Loop Modeling in Proteins Using a Database Approach with Multi-Dimensional Scaling

by

Daniel Holtby

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Modeling loops is an often necessary step in protein structure and function determination, even with experimental X-ray and NMR data. It is well known to be difficult. Database techniques have the advantage of producing a higher proportion of predictions with sub-angstrom accuracy when compared with *ab initio* techniques, but the disadvantage of often being able to produce usable results as they depend entirely on the loop already being represented within the database. My contribution is the LoopWeaver protocol, a database method that uses multidimensional scaling to rapidly achieve better clash-free, low energy placement of loops obtained from a database of protein structures. This maintains the above-mentioned advantage while avoiding the disadvantage by permitting the use of lower quality matches that would not otherwise fit. Test results show that this method achieves significantly better results than all other methods, including Modeler, Loopy, SuperLooper, and Rapper before refinement. With refinement, the results (LoopWeaver and Loopy combined) are better than ROSETTA's, with 0.53Å RMSD on average for 206 loops of length 6, 0.75Å local RMSD for 168 loops of length 7, 0.93Å RMSD for 117 loops of length 8, and 1.13Å RMSD loops of length 9, while ROSETTA scores 0.66Å, 0.93Å, 1.23Å, 1.56Å, respectively, at the same average time limit (3 hours on a 2.2GHz Opteron). When ROSETTA is allowed to run for over a week against LoopWeaver's and Loopy's combined 3 hours, it approaches, but does not surpass, this accuracy.

Acknowledgments

I would like to thank my supervisor, Professor Ming Li, for his time, patience, and encouragement. Without him none of this would have been possible.

I would also like to thank my fellow students (past and present) Shuai Cheng Li, Guangyu Feng, and Xuefeng Cui, all of whom have provided valuable discussions and new ideas.

I would also like to acknowledge Sharcnet for the use of their high throughput computing facilities. Without these clusters would have taken a great deal longer to get all of the results back.

Dedication

This is dedicated to my wife, Diane, for her love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Proteins and Protein Structure

A protein is a biomolecule made up of one or more polypeptide chains, which are polymers made up of a sequence of amino acids.

Each amino acid is an amine group (NH_2) and a carboxyl group (COOH), connected together by a central carbon atom (The alpha carbon or C_α), as shown in Figure 1.1. The amino acids differ by the *side group* (more often called the *side chain*) which is attached to the alpha carbon. This is shown as the R box in Figure 1.1.

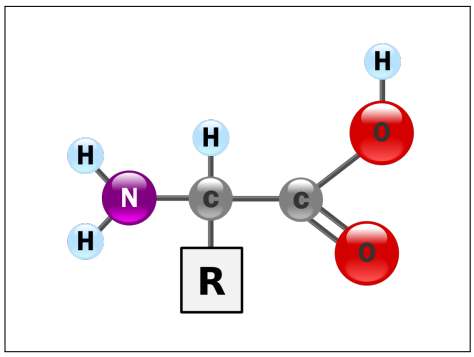


Figure 1.1: Amino acid 2D diagram[1]

A polypeptide chain is a chain of these amino acids, connected together by a *peptide bond* formed where the carboxyl group of one amino acid forms a bond with the amino group of another, with these groups losing an OH and H respectively as water. (The peptide bond is marked in Figure 1.2).

Traditionally the N atom to C atom direction is viewed as the forward direction in the chain as this is the order that the protein chain is assembled. The *backbone* of a protein refers to the common structure, the amine group, alpha carbon, and carboxyl group. The three atoms that bond together to form the chain (Nitrogen, Alpha Carbon, and Carbon) as well as the doubly bonded oxygen in the carboxyl group are referred to as the *heavy backbone atoms*. There are several important pieces of geometry involved in the backbone chain of a protein. Every pair of atoms defines a *bond distance*, and these distances are fairly tightly constrained. Every three adjacent atoms defines a *bond angle*, and every four define a *dihedral angle*, which characterizes the rotation about the bond between the middle two atoms. The three dihedral angles are called φ, which involves atoms C-N-C_α-C, ψ, which involves atoms N-C_α-C-N, and ω, which involves C_α-C-N-C_α. The peptide bond is highly planar and so the ω angle is restricted to be very close to 180° (the *trans* case), or in rare cases, 0° (the *cis* case). Figure 1.2 shows the three dihedral angles, as well as an example bond angle τ. The bond angles are regular triangular bonds, or in the case of τ (the N-C_α-C bond angle), tetrahedral. So they have the idealized values of 120°, or 109.47° for τ. The exact geometry depends on the other atoms bonded to the chain, so the averages for each bond angle are not equal to the idealized triangular or tetrahedral values. For example, Engh and Huber[4] report average τ values of 109.1° to 110.1° depending on the amino acid, with standard deviations of approximately 2 degrees.

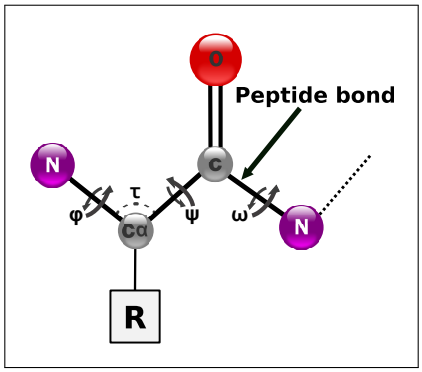


Figure 1.2: Dihedral angles of protein backbone. Adapted from [1]

As with the bond angles, the mean bond lengths depend on the amino acid involved, but the standard deviations for each amino acid type are all 0.02Å or lower (for the backbone bonds). So, given an amino acid sequence, one can assume that the bond angles and bond distances are equal to their expected values without introducing too much error. As the ω torsion angle is also fairly restricted (with a standard deviation of only a few degrees), one can characterize a protein chain as a series of torsion angle pairs (φ,ψ)

without much loss of accuracy.

1.1.1 Protein Structure

The structure of a protein is split into four levels, called the primary through quaternary structures. Figure 1.3 shows these levels.

Primary Structure

The primary structure of a protein is the sequence of residues that make up the chain(s). The amino acids are commonly abbreviated using either 3 letter or 1 letter abbreviations. The primary structure of a protein is thus a sequence of characters drawn from an alphabet of 20.

Secondary Structure

Secondary structures are regular local structures (or the lack thereof) and these structures are broadly grouped into three categories. The most common sort of secondary structure is the α helix. An α helix is a right-handed helix which involves the amine group of each residue donating a hydrogen bond to the carbonyl group four residues back in the chain, forming the backbone into a coil shape. Two less common forms of helix are the 3_{10} helix where the hydrogen bonds form between residue i and $i+3$, and the π helix, where the hydrogen bonds form between residue i and $i+5$. In an α helix, there is one complete turn of the helix per 3.6 residues, while in 3_{10} and π helices there are 3.0 and 4.4 residues per turn, respectively. Helix structures are the most common structure involved in membrane spanning proteins[5, Chapter 10]. Helices can be made up of any amino acids, but proline and glycine are rare. Proline has no amide hydrogen to contribute, so it cannot form the requisite hydrogen bond. This combined with steric considerations due to its cyclic sidechain means that any proline present in a helix will result in a break or a kink. Glycine is rare for the opposite reason. Glycine's flexibility due to its lack of a side group means that it is expensive in terms of energy for it to adopt the rigid helix structure.

