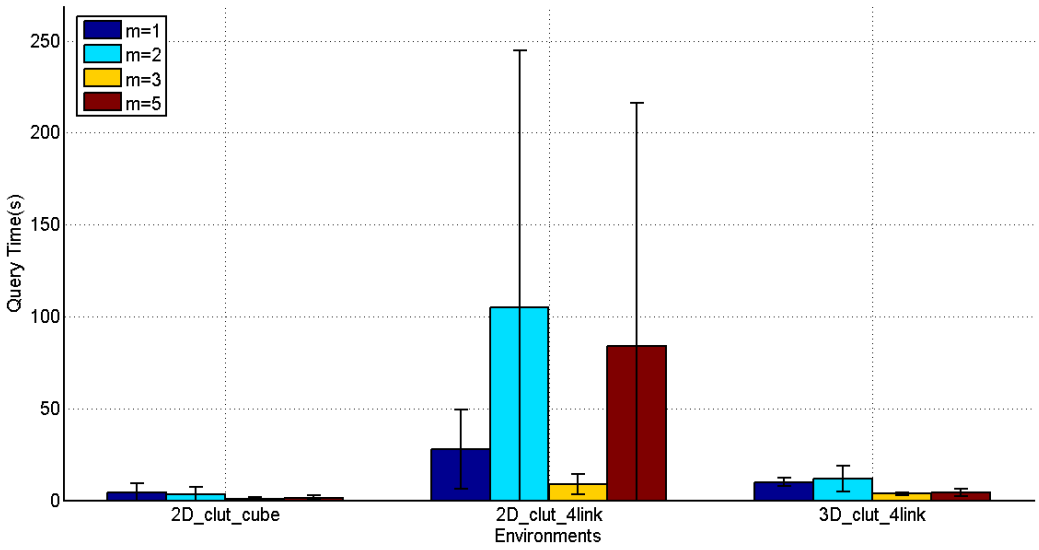


Figure 9: Comparison of query time needed for different m values.

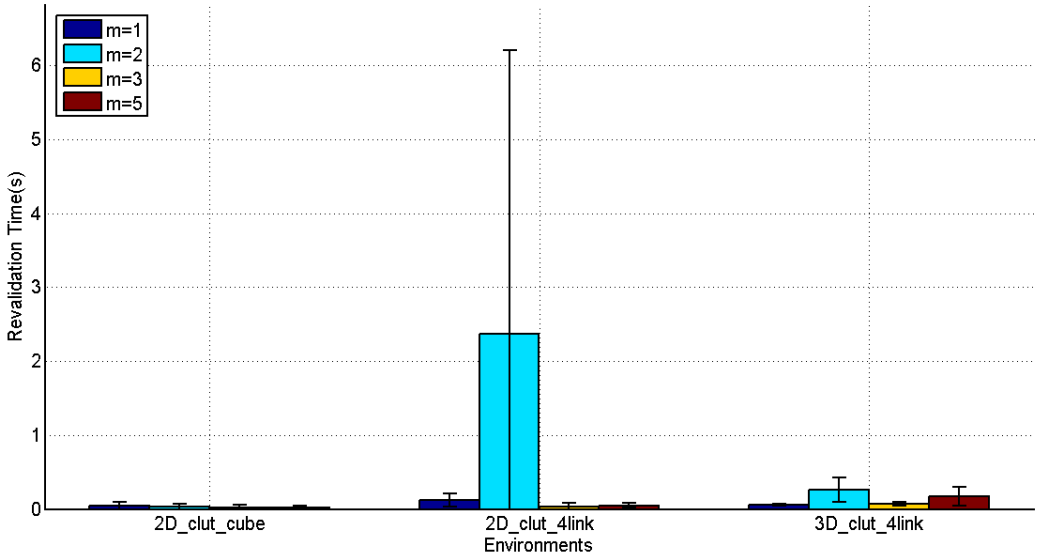


Figure 10: Comparison of revalidation time needed for different m values.

- **Roadmap Generation.** For RRT/mRRG, we first generate a roadmap with a fixed number of vertices and stop. In this experiment we use 200 vertices.

