# Modeling Protien-Protein Interactions with the Aid of Motion Planning Algorithms

Shawna Miller
Modeling Protein-Protein Interactions with the Aid of Motion Planning Algorithms
shawnam@tamu.edu

September 18, 2001

## 1 Introduction

Bioinformatics has been receiving a lot of attention lately from the research community. Bioinformatics blends computer science, biology, and chemistry together. It uses computers and techniques developed in computer science to solve many problems in biology and chemistry. Several applications include molecule and protein modeling, protein sequence alignment, protein folding, rational drug design, and database searching to cull information from large genomes and protein databanks. Bioinformatics is increasing in popularity because it has applications in all facets of life, and it is a relatively new field with very fertile ground.

Proteins interact with each other in most reactions that occur in the body. They are involved in everything from DNA transcription and replication to viral protection to energy consumption and distribution among the cells. Understanding how a protein interacts with other proteins and how it functions is crucial to understanding how the body works and functions.

There are three main types of protein-protein interactions: protein-protein interactions, protein-DNA interactions, and the interaction between monomers of multimeric proteins. Protein-protein interactions occur between two or more proteins. Some examples are the interaction involving GroEL and GroES to aid in protein folding, the interaction between calmodulin any myosin to produce muscle contraction, and protein/antibody binding. Protein-DNA intreactions involve a protein and a piece of DNA. This situation occurs mostly in DNA replication. Finally, some proteins are made up of several chains or loops. These chains are intertwined and interact with each other to dictate how the protein folds, its function, and how it interacts with other things. For example, HIV-1 protease is make up of two chains (or monomers) that move with each other.

Our goal is to study these reactions and to simulate them. The difficulty is that proteins have hundreds to thousands of degrees of freedom. Even when assumptions are made and their structures are simplified, they are still very complex and difficult to simulate in a reasonable amount of time. Because of their complexity, most simu-

lations consider proteins as rigid objects. This is an unreasonable assumption because some proteins are known to undergo large conformational changes. (They exhibit large movements durring interactions.) They should be considered flexible, not rigid.

Motion planning techniques are good at computing paths when the robot has many degrees of freedom. Several algorithms, especially Probabilistic Roadmap Methods (PRMs) and their variations, have a lot of success in a situation where the robot is complex. By considering the protein to be an articulated robot (a robot with several links), we can apply the same techniques developed for robot motion planning to protein simulation.

# 2 Evidence that Proteins are Flexible

Show proof that proteins do undergo large conformational changes. Show that the rigid assumption in grossly inadequate in some cases. Give examples and pictures.

## 2.1 GroEL/GroES Complex

Proteins may make ”bad” connections when trying to fold to their native state. These ”bad” connectections, or aggregates, can cause the protein to function improperly. Chaperones can prevent and reverse such ”bad” connections by binding to and releasing the unfolded or aggregated protein during the folding process. Chaperones do not increase the rate of protein folding, they only increase its efficiency.

GroEL and GroES work together (interact) to help peoteins fold into their native state properly. They do this by surrounding the protein like a cage thereby providing a safe environment for the protein. GroES binds to the top of GroEL and forms a cage around the protein. During this interaction, GroEL undergoes large conformational changes. Pictures of these proteins (obtained from the Protein Data Bank and viewed through RasMol) are shown in Figure 2.