The next most common sort of secondary structure is the β sheet. A β sheet is made up of two or more β strands. A β strand is a length of

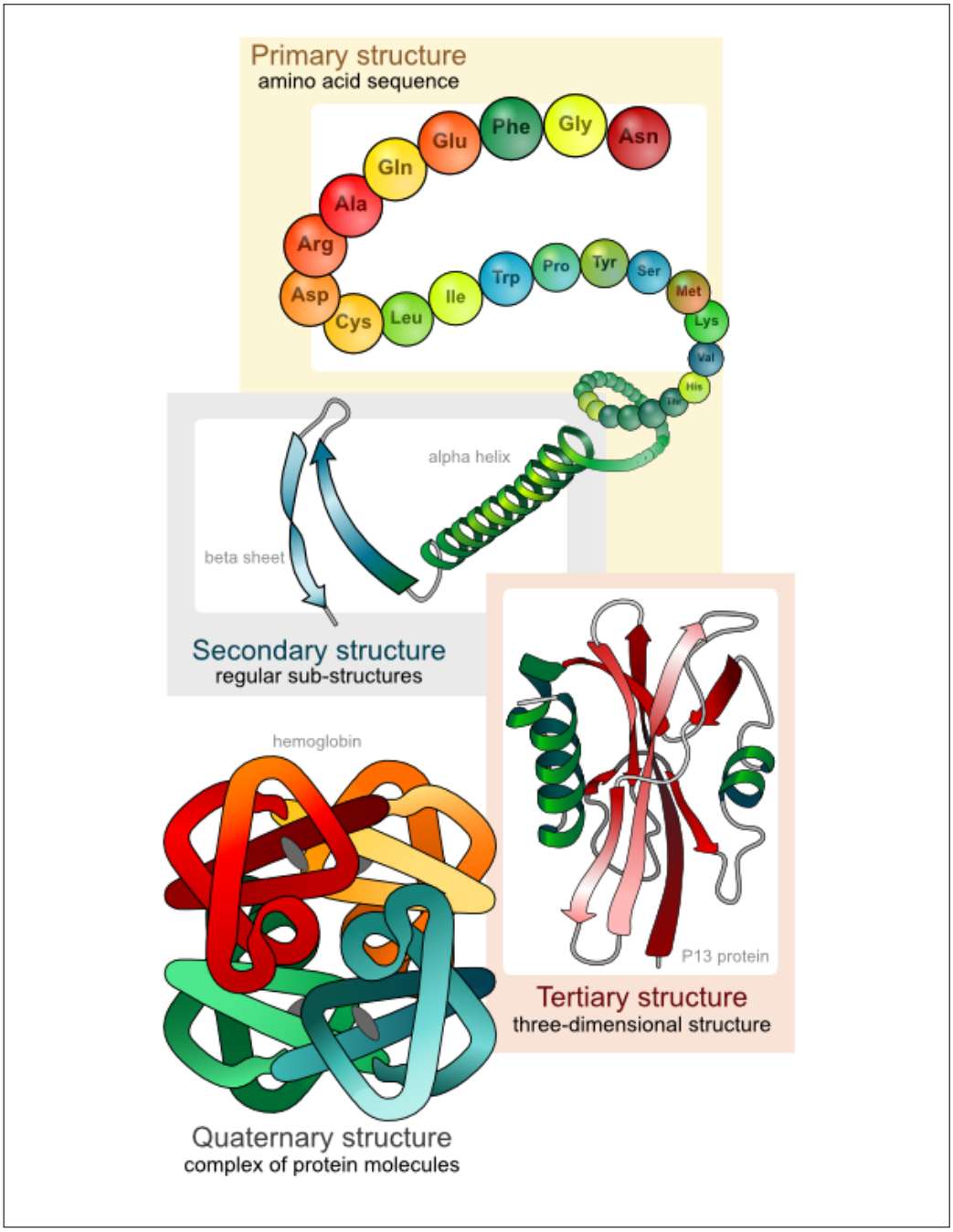


Figure 1.3: Protein Structure Levels[2]

chain that has taken on a conformation that is nearly fully extended. For this reason such structures are also referred to as *extended* structures. Beta sheets form when two strands are oriented side-by-side, and hydrogen bonds form between the carbonyl group in a residue in one strand, and the amide group in a residue in the other strand. A β strand is often drawn as a ribbon terminated with an arrowhead indicating the N-to-C direction of the chain. A β sheet can form when two strands are oriented in opposite directions (meaning that the lowest numbered residue in one strand is bonded with the highest numbered residue in the neighboring strand), in which case it is an anti-parallel β sheet, or when they are both oriented in the same direction (meaning that the lowest numbered residue in one strand is bonded with the lowest numbered residue in the other strand), in which case it is a parallel β sheet. In an anti-parallel sheet the hydrogen bonds between the carbonyl and amide groups are planar, which is their preferred orientation, so this configuration is the strongest. The offset between these groups in a parallel sheet makes this configuration slightly less stable. As only every second residue is involved with the hydrogen bonds between two strands, additional strands can be incorporated on the open sides. It is possible for a strand to be parallel to one neighbor, and anti-parallel to the other.

Any region that is not either a helix or a strand is called a loop region, though they are sometimes called by other names. Turns are a subcategory of loop, where a small number of residues connect adjacent secondary structures. A turn connects two secondary structures with a C_α to C_α distance of less than 7Å, and involves 1 to 5 residues, called respectively δ-,γ-,β-,α-, and π-turns, with the 3 residue β turn being the most common. Under this nomenclature anything longer which connects nearby structures is an ω-loop, named for the shape of the Greek letter Ω. A turn that connects two bonded anti-parallel $\beta-$strands is called a β-hairpin. In this name the β refers to the sheet it is part of, not the turn type, and a hairpin turn does not need to be a length 3 β-turn.

A region that connects more distant end-points is called a loop (without a Greek letter). These are also sometimes called coil regions (such as in the DSSP[6] classification of secondary structures). Confusion can arise as in some cases helix regions are also called coils (such as in the "coiled coil" motif, where two helices are wrapped around each other), so loop is the less ambiguous term.

Although loops and turns are defined by their lack of regular hydrogen

bonding, they can be involved in some hydrogen bonds. Turns often have an internal hydrogen bond, in which case they can be called hydrogen-bonded turns. Loops can also form hydrogen bonds between one another, an occurrence called a β-bridge.

Tertiary and Quaternary Structure

The tertiary structure of a protein chain is the three dimensional configuration of the atoms that make up the protein chain, while the quaternary structure refers to the three dimensional configuration of multiple protein chains making up a protein. The tertiary structure is sometimes called the *fold* of the protein, although more often the fold of a protein refers to the general architecture rather than to the specific atomic coordinates. The formation of tertiary and quaternary structure is largely driven by hydrophobic interactions inducing a state where the hydrophobic core of the protein is protected from water, and stabilized by non-covalent interactions and disulfide bonds.

Super-secondary Structure

Proteins are sometimes viewed as having a less well-defined level of structure, called super-secondary structure. These can also be called *structural motifs*, or *local folds* (as opposed to the global fold referring to the tertiary structure). Usually local fold is used to refer to the tertiary structure of such a subunit, while super-secondary structure refers to the general architecture. For example, when making such a distinction, helix-turn-helix and beta-hairpin-beta would be motifs, while the specific atomic coordinates of these regions would be local folds.

Ramachandran Basins

Although the φ and ψ torsion angles have a much wider range of values than the other geometric parameters of a protein chain, they are still restricted to certain ranges. In 1963, Ramachandran, Ramakrishnan, and Sasiskharan introduced what is now called the Ramachandran Plot, a way of visualizing the φ and ψ angles of a protein chain[7]. They also obtained feasible

ranges for the φ,ψ angle pairs based on hard-sphere calculations. Different parameters to these calculations yielded different ranges, resulting in an approximate density map. These density maps are commonly referred to as Ramachandran basins. Experimental data has validated the shape of these density maps, which are generally the same shape for all amino acids other than proline and glycine.

1.1.2 Determining Protein Structure

Protein structures are typically determined experimentally by using X-Ray Crystallography. This involves crystallizing the protein so that the protein crystal will diffract an X-Ray beam into many directions. Measuring the angle and intensity of these diffractions allows a researcher to produce a three dimensional picture of the electron density within the crystal. These density maps can be used to produce the mean positions of atoms within the structure, as well as to locate and measure various chemical bonds. The electron density map yields only averaged atomic coordinates so it is not possible to create a model that exactly fits the data. Each residue in an X-Ray model has an R-Value, which measures the difference between the observed electron density map, and the map that would result if the model were the true structure.