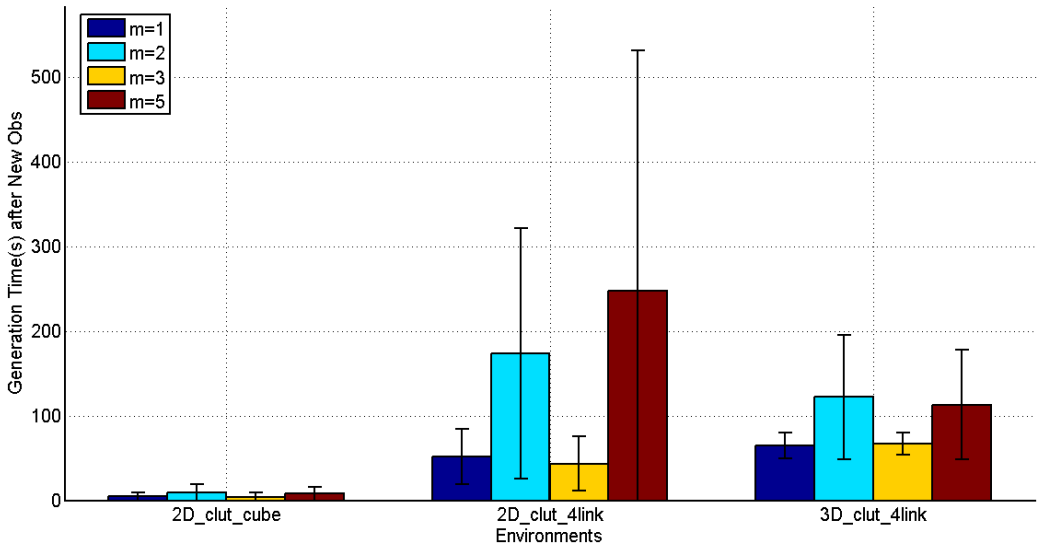


Figure 11: Comparison of generation Time (after new obstacle) for different m values.

- k **Value.** k is automatically tuned as in [19]. For the experiments, k ranges between 1 and 28.

The goal of our hop limit neighbor finder is to find neighbors with low computational cost with high accuracy compared to an exact neighbor finding method. To compare the quality of the solution, we used a variety of performance-based metrics to evaluate the: (a) computational cost and (b) accuracy of the methods. We also used some other metrics to understand the effect of the h value in neighbor finding. The metrics are listed below:

- **Neighbor Finding Time**: The total time spent to find neighbors for all the nodes. This metric tells us how costly the different neighbor finding methods are.
- **Same Edge (%)**: The percentage of the edges the hop limit method found as compared to the brute force neighbor finding method. This metric tells us how accurate our heuristic based method is compared to the exact neighbor finding method.
- **Local Planner Attempts**: The total number of connection attempts made by the local planner. This metric is correlated with how many neighbors could be found by different

methods.

- **Local Planner Success (%)**: The percentage of successful local planner connections. This metric indicates the effectiveness of each method at finding connectible neighbors.
- **Number of Edges**: The total number of roadmap edges. This metric tells us how many edges are produced by different methods.