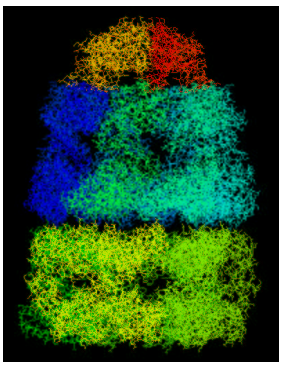
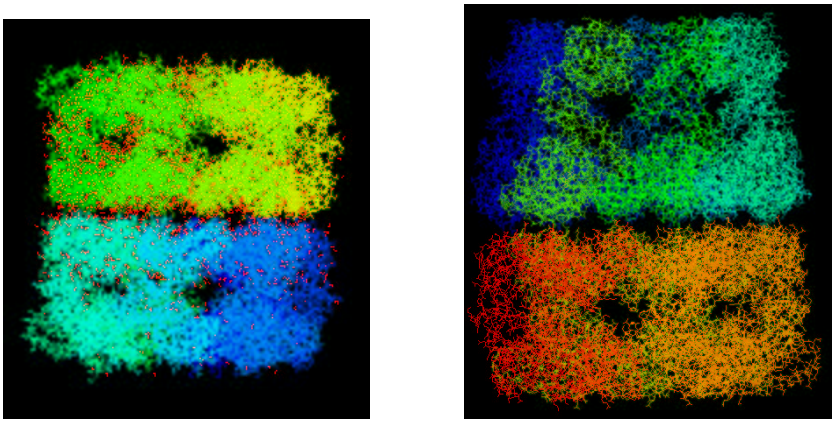


Figure 1: GroEL/GroES are two chaperones that work together to increase the efficiency of protein folding. The top “cap” is GroES and the bottom two rings is GroEL.



(a) (b)

Figure 2: GroEL is shown before (a) and after (b) binding to GroES. GroES is removed for clarity. GroEL undergoes large conformational changes during the binding process. It stretches upwards and twists in the presense of GroES.

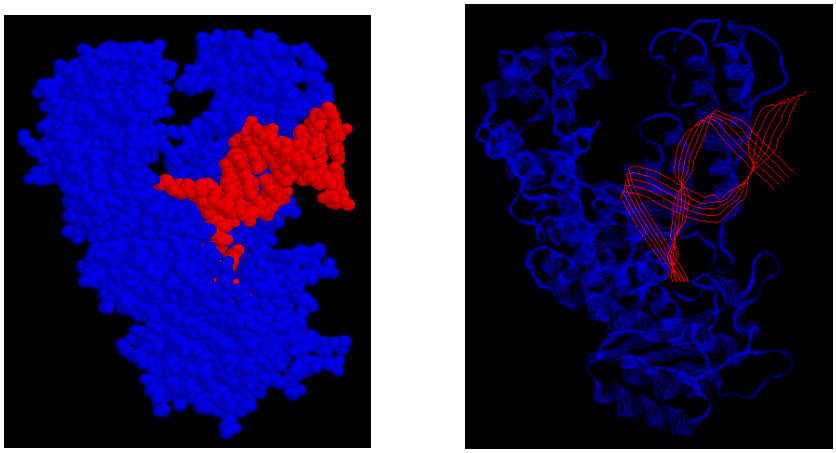
## 2.2 DNA Polymerases

DNA is made up of two helices. The are designed in such a way that given one helix, the other helix can be easily determined. Durning DNA replication, these two helicies are split apart. For each helix, the cell creates the other half. The cell taken one piece of DNA and made a copy of it. DNA polymerases catalyze this process.

DNA polymerase I (Pol I) was the first enzyme discovered to help synthesize DNA. Pol I has three main functions: acts as a polymerase, acts as an exonuclease in the 3'→5' direction, and acts as an exonuclease in the 5'→3' direction. As a polymerase, it helps create the second helix by binding the correct bases. As an exonuclease, it can correct its mistakes. The exonuclease activity is like proofreading.

This enzyme is shaped like a hand (called the Klenow fragment), see Figure 3. When it functions as a polymerase, it binds to the DNA just like you would grab a rod with your hand. When it functions as exonuclease 3'→5', the protein undergoes a large conformational change and forms another cleft perpendicular to the cleft that contains the polymerase site. There is yet another binding site for the third function, exonuclease 5'→3'.

The DNA also changes conformation during interaction with Pol I. As Pol I "grabs" the DNA, it bends it about 80 degrees. This is large enough that it is no longer realistic to consider it a rigid object. The DNA, as well as the protein, must be thought of as flexible.



(a) (b)

Figure 3: DNA Polymerase I is shaped like a hand — shown in blue. It "grabs" the piece of DNA during DNA replication when it functions as a polymerase. The protein is shown both space-filled (a) and as a ribbon (b).