An alternative method of determine protein structure is a Nuclear Magnetic Resonance (NMR) spectroscopy experiment [8]. NMR spectroscopy uses the resonance frequencies of atomic nuclei when exposed to a strong magnetic field in order to deduce several chemical and physical properties of the molecules containing said nuclei. Each isotope has a characteristic frequency at which its nucleus resonates. Each nucleus has a *spin*, either $+\frac{1}{2}$ or $-\frac{1}{2}$, and this spin generates a magnetic field. In the presence of a strong magnetic field the nuclei will align with the direction of the field, either with or against it depending on the nucleus' spin. Because there is a difference in energy for these two orientations, the two spin values will have a different energy level. Once this orientation occurs a second, perpendicular, magnetic field is used, allowing the researchers to measure the energy difference. The electrons involved in different chemical bonds involving each atom provide a shielding effect, and thus different chemical bonds shift the magnetic field strength that will result in a given energy difference. This is referred to as

a *chemical shift*. In small molecules, each nucleus undergoes a unique chemical shift. Proteins and other biomolecules are too large for this to be true. This necessitates using multiple experiments to correlate chemical shifts and uniquely identify each nucleus. These experiments also yield restraints on the physical structure of the protein. For example, NOE (nuclear Overhauser effect) experiments result in maximum distance constraints between atoms. With enough constraints on the geometry of the atoms, a structure can be deduced.

1.1.3 Predicting Protein Structure

A protein's sequence can be determined by sequencing the protein directly, or by sequencing a genome and using gene finding to identify protein encodings. While protein sequencing is not simple, it is a cheaper and faster experiment to perform compared to the experiments used to determine tertiary structure. Because of this disparity, there are hundreds of protein sequences deposited into public repositories for every three dimensional structure deposited. For example, in 2008, the UniProtKB database [9] contained over 5 million protein sequences, while the Protein Databank (PDB) [10] contained approximately 50,000 structures, representing 1% of known protein sequences. As of May 2013 the UniProtKB database now contains roughly 36 million sequences, while the PDB has expanded to 90,000 entries. Table 1.1 shows how the ratio of sequences to structures in these two public databases has changed over time. We can see that the sequence-structure gap is increasing as sequencing becomes faster and cheaper, while X-Ray experiments remain difficult to set up and conduct. Additionally, while X-Ray techniques continue to overcome previously formidable hurdles, the approach still faces difficulty for some types of proteins, particularly membrane proteins [11]. This creates substantial interest in accurate tertiary structure predictions based on the sequence of a protein.

Energy Functions

Under the *principle of minimal frustration*, proteins are viewed as having been selected by nature to be very stable. That is, to have very low *free energy*. Further, they are believed to have a *funneled energy landscape,* in which random three dimensional configurations for a given protein sequence

Table 1.1: Sequence-Structure Gap Over Time

Sequence Structure Gap, as the ratio between UniProtKB/TrEMBL sequences and PDB structures. Values are taken from the annual UniProtKB/TrEMBL[12] and PDB[13] release statistics.

Year	Sequence-Structure Gap
2012	223
2011	165
2010	151
2009	104
2008	81
2007	68
2006	53
2005	57
2004	38
2003	21
2002	14

are generally directed downward, in terms of their free energy, before converging on the native state at a global minimum energy level [14]. It is not practical find the protein model with minimum free energy, as it is difficult to compute the free energy, and the search space is extremely large. Nevertheless, approximations of the free energy, called *energy functions*, are useful for evaluating protein model candidates. Further, protein models can be generated and refined by simulating the natural folding process whereby a chain progresses through this funneled energy landscape toward the native conformation. Because the energy landscapes are only generally funnel shaped, such an algorithm must be *hill-climbing* in order to escape local minima (sometimes called energy basins).

Ab Initio Protein Folding

Ab initio, or *de novo*, protein folding is the attempt to build a tertiary protein structure using only the primary structure and without referencing existing protein models. In general the approach involves sampling from some form

of conformation space, and adjusting the sampled candidates (or *decoys*) to reduce their free energy. One can, for example, sample torsion angles from an angle space based on the Ramachandran basins for each residue in the sequence. In ROSETTA[15] the angles are sampled from a constrained angle space that is generated using fragment matches. That is, ROSETTA locates many 3 and 9 residue long protein segments with very high sequence identity to the query sequence, and uses these fragments to generate a more constrained angle space from which to sample conformations. Although it does rely on database fragments, ROSETTA follows the general *ab initio* paradigm of sampling, ranking, and refining candidates.

While *ab initio* methods such as ROSETTA work with any protein sequence, and can eventually generate very native-like structures, this often requires substantial amounts of computational power. In [16] Qian, Raman, et al. generated predictions for several CASP7 targets with very high accuracy, but this required approximately 100,000 hours of CPU time on IBM's Blue Gene cluster.

Template Based Protein Folding

A *template based* protein folding technique is one where the predicted structures depend wholly or in part on one or more *templates*, or probably-similar proteins with known tertiary structure.

In homology modeling, these templates come from homologous proteins (or at least those suspected of being homologous due to sequence similarity) and the predicted coordinates for the query protein come from the corresponding coordinates in the template(s), based on a (multiple) sequence alignment.

In *protein threading*, or *fold recognition*, the templates are those believed likely to have fold similar to the query protein, regardless of homology. The approach is derived from the observation that of the tens of thousands of proteins with experimentally determined structure, there are only a relatively few number of unique folds. As of 2013 the Protein Databank[10] contains 1393 unique folds as defined by SCOP[17] or 1313 as defined by CATH[18]. There have been no proteins with unique folds deposited since either 2008 by the SCOP definition, or 2009 by the CATH definition. So, while homology modeling is searching for homologous protein templates, threading is search-

ing for smaller, fold-level templates. The regions matched by fold recognition are then predicted based on these templates.

There is no hard line between the two approaches. Although the best homology modeling is done when there is a full domain template available, a protein domain can be predicted if there are several smaller homologous templates that can be used. The problem of assembling multiple templates into a single prediction is NP-complete[19], so it is much easier to work with a small number of large templates than it is to work with many small templates. Once a set of templates has been selected, a protein is modeled based on assembling the pieces and connecting them together using loop modeling. In two popular threading systems, HHPred[20] and RAPTOR[21]/RaptorX[22], this step is done by MODELLER[23].

1.2 Loop Modeling

With template based modeling, there are typically gaps that must be filled in. When threading, or when doing homology modeling with partial-domain templates, there will be regions that no templates cover, and these regions must then be filled in using an alternate method. Additionally, a sequence alignment can contain insertion-or-deletion (*indel*) events where either the query or the template sequence has more residues than the other. Atomic coordinates for residues involved in an indel cannot be derived from the template because there is not a one-to-one matching between the two sequences for these residues.

Indel events occur most often in the loop regions of proteins, as loops' flexibility and general lack of uniform structure allow them to accept such changes without substantially altering the overall structure of the protein. The divisions between templates also typically occur in the loop regions. For this reason, the task of filling in such gaps is referred to as *loop modeling*, although there is no strict requirement that these gaps are composed entirely of loop region (or indeed that they contain any actual loop regions at all).

Gaps can also occur in experimental protein models. X-ray experiments tend to have gaps corresponding to loop regions, for the same general reason. The flexibility and disorder of these regions result in multiple possible crystal conformations, yielding either no usable electron density map, or an electron

density map that results from averaging two very different conformations. This can make it difficult or impossible to make a model consistent with the density map, leading to a very poor R-Value for the loop.

In an NMR experiment, protein structure is modeled by examining the way in which a protein interacts with strong magnetic fields. The more *intrinsic disorder* there is in a region, the more noise that results when examining that particular region. Since loops tend to have more intrinsic disorder, loop NMR data tends to be very noisy.

1.2.1 Problem Definition

In the loop modeling problem, one is given a *query protein structure* with a *gap*. This gap is a region of consecutive residues where the protein structure is missing atomic coordinates, or where coordinates are given but a new model is desired (for example, an X-ray obtained loop with a poor R-Value).

The goal of loop modeling is to generate a *loop* in order to fill in the gap and obtain a protein structure model with no break in the backbone. One would prefer that this loop be *realistic*, meaning that it contains bond distances, bond angles, and torsion angles that are within the range typically seen in experimental data. For example, the majority of the models should fall within two standard deviations of the mean experimental values for these features, while very few should differ by more than four standard deviations. The residues immediately before and immediately after the gap are called the *stems* of the gap. The Euclidean distance between the exposed C atom in the first stem and the exposed N atom in the second stem is called the *span* of the gap, while the number of residues missing is the *length* of the gap.