5.2.3.2 Results for h Study

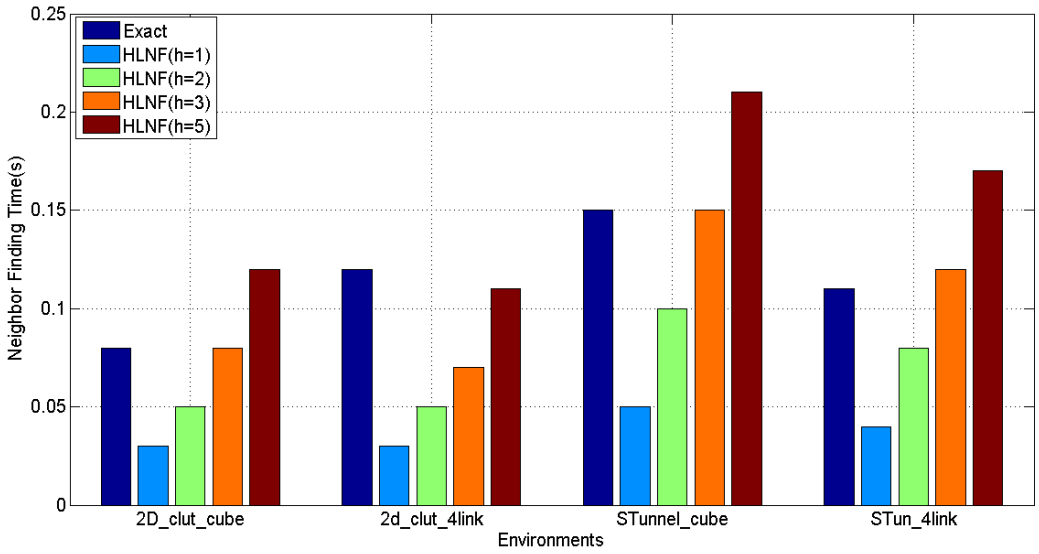


Figure 12: Comparison of neighbor finding time.

We compare the performance of different neighbor finders (brute force neighbor finder and hop limit neighbor finder with different hop values) in the 2D cluttered environment and S-Tunnel environment with the cube robot and 4-link articulated robot for RRG . As expected, the neighbor finding time is significantly reduced (see Figure 12) as the hop limit allows the finder to search for neighbors in a reduced candidate set. As the hop limit

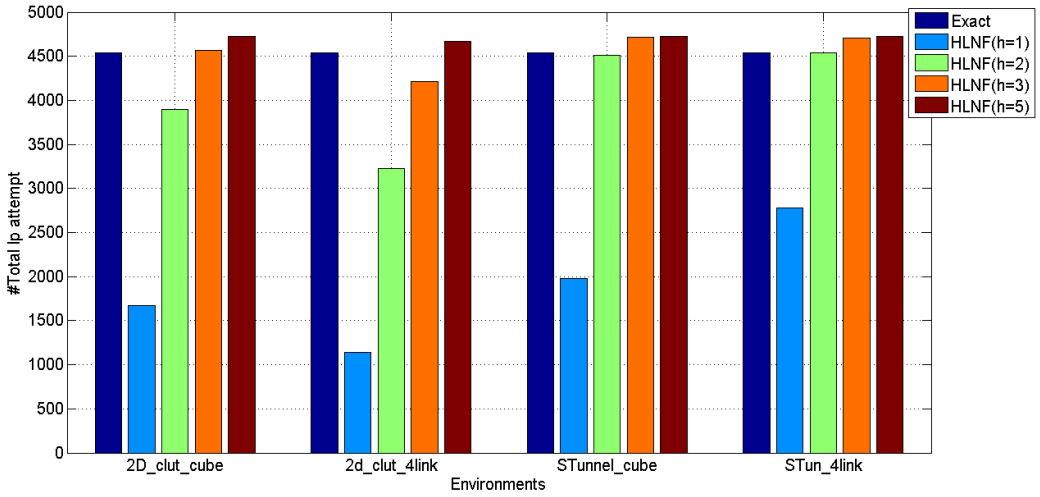


Figure 13: Comparison of #local planner attempts.

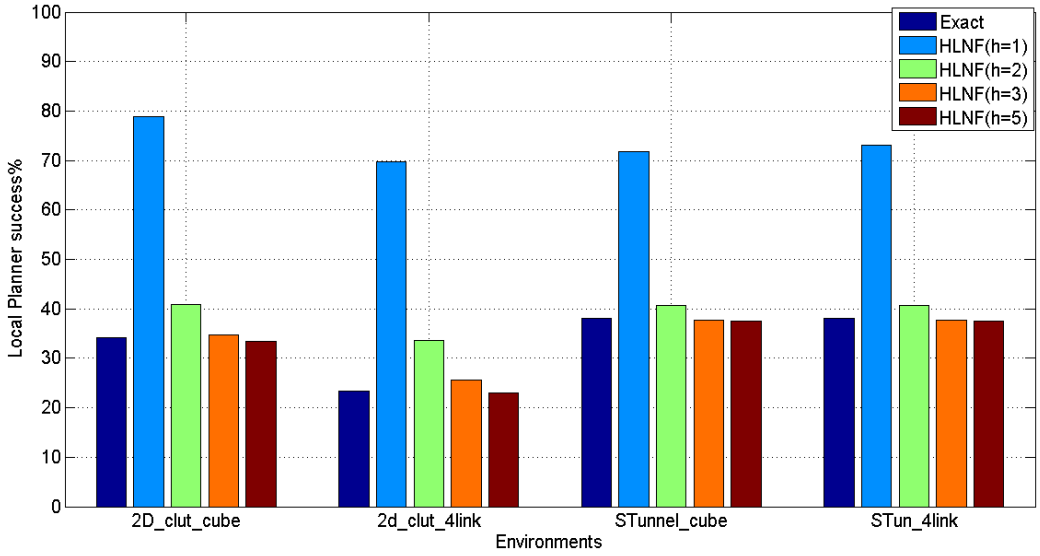


Figure 14: Comparison of local planner success (%).