### 2.3 Calmodulin

Calmodulin regulates many important functions in the body by reacting to changing calcium (Ca2+) levels. For example, it interacts with myosin to perform muscle contractions in the body. Its sensitivity to calcium levels is due to its readiness to bind to calcium.

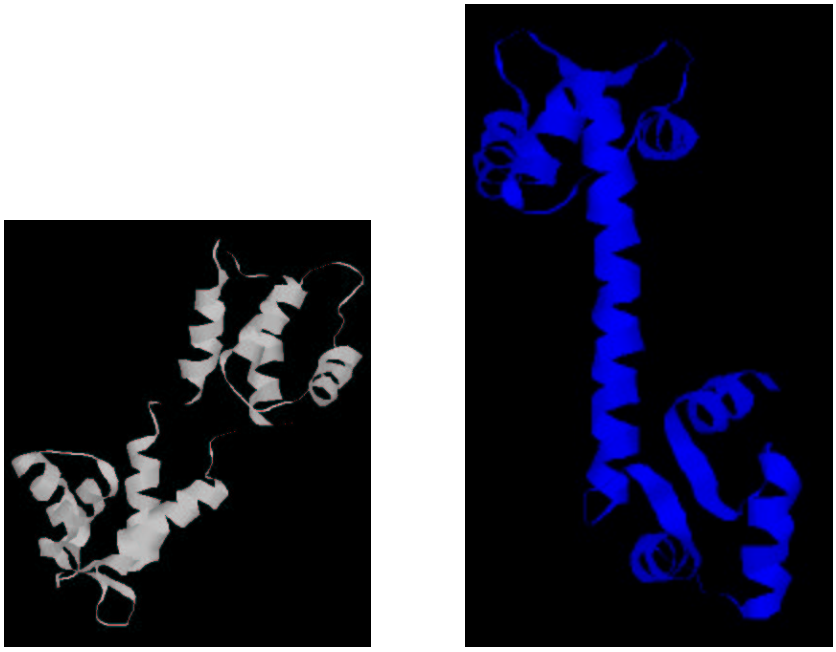
Calmodulin has two globular domains connected by a single alpha helix (see Figure 4, b). Each globular domain contains 2 Ca2+ binding sites. Calmodulin undergoes *large* conformational changes when bound to Ca2+. Also, when bound to myosin, the globular domains remain relatively unchanged, but the alpha helix connecting them unwinds and contains a sharp bend. The drastic change in conformation is mainly due to the change in the alpha helix.

## 3 The Project

The goal of our research this summer is to simulate interactions between proteins. As discussed above, proteins are complex, *dynamic* structures. Some motion planning algorithms have had much success in computing paths for very complex robots. We want to apply these techniques from robotics to protein-protein interaction simulation.

### 3.1 Background

One class of motion planning algorithms, Probabilistic Roadmap Methods (PRMs), have been very successful in computing paths where the robot has many degrees of



(a) (b)

Figure 4: Calmodulin is shown in its unbound (a) and bound (b) states. This large conformational change is due to the central alpha helix as it winds and unwinds.

freedom in a reasonable amount of time. Although PRMs are not complete (i.e. the are not guaranteed to find a path if one exists), they are able to find solutions to many problems quickly. Complete algorithms do exist, but they are prohibitively long and computationally expensive. PRMs sacrifice completeness for speed.

PRMs build a roadmap through the robot's configuration space (C-space) that the robot can use to navigate its environment. A configuration is a unique position and orientation of the robot. The C-space consists of all configurations, valid or not. In robotics applications, a valid configuration is considered to be one that is entirely collision-free. The roadmap is much like a state highway map. It consists of nodes (cities) and edges (streets).

Roadmap construction consists of two phases: node generation and node connection. During node generation, nodes are created that form the basis of the roadmap. These nodes can be generated in a number of ways. The traditional PRM generates these uniformly at random. These are easy to compute and provide good coverage of the C-space. Variations of the traditional PRM used other methods to generate nodes. During node connection, PRM tries to connect each node with its $k$ closest neighbors via some local planner. Variations of PRM use different methods to connect the nodes and different local planners.