Loop regions are the regions of the protein that cannot be classified as a well-defined secondary structure (helices and extended structures), and are highly flexible due to the lack of regular hydrogen bonds. This freedom of movement makes them hard to model compared to more rigid regions[24]. So, while the loop modeling problem is not strictly limited to the actual loop regions of a protein structure, in practice these are the regions where gaps are most likely to arise, and also where the corresponding loop is most difficult to model.

There are a number of ways that predictive protein models are compared to the native structure, each with strengths and weaknesses. However, not all of these are suitable for comparing accuracy in loop modeling, as the loops tend to be very short. A percentage similarity score, such as TM-score[25] or GDT[26, 27], is not meaningful for loops. However, the *root mean squared-deviation* (*RMSD*) score works well for comparing loops. For two sets of n points $X = \{x_1, x_2, \ldots x_n\}$ and $Y = \{y_1, y_2, \ldots y_n\}$, the RMSD of X and Y is defined as

$$\text{rmsd}(X, Y) = \sqrt{(\sum_{i=1}^{n} ||x_i - y_i||^2)/n} \tag{1.1}$$

In the context of proteins, the points are usually the alpha carbon atoms, the heavy backbone atoms, or all of the heavy atoms. Because protein models will not be in the same coordinate system, it is necessary to find the superposition of the two sets of atoms that will minimize the RMSD value. The minimized RMSD between two sets of points X and Y is defined as

$$\text{rmsd}_{\min}(X, Y) = \min_{R,T} \text{rmsd}(X, RYT) \tag{1.2}$$

Where R is a rotation matrix and T is a translation matrix.

When evaluating loop models, the native loop and the loop candidates will all be in the same coordinate system, so minimizing the RMSD is not necessary. However, the unminimized RMSD score will be sensitive to hinge action due to minor changes to the backbone. For this reason, both types of RMSD score are used when scoring the accuracy of loop modelers. The unminimized RMSD score is called the *global* RMSD, and the minimized RMSD score is called the *local* RMSD. Both local and global RMSD measure the similarity of shape between the two loops being compared, and global RMSD also measures the similarity of orientation. For the purposes of loop model evaluation, the RMSD is most often computed for the heavy backbone atoms (N,C_α,C,O). This is the atom set I used for my evaluations.

1.2.2 Motivation

Loop modeling is important to any kind of template based predictive modeling, as there are many gaps to be filled, particularly for threading. It is

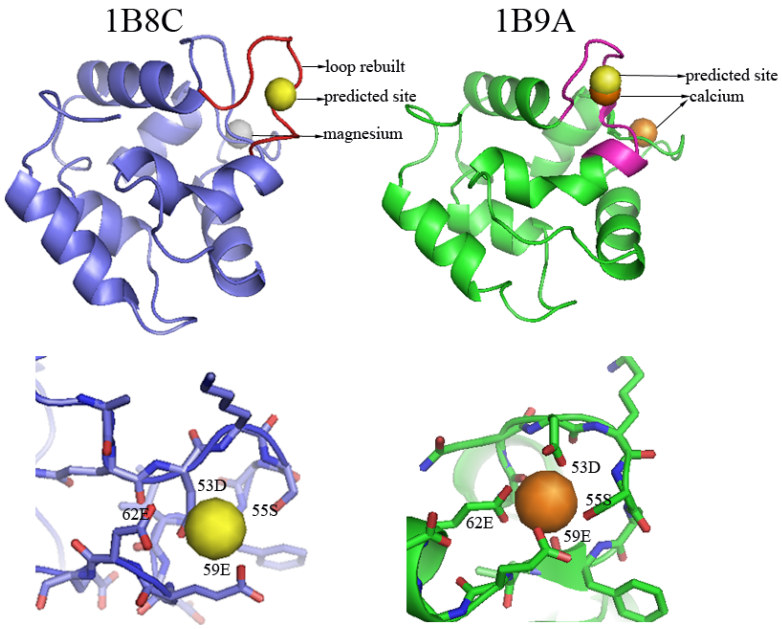
important that these loops be realistic, so that a reasonable attempt can be made at selecting between potential models. When using template based modeling, there are many potential alignments and templates to use, which will result in different models. In RaptorX[22] the highest ranked alignments generally yielded the best templates in terms of structural similarity, but the correlation is not perfect. While the best candidate is often in the top 10, it is not always the highest scoring candidate according to the template ranking. Selection between candidates is typically done using some form of energy potential function. Energy potential function values have limited physical meaning unless all of the requisite atoms (such as all heavy atoms, all heavy backbone atoms, or all atoms including hydrogens) are present in the model, so it is necessary to fill in any gaps with loop modeling. Accurate loop modeling will result in more accurate energy potential values, leading to more accurate candidate selection.

Loop modeling also has applications for protein function recognition. Structural similarity is often better conserved between proteins with similar function, so structural similarity is a good indicator of functional similarity. Such examinations focus on the shape and chemical properties of the molecular surface, or on 3D motifs associated with certain functions (such as binding sites or active sites)[28]. Loop regions are often exposed as part of the molecular surface, so they play an important role in a protein's function. When examining specific 3D structural motifs, loops also play several important roles. For example, in an enzyme protein, the most important structural feature is the *active site*, the location where molecules bind and undergo a chemical reaction catalyzed by the enzyme [29]. There are many examples of loops playing an important role in the active site. Loops can act as a stereochemical gate[30] for the active site, and they can play a part in the chemistry of the active site, either as part of the binding interaction (such as with phosphate binding loops[31]) or as part the post-binding chemistry[32]. Loops also play similar roles in the binding sites of receptor and ligand proteins. It is useful for comparative protein modeling to be able to reason about any changes to the chemistry or specificity of such proteins, and this requires accurate loop models[33]. In some cases, such as in a phosphorus binding loop (P-loop)[34], the loop is functionally important but well conserved in terms of sequence and 3D structure, and so likely to be identifiable by fold recognition. This is not always the case, and loop modeling can be helpful for identifying binding sites. In [3] Liu and Altman have used Wei's and Alt-

man's FEATURE[35] binding site recognition tool in combination with loop modeling in order to identify potential calcium binding sites in disordered regions. In Liu's and Altman's Figure 1, reproduced here in Figure 1.4, we can see that, while not completely accurate, modeled loops are sufficient in many cases to use FEATURE for identifying calcium binding sites.

Loop regions are often fairly disordered, meaning they take on multiple conformations and/or undergo a large degree of thermal motion. NMR models capture this disorder by having multiple loop models. X-Ray models sometimes also have multiple models, but more often capture this motion through the *B-value* or *temperature* of an atom. This is a function of the electron density map. This measure captures both the disorder of an atom and also the error in the model. It is difficult to separate the B-value into error and disorder components, it is not possible (without an additional source of data) to separate the B-value into components based on *static disorder*, where there are several stable conformations, and *dynamic disorder*, or thermal motion. Because of this, the B-value only partially represents the movements of loops.

One may question the validity of generating fixed loops that mimic the crystal conformations of loops from x-ray models, as these x-ray loops are averaged values. However, in the above examples (using an energy function to rank protein decoys, and using FEATURE to identify binding sites) the loops are given to functions or algorithms that have been trained on existing x-ray data. Thus, one could conclude that generating averaged loop coordinates is acceptable for the purposes loop modeling is currently used for.



"Parvalbumin-beta from cyprinus carpio is a calcium-binding protein and two structures 1B8C and 1B9A have been resolved experimentally. 1B9A binds to two calcium ions via loop 90-97 (residue 90-97) and loop 51-62 (pink, residues 51-62: AQDKSGFIEEDE). 1B8C binds to one magnesium ion via loop 90-97. The loop 51-62 in 1B8C does not bind any metal ions, thereby considered as apo form. The RMSD between 1B9A and 1B8C is 1.48 Å. The RMSD of loop 51-62 in 1B9A and 1B8C is 2.68 Å. By scanning loop 51-62, FEATURE successfully identifies the sites in 1B9A, but not in 1B8C. In 1B8C, FEATURE can identify the site only when the loop 51-62 is rebuilt using modeling methods. The close-up view shows (red for oxygen atoms) that the predicted site in 1B8C and the experimentally observed site in 1B9A are similar. Both sites are in close associations with oxygen atoms from four residues 53D, 55S, 59E and 62E. This example demonstrates that FEATURE can successfully identify calcium-binding sites in holo structures and in apo structures with binding loops rebuilt by modeling methods."