increases, this set also increases and thus neighbor finding time increases. The hop limited BFS takes $O(b^h)$ time and thus the total neighbor finding time increases as h increases. The number of local planner attempts is much lower (see Figure 13) for $h = 1$ as the neighbor finder may not find enough neighbors ($< k$) within the hop limit. For higher h

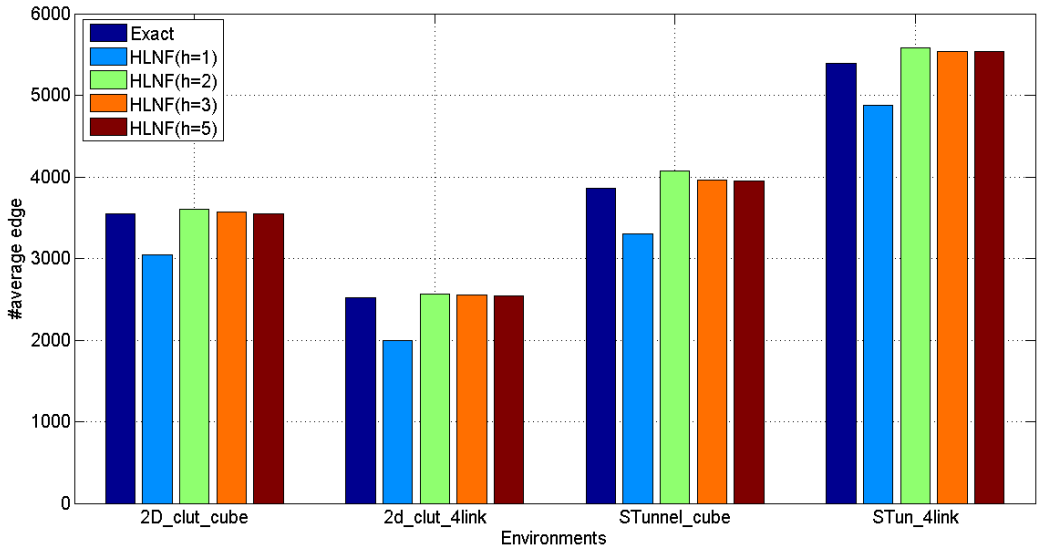


Figure 15: Comparison of #edges.

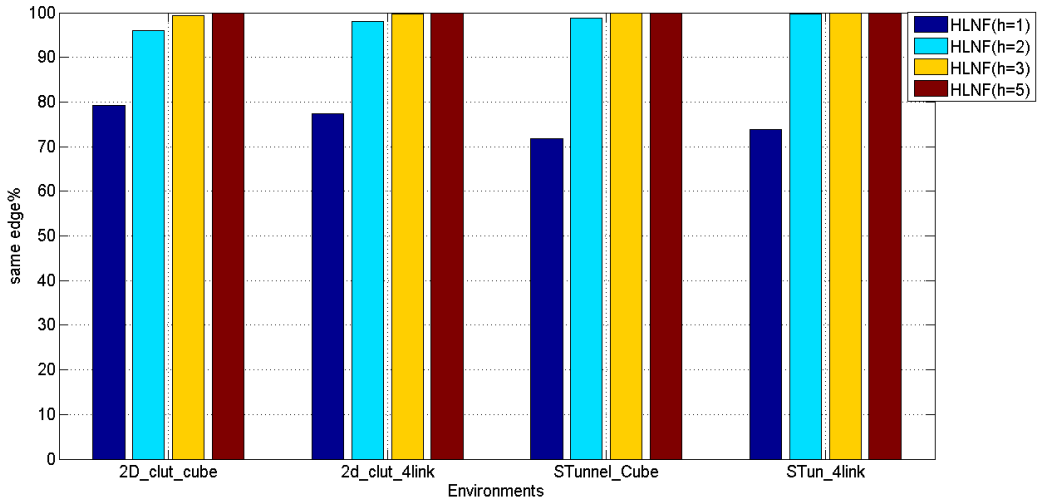
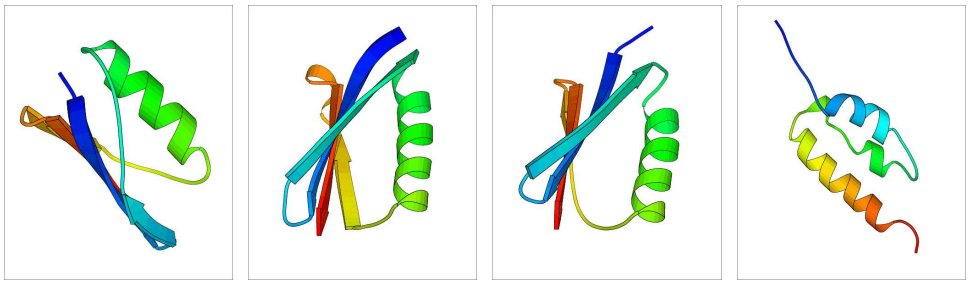