Once the roadmap is built, a path between any start configuration and goal config-

uration is easily found. First the start and the goal are connected to the roadmap. Then the roadmap is searched for the shortest path between the two nodes using a graph search algorithm.

One particular variation of PRMs are focused on single-query planning. Instead of building a roadmap that can solve multiple queries, or start and goal pairs, they tailor the roadmap for one particular query. Two similar methods were developed independently at Iowa State University and Stanford University, Steve LaValle's Rapidly-exploring Random Trees (RRT) and David Hsu's planner for expansive configuration spaces. Both of these methods grow a roadmap from the start towards the goal and from the goal towards the start until they meet.

RRT alternates node generation and node connection as it expands, or grows, the roadmap. First a node is generated at random. This node specifies the direction of expansion. Then the algorithm selects the closest node to the new node. It makes a small step from this node towards the direction node. If it is collision-free, it adds this step to the roadmap and connects it to the node it began from. After each new node and new edge is added, the algorithm checks to see if the goal has been reached or, in the case of two trees, if the trees meet each other. This process is repeated until a solution is found. Since the growth is biased by random nodes, the expansion tends to be a global one, pulling the roadmap out into unexplored regions of the C-space.

Hsu's algorithm differs in how expansion is biased. Instead of picking a random node and walking towards it, his algorithm picks a node, $x$, in the tree based on some probability. Then several new nodes are generated in the neighborhood of $x$. Some of these nodes are kept and only added to the roadmap if an edge exists between it and $x$. Again, after eact new node and new edge is added to the roadmap, the algorithm checks if the goal has been reached or the two tree meet. This algorithm implements local expansion through neighborhoods while RRT uses a global expansion by random sampling.

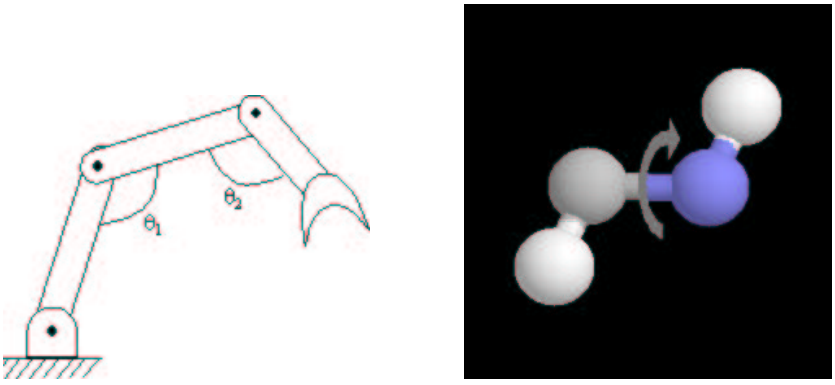
## 3.2 Biology Considerations

Motion planning algorithms were designed for robotics applications. With just a description of the robot, the environment, and a collision-checker, difficult motion planning problems can be solved with ease. These same techniques, although originally intended for robots, can be applied to proteins.

Proteins are made up of atoms and bonds. Each atom can be modeled as a sphere, and each bond can be modeled as a rod that connects two atoms together. The protein can be considered to be an articulated robot, or one with multiple links. Here, the bonds are the robot's links and the atoms are the robot's joints.

Linked robots move based on changes in joint angles, see Figure 5, (a). A joint angle is the angle between two consecutive links. The number of joint angles plus the position and orientation of the base are the robot's degrees of freedom.

Proteins behave slightly differently. Chemists have discovered that bond lengths and bond angles (the angle between two consecutive bonds) do not change significantly during conformational changes. We can safely assume that the bond lengths and bond angles are fixed. Torsional angles, on the other hand, do change significantly when the protein's conformation changes. It is *the* contributing factor to changes in




(a) (b)

Figure 5: (a) Linked manipulators move based on their joint angles, $\theta_1$ and $\theta_2$. (b) Likewise, proteins move based on their torsional angles.

conformation, see Figure 5, (b). The number of torsional angles plus the position and orientation of the root atom are the protein's degrees of freedom.