Figure 1.4: An example illustrating a successful FEATURE prediction with the help of modeling methods. Reproduced from [3] (Figure 1).

1.3 Previous Work

Solutions to the loop modeling problem can be broadly grouped into two classes. The first is the *statistical* technique, where loops are sampled using a statistical model. The second category is the *knowledge-based* or database technique, where loops are found by searching a database of known protein structures.

1.3.1 Statistical

Statistical loop modeling techniques, also called *de novo* or *ab initio* techniques, work by sampling loops from a statistical model and ensuring they fit properly into the gap. The statistical model is usually in the form of probability distributions for φ/ψ torsion angle pairs. Care must be taken to ensure that the loops generated are closed loops, meaning that they have realistic bond distances where the loop's backbone connects with the backbone of the two stems. There are many approaches to the sampling of loops, and the way they are made to fit into the gap properly.

ModLoop

Perhaps the most commonly used statistical tool (at the very least the most compared with) is ModLoop, the loop modeling tool that is part of the MODELLER package [23, 33]. ModLoop works by beginning with a trivial closed loop, created by placing the backbone atoms of the loop in a uniform line connecting the two stems. From this initial configuration, the loop is then randomized by adding a value sampled uniformly from between -5 and +5 Å to each coordinate. This randomized loop is then optimized and output. Optimization is done against their energy function, which is a combination of physics and statistical constraints and pseudo-energy terms. The optimization of the loop takes place in six phases. In the first, the random loop conformation is optimized by conjugate gradient minimization of the energy function. This initial phase is intended to relax the system and allow the randomly positioned atoms to pass close to each other without having to overcome large energy barriers. The system then undergoes simulated annealing, first being rapidly heated, then slowly cooled. The third step of the

optimization is another round of conjugate gradient minimization. In the first three steps, the energy function is calculated excluding any non-bonded atom pairs where one or both atoms are not part of the loop. That is, the energy function in these steps does not take the environment into account. The three steps are then repeated with non-loop atoms allowed in the energy function, so that the molecular dynamics respond to the local environment.

ModLoop's output is selected by energy function from several loop candidates created by independent applications of this generation and optimization procedure. The recommended number of independent candidates to generate is between 50-500. Going below this threshold yields poor results, and going above it results in diminishing marginal returns for the computational power invested.

Loopy

Another statistical loop modeling tool is the Loopy program[36], which generates loops by sampling torsion angle pairs, regardless of whether this results in loops that fit into the gap. The loops are adjusted to fit properly later in the algorithm. This generation is very rapid, allowing thousands of loops to be generated in a very short amount of time. From here the loops are closed using the *random tweak* method as described in [37]. The random tweak method takes randomly generated loop conformations and subjects their dihedral angles to an iterative linearized Lagrange multiplier procedure, which adjusts the loop to fill the gap with minimal perturbations. Once the closed candidates have been generated by random tweak, they are subject to energy minimization using the softened the van der Waals energy (Equation. 10 in [36]). This minimization is done using a fast torsional minimizer, which is unpublished. The 1000 (out of 2000) lowest energy loops are retained, and 30% of these are retained using the *colony energy function*[36] introduced by Xiang, Soto, and Honig. This function works by viewing the colony energy of a conformation as the weighted energy of all nearby conformations (estimated using the other conformations generated, which is why the plain energy function must be used until enough conformations have been sampled). The colony energy for a given conformation can be viewed as an estimate of the overall depth of the energy basin that the conformation lies in. In this way it represents the potential for a nearby low energy conformation, even if the current conformation is not particularly low energy. After filtering by colony

energy, loops which are similar but not exact (meaning they have an RMSD value of between L/10 and L, where L is the number of residues in the loop) are joined together to create new loops. The loops being fused are connected in the middle by using the random tweak method to satisfy the closure constraints with low perturbation to the angles involved. These new loops are then pooled with the old loops and the best 30% (with an upper bound of 300 candidates) are retained using the colony energy function. This procedure is repeated until the pool of conformations is reduced to the desired number of candidates (default 1) or until 5 rounds of iteration have passed.

LoopBuilder

LoopBuilder [38] is a protocol that is partially built upon the Loopy tool. LoopBuilder uses the Loopy tool to generate 1000 closed loops using the random tweak algorithm. However, rather than continuing with the Loopy algorithm, LoopBuilder adds sidechains using a modified version of the SCAP [39] algorithm and ranks them by their DFIRE[40] energy function. The 50 loops ranked highest by DFIRE energy are then subjected to minimization by PLOP[41]. This results in better loop predictions at the cost of taking longer in order to do the PLOP energy minimization.

RAPPER

The RAPPER tool [42] builds a fragment starting at one stem and working toward the other. These fragments are built iteratively in a round-robin queue, where in each round the fragments are extended in the N-to-C direction by sampling φ and ψ angles from residue-specific distributions. Fragments with backbone atoms that clash with the local environment are immediately discarded. To ensure closure, fragments are also discarded if the alpha carbon to alpha carbon distance between the current edge of the fragment and the C-stem is larger than the distance that can be spanned realistically by the remaining residues. Additionally, fragments are discarded if they have a global RMSD score of less than 0.2Å, so that the conformations sampled are diverse. Fragments are generated up to and including a *dummy* stem residue for the fragment to connect to. After a fragment has been generated, it is subjected to a final optimization step where randomly selected dihedral angles are varied in order to improve the overlap between the dummy

stem residue and the actual stem residue in the target protein. In their tests, de Bakker, DePristo, Burke, and Blundell generated 1000 such loop candidates, and used the SCWRL [43] program to pack the sidechains. Candidates are then ranked by RAPPER's RAPDF energy function. By using this early-terminating fragment building approach they are able to generate closed loops where all of the angles are close to their original sampled values.

ROSETTA CCD

ROSETTA's loop modeling tool samples from a sample space that has been reduced using fragment matches. These loops are closed using cyclic coordinate descent (CCD) [44]. CCD, like the random tweak method used in Loopy, is a method for taking a non-closed loop and ensuring it is closed. This algorithm has applications in robotics, animation, and other kinematics problems. In the robotics version, one has a robotic arm with multiple joints. The goal is to move the end of this robotic arm from its current position to a desired position in order to grab something. This can also be applied to protein loops, where the protein backbone has two joints per residue, the φ and ψ torsion angles (the ω angle is highly planar so it is usually not altered). The CCD algorithm is very simple. To start, the first joint is selected. Rotating about this joint will result in rotating the end point of the loop, and it is simple to calculate the rotation that will result in a minimum distance between the end point and the desired end point. CCD is the repeated setting of angles to minimize the end distance until this distance is below the desired threshold (in the case of loop modeling, this would be when there is a reasonable bond distance connecting the backbone between the loop and the stem). The "cyclic" in CCD's name comes from the fact that the default way to perform this algorithm is by cycling through the joints one at a time, and then starting over. This is a greedy algorithm so there is a tendency to make the largest changes to the first angles adjusted. CCD can have its moves filtered to avoid physically improbably angle assignments, but the algorithm must sometimes accept improbably torsion angles in order to successfully achieve closure.

Because of the potential for unrealistic conformations, ROSETTA's loop candidates are then optimized by simulated annealing. Angles are resampled, and closure is maintained by using CCD to close any gap re-introduced by this resampling. Moves are accepted and rejected based on the overall

change to the energy function by the resampling and requisite CCD closure adjustments. This is called the "perturb" phase, as the resampled angles result in large perturbations. After this is complete, the candidates are then refined. The procedure is broadly the same, but instead of large changes, the angles are changed only slightly toward reducing the candidates' overall energies.