Figure 16: Comparison of same edge (%).

values, the local planner attempts increase as the subgraph for finding the neighbors grows bigger. The local planner success rate is very high for $h = 1$, as the vertices within this hop limit are very near to the query vertex and the chance of making successful connections with these vertices is high. As the hop limit grows, the local planner success rate (see

Protein	pdb	# Residues	Secondary Structure Makeup	Experimental Formation Order
G	1PGA	56	$1\alpha + 4\beta$	[α,β1,β3,β4],β2[1] [α,β4],[β1,β2,β3][2]
G Variant	NuG1/NuG2	57	$1\alpha + 4\beta$	β1-2,β3-4[3]
A	1BDD	60	3α	[α2,α3],α1[1] [α1,α2,α3][2]

Table 5.2: Proteins studied and their secondary structure formation order from: [1]hydrogen out-exchange experiments [28], [2]pulsed labeling/competition experiments [28], and [3]Φ-value analysis [36]. Brackets indicate no clear order.



(a) Protein G (b) G Variant NuG1 (c) G Variant NuG2 (d) Protein A

Figure 17: Proteins studied.

helices and β-sheets form) along the pathways does not vary between iterations by more than some threshold (10 %). This is the same evaluation scheme used previously in applying PRMs to study protein folding [41]. For the results presented here, we evaluate the secondary structure formation order after every 250 samples.We use different evaluation metrics to evaluate the quality of the roadmap for RRT and mRRG.

- **Secondary Structure Formation Order**:

  We validate a method's results by comparing its dominant secondary structure forma-

tion order to the experimentally determined order from hydrogen out-exchange [28], pulse-labeling data [28], and/or Φ-value analysis [36].

- **# Folding Pathways**: Folding pathways can be extracted by calculating the most energetically feasible path (shortest path) from every unstructured conformation to the native state of the protein.