The goal of motion planning algorithms is to produce feasible paths for the robot. These paths must be realistic. In order for a path to be feasible, at every point along the path the robot must be collision-free. (A collision-free robot is one that is not colliding with itself or any other obstacle in the environment.)

The same principle holds true for proteins. Any computed path must be feasible so the simulation is realistic. The notion of a valid/feasible configuration is more complicated for a protein than for a physical robot. Not only must the protein be collision-free, it must also be energetically reasonable. In nature, protein conformations typically have low energies. The same must be true for computed conformations.

This property can be easily included in the collision-checker. Instead of merely checking for collision, the collision-checker will now also check the energy. By modeling proteins as articulated robots and including the energy function in the collision-checker, the same motion planning algorithms developed for robots can be directly applied to proteins.

Unfortunately, exact energy calculations are very time consuming. They would be inappropriate to include in a collision-checker that is called thousands of times during roadmap creation. Some assumptions can be made to reduce the running time of the energy calculations. First, we will only consider the van der Waals energy. Electrostatic energy and torsional energy make up a small part of the total energy, so they can be safely neglected.

To compute the van der Waals energy, every pair of atoms must be considered. Atoms that are close together contribute largely to the total energy. Likewise, atoms that are far apart do not contribute a whole lot to the total energy. We can approximate the van der Waals energy by neglecting pairs of atoms that are far apart. To do this, we impose a cutoff distance. If the distance between the two atoms is larger than the cutoff

distance (8 Å in our case) then the energy for that pair is not computed. This reduces the running time of the energy calculation and still gives a good approximation of the energy.

In summary, motion planning algorithms like PRM and RRT can be applied to protein-protein interactions by modeling the protein as an articulated robot and including the energy function in the collision-checker. If the energy calculation is accurate and efficient, then these motion planning algorithms will perform well on proteins and could provide greater insight into how proteins move and function.

## 3.3 Basic Algorithm

Our algorithm for modeling protein-protein interactions is an extension of RRT and Hsu's planner for expansive C-spaces with one large variation. Instead of building trees, we will build graphs. This way we can look at many different paths from the start (unbound state) to the goal (bound state) instead of just one solution. This will improve the quality of the "best" solution and increase our understanding of protein-protein interactions.

The first conscern is to generate energetically feasible nodes to build the roadmap with. Because the conformational space is n-dimensional, it is nearly impossible to perform a systematic search. If we just generate the nodes randomly, the likelihood that we will find "good" nodes is very small. To combat these difficulties, we first generate a node randomly and then perform a gradient descent to minimize its energy.

Performing an exact gradient descent is too time consuming to consider. Instead, we approximate a gradient descent. To do this we first generate a random node near the original node and compute its energy. If its energy is less than the original node, we declare it to be the new minimum and replace the original node. If its energy is greater than the original node, we throw it away. This process is repeated many times, typically anywhere from 10 to 30 iterations.

To generate a random nearby node, we first select a few rotable bonds at random. We then apply a small, random displacement in the torsional angles of these bonds. The resulting conformation is randomly generated, but very similar to the original conformation.

Now we have all the building blocks to implement a variation of RRT and Hsu's algorithm. Roadmap construction is as follows:

BUILD_ROADMAPS($q_{start}$, $q_{goal}$)
1. $R_{start}$.init($q_{start}$)
2. $R_{goal}$.init($q_{goal}$)
3. **for** $n = 1$ to $N$ **do**
4. EXTEND($R_{start}$)
5. EXTEND($R_{goal}$)
6. CONNECT_MAPS($R_{start}$, $R_{goal}$)

EXTEND($R$)
1. $q_{orig} \leftarrow$ SELECT_NODE($R$)
2. $q_{new} \leftarrow$ GENERATE_NEIGHBOR($q_{orig}$)
3. MINIMIZE_ENERGY($q_{new}$)
4. $R$.add_node($q_{new}$)
5. $R$.add_edges($q_{new}$)

We build two roadmaps, one rooted at the start conformation and one rooted at the goal conformation. During each iteration, each roadmap is extended. Then the altorithm attempts to connect the two roadmaps together. We are looking for multiple paths, so we do not stop the algorithm once CONNECT_MAPS() is successful. If we were looking to save time and only compute one path, we would stop the algorithm as soon as CONNECT_MAPS() is successful and a path is found.