ROSETTA KIC

The Kinematic Closure (KIC) algorithm, used as part of ROSETTA's loop modeling package [45], can be used to create closed loops from a set of sampled φ, ψ angles by altering only six of the angles. This is, all but six angles can be sampled statistically, while the other six must be set in order to maintain closure and may not be consistent with the statistical model. ROSETTA's general process of perturbation and refinement remains the same. The difference is that rather than perturbing or adjusting one angle at a time, all 2L-6 (where L is the number of residues involved) torsion angles can be adjusted simultaneously, and the KIC algorithm[46] is still able to maintain closure by setting the other 6 angles appropriately. This algorithm is inspired by similar problems in robotics, where there is interest in enumerating possible configurations for a robotic arm subject to constraints (such as that the shoulder is at a fixed position, and one wishes the robotic hand to reach a given target). The KIC algorithm begins with a closed loop, and selects 6 φ or ψ angles to be used as pivots. The loop closure constraints are formulated as a series of polynomials in these pivot angles. By taking the resultants of these polynomials, the KIC algorithm is able to sample all other torsion angles (and the τ bond angle) and determine all mechanically possible assignments for the 6 pivot angles that maintain closure for a given sampling.

As with CCD, the angles that results are not always realistic, but in this case there are only six angles in the resulting closed loop that are not equal to the statistically sampled values. Still, they are potentially unrealistic, and so ROSETTA uses the KIC algorithm in a similar way to how it uses the CCD algorithm. Pivots are selected at random, and the other angles are sampled. Monte Carlo simulation is used to accept or reject these changes based on the suitability of the pivot angles required to maintain closure. This can be repeated with different choices for pivots. Perturbation refinement can

also be easily done by setting angles to be similar to their existing values, rather than fully resampling, just as is done with ROSETTA's CCD loop perturbation step.

The ROSETTA KIC protocol shows much better average results for their test set as compared with their previous CCD protocol, as well as compared with molecular dynamics simulations. They also show a much tighter clustering for the sampled loops, indicating a more complete exploration of the conformation space around the local energy basins.

1.3.2 Knowledge Based

Another category of loop modeling techniques is the *knowledge based* or *database* category. Methods in this category work by finding existing loops that can be placed into the gap.

FREAD

An example of a knowledge-based loop modeler is FREAD [47], which recently has been reevaluated with newer data and improved methodology[48], giving improved accuracy at the cost of being less likely to locate a database template that passes the new filters. FREAD finds matches based on high sequence similarity, and filters these matches down to those that fit by requiring a very similar C_α to C_α span. Although this span does not capture information about the relative orientation of the two stems, the span and sequence similarity filters used in FREAD are very strict, so if any matches pass the filters, they will fit very well into the gap. For loops of 10 residues, FREAD is able to find matches in about one in ten test loops (a rule of thumb confirmed by my own use of FREAD) but if a prediction is made, it is almost always very accurate (having a local RMSD of less then 0.25Å).

So, while this approach is useful as a supplement to statistical approaches, it cannot be used as a stand-alone method for loop modeling.

LIP and SuperLooper

A second database for loop modeling is the LIP database[49], with which Michalsky, Goede, and Presner demonstrated that there are many cases

where a very close match from the database can be used to obtain a very accurate prediction. A more recent version of the LIP database is used by the SuperLooper server [50]. LIP and SuperLooper work by finding matches within a database of known protein structures. Matches are found by finding database segments of the requisite number of residues, and selecting those segments with a low $\mathrm{RMSD}_{\mathrm{stem}}$, defined as the minimized RMSD between the stem residues in the query protein, and the corresponding residues in the template.

If the $\mathrm{RMSD}_{\mathrm{stem}}$ is low enough, then not only must the database loop have very similar span, but it also must have similar orientation. Matches are ranked both by their $\mathrm{RMSD}_{\mathrm{stem}}$, and by sequence similarity.

Using this measure rather than the C_α to C_α span used by FREAD means that there are many more matches that can be used, since measuring the relative orientation of the stems lets the algorithm be less exact about the span while still guaranteeing the match will fit within the gap. However, computing $\mathrm{RMSD}_{\mathrm{stem}}$ for all potential template stems is very time consuming. LIP avoids this by indexing the loop database using a rounded two dimensional vector based on the stems of each loop. These indices are obtained by creating a coordinate system with an x-y plane defined by the C_α and C atoms from the starting stem, and the N atom from the ending stem. The distance between the C atom and N atom is then represented as a two dimensional vector in this plane. The x and y coordinates of this vector are rounded and the pair (x,y) is used as an index into the loop database.

The authors used a "goodness" measure defined in terms of this vector and two additional angles stored in the database, and show that this measure of goodness bounds $(\mathrm{RMSD}_{\mathrm{stem}})^2$ from above. For the authors' cutoff of 0.75Å only four database index pairs need to be read, as no loops in any other tables can have less than the desired $\mathrm{RMSD}_{\mathrm{stem}}$ cutoff value. Full RMSD computations are only done after matches are found and filtered by other measures. In this way the candidates can be ranked by their full stem RMSD values, but very few of these RMSD calculations need to be done.

SuperLooper results are much more common than FREAD results, because of the less strict filters. Nevertheless, there are still situations where there are either no matches in the database, or the matches that do exist do not fit into the local environment. Unlike FREAD, where any predictions that are made are almost always very good, LIP/SuperLooper often makes

poor predictions as well as good ones.

1.3.3 Comparative Advantages Between Approaches

Statistical methods have the advantage of always being able to produce a usable loop candidate if possible, while database methods have the advantage that if a very similar loop is represented somewhere in the PDB, it can likely be found and used. For this reason the two are typically viewed as complementary[48]. As Fiser et al. say in [33], statistical methods can be improved whenever there is new understanding of the physics of protein folding in general, or loop dynamics in particular, while database methods are improved by virtue of the exponentially expanding PDB. However, since typically database loops are optimized after selection, or at least ranked by an energy function, advances in protein physics do apply to both, if less so to a database approach.

One problem I have observed with database matches is that, even for fairly good results based on finding very similar stems, there will often be unrealistic torsion angles at the edges of the loop. Even a few bad angles can cause serious problems with many energy potential functions. This can be seen when comparing results from FREAD and SuperLooper. In the FREAD papers, which are consistent with my experience using FREAD, the tool typically produces a prediction for one out of every ten gaps. Although these predictions are almost always very accurate, one must rely on other techniques for most gaps. SuperLooper does not have FREAD's strict filters and so it can generate candidates for almost all cases. However, as the medium quality loops do not fit perfectly, there are many cases for which unrealistic torsion angles and peptide bond angles and distances result in energy scores that are worse than they should be given the quality of the loop prediction. The problem with database methods is then finding a good way to fit the loops into the gap in a very rapid manner. Time consuming refinement such as that available in the ROSETTA or MODELLER packages can improve the positioning and shape of a database loop candidate, but as there are thousands of potential matches, many of which may be poor, it is not time effective to do so without first narrowing the selection down.

My contribution is a way of rapid fitting of the loop to correct some of the angle issues. In this way, an energy function can be used to more accurately

rank the many database matches available, greatly reducing the number that must be sent through the more computationally expensive refinement process.

Chapter 2

LoopWeaver

2.1 Method Overview

I developed a new database based method, which I named LoopWeaver (Loop modeling by the WEighted scaling of VERified proteins). As a database approach, this technique begins by finding loops with similar stems by RMSD, and then places them into the gap. Initially it behaves much like SuperLooper, although I have found that the cost of computing $\mathrm{RMSD}_{\mathrm{stem}}$ is not so high as to necessitate precomputing large loop libraries. Instead, loops are first filtered by the C_α to C_α stem distances, as in FREAD, and $\mathrm{RMSD}_{\mathrm{stem}}$ is only calculated for those loops that pass this initial filter. This allows for rapid searches just as in LIP, but does not require large (multi-gigabyte) precomputed index files.

My contribution to the general database approach is an alternate method for placing the database loop into the gap. While a common approach is to use the transformation matrix that yields the minimum $\mathrm{RMSD}_{\mathrm{stem}}$ to place the loop into the query protein, I formulated the problem as an instance of the Weighted Multi-Dimensional Scaling (WMDS) problem [51], and solved it using an established heuristic (the SMACOF algorithm[51]). This improved the orientation of the loop by reducing the unreasonable angles at the edges, which means that energy functions were better able to rank the results. Additionally, database methods can often introduce what is known as a *chain-break*, where a backbone bond (typically the peptide bond) is many standard deviations from the mean value for the bond's type. While

differences of 4 or more standard deviations are sometimes observed experimentally, they are outliers and should be quite rare, so it is not appropriate for a loop modeler to create such situations except as a rare case.