5.3.2.1 Study of m Value

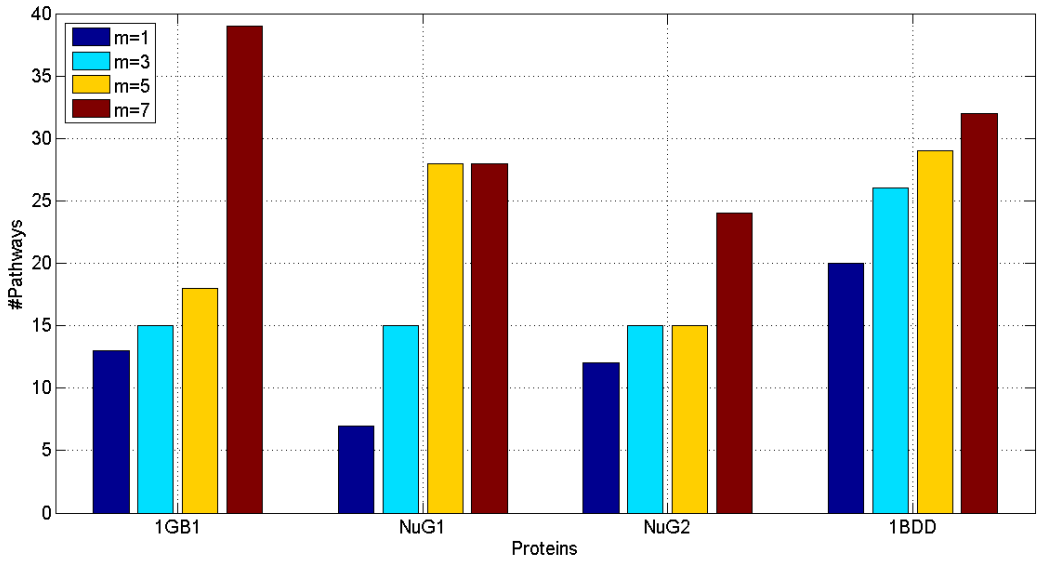


Figure 18: Comparison of #pathways produced for RRT, mRRG with $m = 1, 3, 5, 7$.

We compare the running time and resulting graph size for each method for the proteins with $m = \{1, 3, 5, 7\}$. Every method was able to reproduce the correct secondary structure formation order as seen in the experiment. We see that as m increases, the more pathways were generated (see Figure 18). The k-closest time (neighbor finding time) (see Figures 19, 20) also decreases as m increases as m expansions are done for each k-closest call (nearest neighbor finding) instead of 1 expansion in the RRT. As the k-closest time is significant,

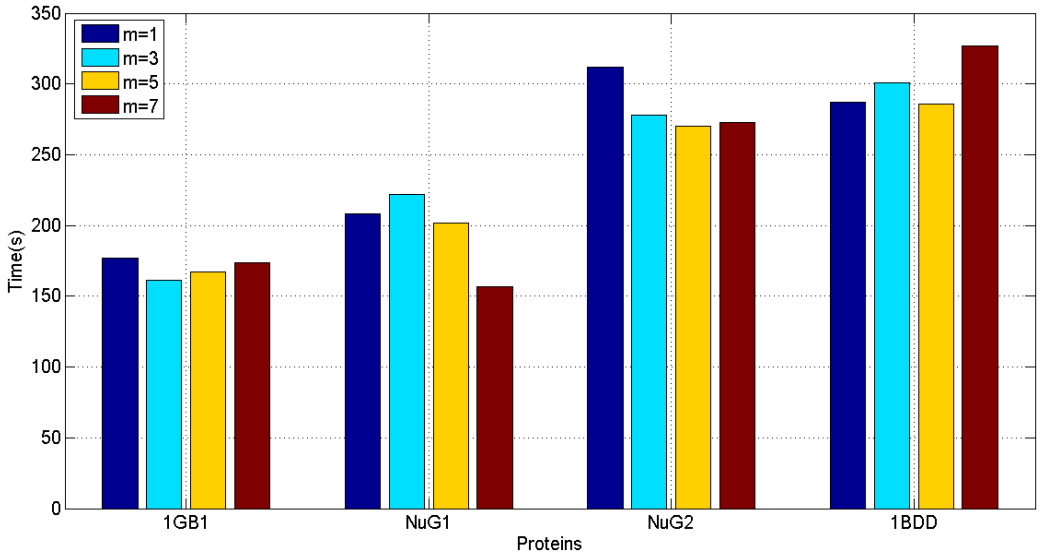


Figure 19: Comparison of total time.