The EXTEND() method simply generates a new nearby node, minimizes its energy, and then adds it to the roadmap. Then add_edges() checks for connections between the new node and its $k$ closest neighbors. This allows us to build a graph, instead of a tree, and search for multiple solution paths.

When edges are checked for validity, every node's energy along that edge is computed. We can use this information to compute an edge weight. The simplest scheme is to let the edge weight equal the sum of all node energies along that edge. This gives a higher weight to paths with higher energies. It also gives longer edges a higher weight. This may unduly bias the algorithm to look for shorter paths, ignoring longer paths that may be energetically feasible. To avoid this, the average energy along the edge could be stored as the edge weight intead.

As long as higher weights correspond to edges with high energies and lower weights correspond to edges with low energies, edge weights can identify the most energetically feasible paths. A graph search that looks for a path with the lowest total weight would pull out the most energetically feasible, or "best", path.

## 3.4 Implementation

So far, we have implemented several pieces of the algorithm in C++. We began with a framework developed by Ming Zhang, a postdoc in Dr. Kavraki's research group. This framework supplied a working representation of molecules and proteins. We are using the Atomroup Local Frames approach, developed by Zhang, to quickly calculate the new xyz postitions of the atoms. An atomgroup is simply a group of atoms that remain fixed relative to each other. Such groups may be rings, protein sidechains, or atoms connected by nonrotable bonds.

The code can input and output files in the `mol2` format[1]. We selected this format for a number of reasons. First, it is very intuitive to use and code. Second, Rasmol, a free molecule visualization tool, works with `mol2` files. Finally, and most importantly, we can utilize the thousands of protein structures stored in the Protein Databank (PDB). These strutures are stored as `pdb` files, but since they are not as intuitive, we use Sybyl[2] to convert them to the `mol2` file format.

[1] The `mol2` file format developed by Tripos is used by many biochemists.

[2] Sybyl is an extensive tool for visualization and biological computation developed by Tripos.

With Paul Murphrey's help, another member of Dr. Kavraki's group, we have added basic energy calculations to the implementation. These calculations compute the van der Waals energy of the molecule. This energy calculation uses a distance cutoff of 8 Å. This distance is standard in the biochemistry community. To save computation time, the energy dervived from pairs of atoms in the same atomgroup is only computed in the first energy calculation. Since these atoms do not move relative to each other, their van der Waals energy is constant. Depending on how the atomgroups are defined, this can greatly reduce computation time.

We have also implemented the GENERATE_NEIGHBOR() and MINIMIZE_ENERGY() methods. These were fairly straightforward to implement but need some optimization work.

# 4 Future Research

The next step is to put all the pieces together to develope the entire algorithm. The Robotics group at Texas A&M University has a good implementation of a PRM framework. Working with that group last summer and last year, I have a good working knowledge of their code. To have a working algorithm, all that is left is to integrate the pieces developed this summer into their PRM framework.

As mentioned earlier, some optimization work is needed to reduce the running time. An obvious approach is to compute energy calculations in parallel. Also, every node along an edge must be checked for validity to add that edge to the roadmap. Checking a node is independent of the other nodes, so this can be done in parallel, giving a node (or set of nodes) to each processor. Since most of the running time is spent in checking validity and computing energies, these improvements will produce a significant reduction in running time. We need to look into other ways to parallelize the code.

Once the entire algorithm is implemented and working, we can look at many different protein-protein interactions. We would like to first consider the calmodulin/myosin interaction for two reasons. First, calmodulin undergoes *large* conformational changes, the exact situation our research is targeting. Second, it is already known how calmodulin moves to bind to myosin. We can use this information to test the validity of our results.

The work this summer provides a good foundation for future research. This research will provide insight into how proteins move and function. It will mainly be used to study how proteins interact with each other and other biological substances in the body. This knowledge has the power to impact the bioinformatics community as a whole, especially pharmaceutical drug design and molecular modeling.