On a 2.2 GHz Opteron, it took LoopWeaver roughly 5 minutes to find and close 500 database matches for a length 10 loop. LoopWeaver uses the DFIRE [40] energy potential function to rank the final loop candidates. As this is an all-atom potential function, the results must have accurate side-chains. Side-chains were built using the TreePack tool [52, 53]. This added an average of 15 minutes to the running time. Side-chain packing requires repacking all sidechains that are near the loop, not just those that are part of the loop , so the time required for this step depends more on the environment than it does on the length of the loop. In the test sets used, the minimum energy candidate after sidechain packing was rarely below the top 150 prior to sidechain packing so the time spent packing can likely be greatly improved by filtering many of the candidates prior to sidechain packing without impacting the accuracy except in a few extreme cases. The best candidate was then refined using the KIC refinement protocol included in ROSETTA 3.3.

Figure 2.1 gives an overview of the LoopWeaver protocol in flowchart form. Note that the second phase (clash avoidance) is optional and only useful for cases where there are few or no non-clashing database matches.

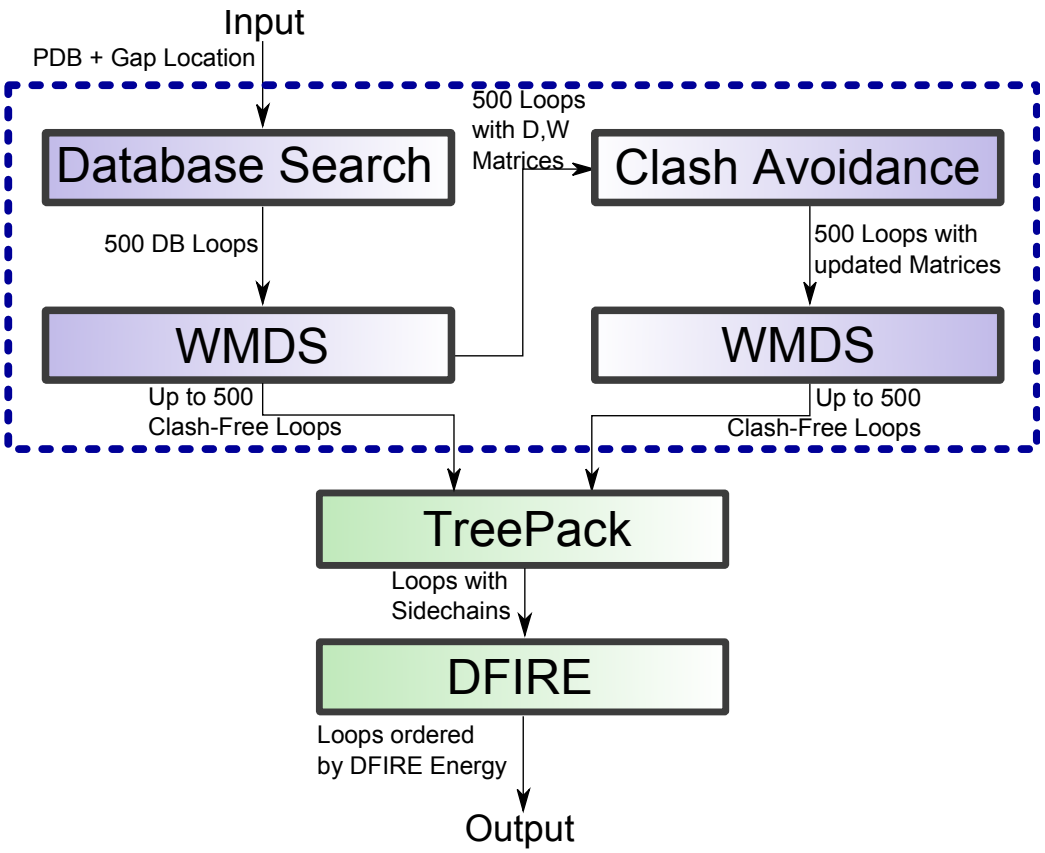


Figure 2.1: LoopWeaver Flowchart. The two-part clash avoidance step on the right hand side is optional.

2.2 Method Details

This section details the algorithms and tables used by the LoopWeaver program. In the pseudocode, array access uses Python-like notation. For example, L[1:-1] selects all but the first and last value, and L[-1] indexes the last element.

2.2.1 Database Matches

The LoopWeaver tests used a database comprising roughly 14,400 protein chains, selected using PISCES [54] with the cutoff values being 3.0Å resolution, 90 percent identity, and 1.0 R value. Because the test sets include targets from CASP8, PISCES was only allowed to select from proteins with a release date prior to the start of CASP8 (May of 2008). This database was used for all tests, even those involving targets from CASP9 rather than CASP8.

Like any database approach, LoopWeaver begins by searching a database of known protein structures, and locating appropriately spaced residues that are similar to the stems of the target gap. The metric used for determining the similarity between a match and the stems is $\text{RMSD}_{\text{stem}}$. This is defined as the minimum RMSD between the N, C_α, and C atoms in the stems, and the N, C_α, and C atoms in the corresponding residues of the database match. This metric is the same as used in LIP/SuperLooper. Unlike LIP, LoopWeaver does not index its individual protein files but instead compacts these 14,400 individual PDB files into a single binary file, allowing each chain to be loaded in only a few read operations. This allows the entire database to be read and searched in under 10 seconds on a standard desktop computer (2GHz Athlon64), as opposed to taking 10-15 minutes to load all 14,400 files individually. Loading many individual files has a high OS overhead, in addition to the processing required to convert text formatted numbers to a computer usable representation. This also means that the database can be fit into a 200MB file rather than the 6GB required for LIP's indexed tables. To avoid excessive $\text{RMSD}_{\text{stem}}$ calculations, the RMSD is only computed if the C_α to C_α distance of a database match is within 20% of the equivalent distance in the target protein. This is a very loose filter, but its purpose is only to decrease the number of RMSD calculations required to rank the

database matches. With this filter is takes only a few seconds to search the entire database and rank the potential loop templates according to their stem RMSD scores.

Algorithm 1 shows the pseudocode for scoring a database match.

Algorithm 1 Match Scoring

```
function MakeScoredMatch(Atom P[],Atom C[])
                                  ▷ P is the input protein, C is the template
    GapStems = P[:4] + P[-4:]
    DBStems = C[:4] + C[-4:]
    if Cα DistDiff(GapStems,DBStems) < 0.20 then
        Score,T = RMSDmin(GapStems,DBStems)
        L = new Loop
        L.score = Score
        L.dbAtoms = T × C          ▷ apply transformation T to the atoms.
return L
    elsereturn FALSE
    end if
end function
```

Once LoopWeaver has obtained this large list, it sorts the list according to $RMSD_{stem}$, and takes the top 500 matches. Where SuperLooper takes only matches that have a very low stem RMSD (otherwise the match will not fit well enough to form bonds with the rest of the backbone) LoopWeaver spends more time fitting the loop so as to resolve these placement issues, so it does not require a strict filter.

2.2.2 Fitting the Loop

Once suitable loop candidates have been selected from the database, they must be placed into the gap in order to connect to the protein backbone correctly. Because LoopWeaver selects matches based on the stem RMSD, this can be done by taking the transformation matrix that yielded the smallest stem RMSD, and applying this transformation to the database loop. For matches with a very low stem RMSD, this transformation will always fill the gap without having unreasonable bond lengths. LoopWeaver is able to use

matches with a higher stem RMSD by using a more complicated technique to place the loop, rather than using a rigid transformation.

One can view the placement of the loop into the gap as an attempt to satisfy two contradictory requirements. The first requirement is that the stems remain the same. The second requirement is that the loop being placed should be the same shape as the database match, including the stem region of the database match. Unless the database stems are identical to the query stems, both constraints cannot be simultaneously satisfied. The simple loop placement approach does not alter the stem residues, and loop angles and distances are not altered from their database values except where the loop connects to the stems. Although the stems are similar to the corresponding database residue, minor differences can have a large impact on the orientation of the loop. So, the goal is to satisfy these two sets of requirements in a more balanced way. This will hopefully not only improve the orientation of the loop, but also resolve any unrealistic bond lengths caused by a less restrictive $\text{RMSD}_{\text{stem}}$ cutoff value.