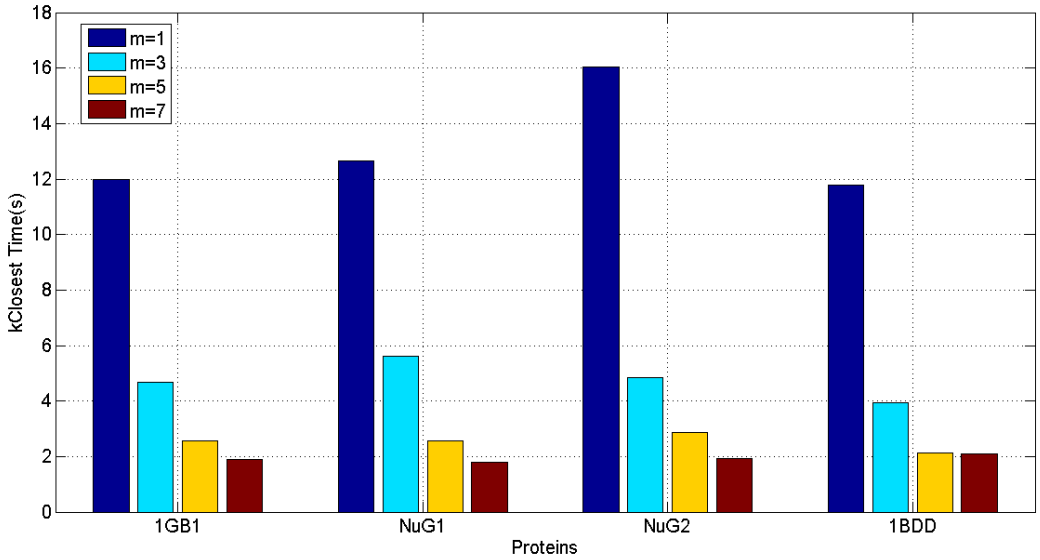


Figure 20: Comparison of k-closest time.

it is beneficial to make multiple expansions. From Figure 18, we can see that mRRG can make more pathways but uses almost the same amount of time as compared to RRT. For all the proteins shown in the experiment, $m = 7$ is the best value as it generates the most

pathways and the running time is comparable to RRT.

6. CONCLUSION

We propose a multi-directional Rapidly Exploring Random Graph (mRRG) motion planner which explores the configuration space more thoroughly compared to prior methods such as RRT and RRG. mRRG achieves this by expanding in multiple directions in a single iteration of the algorithm. This exploration also helps to generate multiple topologically distinct paths from a start to a goal given the same amount of computation time for robots and thus helps to reduce replanning time when a new obstacle appears, invalidating paths in the environment. For proteins, we show that our method is effective in achieving more unfolded pathways compared to RRT. We also show that our hop limit neighbor finder can significantly reduce running time while maintaining a similar number of roadmap edges.

In the future, we plan to develop a method to automatically tune m based on the locality of the vertex under expansion. We also plan to apply mRRG for more robots with complex shapes and to scenarios with moving obstacles. We also plan to apply mRRG to more complex proteins of larger size and to other types of protein movements such as transitions between two conformations.

REFERENCES

[1] CGAL, Computational Geometry Algorithms Library, 1997. http://www.cgal.org.

[2] E. Alm and D. Baker. Prediction of protein-folding mechanisms from free-energy landscapes derived from native structures. *Proc. Natl. Acad. Sci. USA*, 96(20):11305–11310, 1999.

[3] N. M. Amato and G. Song. Using motion planning to study protein folding pathways. *J. Comput. Biol.*, 9(2):149–168, 2002. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2001.

[4] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 271–280, 1993.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[6] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 632–637, 2002.

[7] D. Baker. A surprising simplicity to protein folding. *Nature*, 405:39–42, 2000.

[8] O. B. Bayazit, G. Song, and N. M. Amato. Enhancing randomized motion planners: Exploring with haptic hints. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 529–536, 2000.

[9] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[10] P. Bessiere, J. M. Ahuactzin, E. G. Talbi, and E. Mazer. The Ariadne's clew algorithm: Global planning with local methods. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, volume 2, pages 1373–1380, 1993.

[11] C. Branden and J. Tooze. *Introduction to Protein Structure*. Garland Pub., New York, 2nd edition, 1999.

[12] J.D. Bryngelson, J.N. Onuchic, N.D. Socci, and P.G. Wolynes. Funnels, pathways, and the energy landscape of protein folding: A synthesis. *Protein Struct. Funct. Genet*, 21:167–195, 1995.

[13] H. S. Chan and K. A. Dill. Protein folding in the landscape perspective: Chevron plots and non-arrhenius kinetics. *Proteins: Structure, Function, and Genetics*, 30(1):2–33, 1998.

[14] D.G. Covell. Folding protein α-carbon chains into compact forms by Monte Carlo methods. *Proteins: Struct. Funct. Genet.*, 14(4):409–420, 1992.

[15] V. Daggett and M. Levitt. Realistic simulation of naive-protein dynamics in solution and beyond. *Annu. Rev. Biophys. Biomol. Struct.*, 22:353–380, 1993.

[16] Y. Duan and P.A. Kollman. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282:740–744, 1998.

[17] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Comput. Graph.*, 30:171–180, 1996. Proc. SIGGRAPH '96.

[18] D. Hsu, J-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.

[19] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proc. of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[20] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[21] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe. Planning motions with intentions. In *Proc. ACM SIGGRAPH*, pages 395–408, 1995.

[22] A. Kolinski and J. Skolnick. Monte Carlo simulations of protein folding. *Proteins Struct. Funct. Genet.*, 18(3):338–352, 1994.

[23] J. J. Kuffner and S. M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 995–1001, 2000.

[24] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[25] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *New Directions in Algorithmic and Computational Robotics*, pages 293–308. A. K. Peters, 2001. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Hanover, NH, 2000.

[26] M. Levitt. Protein folding by restrained energy minimization and molecular dynamics. *J. Mol. Biol.*, 170:723–764, 1983.

[27] M. Levitt and A. Warshel. Computer simulation of protein folding. *Nature*, 253:694–698, 1975.

[28] R. Li and C. Woodward. The hydrogen exchange core and protein folding. *Protein Sci.*, 8(8):1571–1591, 1999.

[29] Jyh-Ming Lien, O. B. Bayazit, R.-T. Sowell, S. Rodriguez, and N. M. Amato. Shepherding behaviors. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4159–4164, April 2004.

[30] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[31] E. Mazer, J. M. Ahuactzin, and P. Bessiere. The Ariadne's clew algorithm. In *Journal of Artificial Robotics Research (JAIR)*, volume 9, pages 295–316, 1998.

[32] Troy McMahon, Sam Jacobs, Bryan Boyd, Lydia Tapia, and Nancy Amato. Evaluation of the k-closest neighbor selection strategy for prm construction. Technical Report TR12-002, Texas A&M, College Station, 2011.

[33] Anthony Mittermaier and Lewis E. Kay. New tools provide new insights in NMR studies of protein dynamics. *Science*, 312(5771):224–228, 2006.

[34] V. Muñoz, E. R. Henry, J. Hoferichter, and W. A. Eaton. A statistical mechanical model for β-hairpin kinetics. *Proc. Natl. Acad. Sci. USA*, 95:5872–5879, 1998.

[35] Shuvra Nath, Shawna Thomas, Chinwe Ekenna, and Nancy M. Amato. A multi-directional rapidly exploring random graph (mrrg) for protein folding. In *ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, Orlando,FL, USA, October 2012.

[36] S. Nauli, B. Kuhlman, and D. Baker. Computer-based redesign of a protein folding pathway. *Nature Struct. Biol.*, 8(7):602–605, 2001.

[37] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Trans. Robot. Automat.*, 2005.

[38] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, San Juan, Puerto Rico, October 1979.

[39] Heinrich Roder, Kosuke Maki, and Hong Cheng. Early events in protein folding explored by rapid mixing methods. *Chem. Rev.*, 106:1836–1861, 2006.

[40] G.E. Schulz and R. H. Schirmer. *Principles of Protein Structure*. Springer-Verlag, New York, 1979.

[41] G. Song. *A Motion Planning Approach to Protein Folding*. Ph.D. dissertation, Dept. of Computer Science, Texas A&M University, December 2004.

[42] G. Song and N. M. Amato. A motion planning approach to folding: From paper craft to protein structure prediction. Technical Report TR00-001, Department of Computer Science, Texas A&M University, January 2000.

[43] G. Song and N. M. Amato. Using motion planning to study protein folding pathways. In *Proc. Int. Conf. Comput. Molecular Biology (RECOMB)*, pages 287–296, 2001.

[44] M. J. Sternberg. *Protein Structure Prediction*. OIRL Press at Oxford University Press, 1996.

[45] X. Tang, B. Kirkpatrick, S. Thomas, G. Song, and N. M. Amato. Using motion planning to study RNA folding kinetics. *J. Comput. Biol.*, 12(6):862–881, 2005. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2004.

[46] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(6):175–179, 1991.

[47] Thomas E. Wales and John R. Engen. Hydrogen exchange mass spectrometry for the analysis of protein dynamics. *Mass Spec. Rev.*, 25(1):158–170, 2006.