I chose to solve these requirements by formulating them as an instance of *weighted multi-dimensional scaling* (WMDS) as described in [51]. WMDS is a problem often used in statistics, as its namesake purpose is to take high dimensional data and transform it to 2 or 3 dimensional data while maintaining the scale. This is convenient for taking high dimensional data and rendering it visually without introducing too many inaccuracies. It has also been used in MUFOLD [55] as a method for assembling protein fragments.

A WMDS problem involves a collection of n objects, for which a distance function can be defined. Then one can create a matrix of desired distances

$$D = \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \cdots & \delta_{n,n} \end{pmatrix} \tag{2.1}$$

Where $\delta_{i,j}$ represents the distance between the i^{th} point and the j^{th} point. One also creates a weight matrix

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{pmatrix} \tag{2.2}$$

where $w_{i,j}$ is the weight, or confidence, for the desired distance $\delta_{i,j}$.

The objective is to find a set $X = \{x_1, x_2, \ldots, x_n\}$, where $x_i \in \mathbb{R}^d$ for a given dimension d. that will minimize the *stress*, defined as

$$\sigma(X) = \sum_{0<i<j\leq n} w_{i,j}(||x_i - x_j|| - \delta_{i,j})^2 \tag{2.3}$$

Formulating the Problem

For the loop fitting problem, $d = 3$ and n is the total number of heavy backbone atoms in the both the loop and the stems. So, for a loop of length l, $n = 4(l+2)$. The set of atomic coordinates from the protein being modeled is $P = \{p_1, p_2, \ldots p_n\}$, where the atoms are given in the N-C_α-C-O ordering. That is, p_1 corresponds to the N atom in the first stem, and p_n corresponds to the O atom of the last stem. The set of atomic coordinates from the database loop is $C = \{c_1, c_2, \ldots c_2\}$, with the same numbering as the set P. The set of stem atoms under this numbering is $S = \{1, 2, 3, 4, n-3, n-2, n-1, n\}$. Note that p_i is undefined unless $i \in S$ as these atoms are within the gap, are not present in the input.

With these definitions, we can create the matrix D using distances from the set P when available, and the set C otherwise. That is, D is defined by the equation

$$d_{i,j} = \begin{cases} ||p_i - p_j|| & i \in S \text{ and } j \in S \\ ||c_i - c_j|| & \text{otherwise} \end{cases} \tag{2.4}$$

Algorithm 2 gives the pseudocode for generating the matrix D from the database match C and the query protein P.

I used the weighted version of this problem because not all of the desired distances are equally important. Primarily, I did not want to make any changes to the stem atoms, so the pairwise weights between two stem atoms should be very large. Beyond this, atoms that are closer together should be

Algorithm 2 Building the Matrix D

```
function BuildDistanceMatrix(Atom P[] ,Atom C[],int N,Set S)
    D = Matrix(N,N)
    for i = 1 → N-1 do
        for j = i+1 → N do
            if i ∈ S AND j ∈ S then
                D[i,j] = distance(P[i],P[j]
            else
                D[i,j] = distance(C[i],C[j])
            end if
        end for
    end for
    return D
end function
```

given higher weights. If two atoms are within a few angstroms of each other within the loop, then they should remain close to this distance regardless of other changes made during loop placement. On the other hand, atoms that are far apart are relatively free to be moved without changing the overall shape of the loop (ignoring the effects of that movement on any other atoms). The recommended weight for use with WMDS is either d^{-1} or d^{-2}, where d is the corresponding distance. A value of 2 is recommended [56] if one wants more emphasis on close points than on distant points, which is the case here. I tested on a small number of length 8 loops, and tried exponents ranging from -1 to -3 in increments of 0.1. The best results were achieved at -2.0.

I observed that this weight assignment was often making large changes to the global RMSD of the loops, because the orientations were sometimes very different after WMDS. I speculated that weights for very distant atoms should be somewhat high, rather than very low, as the distances between the most distant pairs of atoms characterize the shape and path of the loop. I determined a useful weighting for the distances empirically, by selecting 10 target loops from the database, and for each target loop, finding the matching loops with RMSD less than 1.5Å. That is, loops that are fairly similar in shape, though not identical. Then I created bins with a width of 0.5Å, and for every pair of atoms within the loop, populated the corresponding bin with the squared deviation of all corresponding pairs of atoms. For example, if a pair of atoms was 9.4Å apart in the target loop, and in a similar database

$$\begin{array}{l} \text{N} \\ \text{C}_\alpha \\ \text{C} \\ \text{O} \end{array} \begin{pmatrix} 32 & 5 & 6 & 32 \\ 32 & 16 & 14 & 54 \\ 32 & 40 & 13 & 24 \\ 10 & 2 & 1 & 0.5 \end{pmatrix}$$

Figure 2.2: Matrix T used for short-distance weights.
Row corresponds to the type of the first atom. Column corresponds to how many atoms ahead the second atom is, in the same ordering. So, row 2 column 3 corresponds to the distance between a C_α atom and the N atom in the next residue.

match the same two atoms were 10.1Å apart, I would add $(9.4-10.1)^2$ to the bin for the 9.25Å to 9.5Å range. I then examined the standard deviations of the values within each bin, and found that could be approximated by the equation $k(\min\{d_{i,j}, r - 0.6d_{i,j}\})^2$, where r is the radius of the loop (the largest pairwise distance within the loop) and k is a constant that changes from loop to loop. I chose to use the reciprocal of this equation as the weights for the WMDS formulation so that the weight for any distance will be equal to the reciprocal of the anticipated standard deviation, and c consequently a difference of one standard deviation will result in a constant penalty to the objective function.

To ensure that bond distances, bond angles, and torsion angles were being maintained, I overrode these weights for atoms that are in the same residue, or in adjacent residues in the backbone. The weights for these short distances are also derived from the standard deviations observed in database loops. The table of these weights is called T, and is shown in Figure 2.2. Each row of this table corresponds to one of the four backbone atoms (N,C_α,C,O) and each column corresponds to a second atom further along in the backbone. For example, row 2 corresponds to a C_α atom, and column 3 corresponds to atoms 3 ahead in the backbone. The cell at row 2, column 3 contains the weight used for the distance between a C_α atom and the N atom in the next residue.

Following these rules, the defining function of the matrix W is

$$w_{i,j} = \begin{cases} 10,000 & i,j \in S \\ T((i-1) \pmod 4 + 1, j-i) & j - i \leq 4 \\ (\min\{d_{i,j}, r - 0.6d_{i,j}\})^{-2} & \text{otherwise} \end{cases} \tag{2.5}$$

Note that the value $\delta_{i,i}$ is always 0 for all i, and as such $w_{i,i}$ does not need to be defined.

Algorithm 3 shows the pseudocode for creating the matrix W from the matrix D.

Algorithm 3 Building the W Matrix

```
function BuildWeightMatrix(Matrix D,int N,Set S)
    maxDist = max(D)
    W = Matrix(N,N)
    for i = 1 → N-1 do
        for j = i+1 → N do
            if i ∈ S AND j ∈ S then
                W[i,j] = 10,000
            else if j − i ≤ 4 then
                W[i,j] = T((i-1) (mod 4)+1,j-i)
            else
                W[i,j] = pow(min(D[i,j],maxDist - 0.6× D[i,j]),-2)
            end if
        end for
    end for
    return W
end function
```

Solving the Problem

I used the *SMACOF* (Scaling by Majorizing a Convex Function) algorithm [51] for solving the WMDS problem. The SMACOF algorithm works, as the name suggests, by majorizing the stress function with a function that is easy to minimize. The stress function is expanded as follows

$$\sigma(X) = \sum w_{ij}(||x_i - x_j|| - \delta_{ij})^2 \tag{2.6}$$

$$= \sum w_{ij}\delta_{ij}^2 + \sum w_{ij}||x_i - x_j||^2 - 2\sum w_{ij}\delta_{ij}||x_i - x_j|| \tag{2.7}$$

$$= K + \text{tr } X'VX - 2\text{tr } X'B(X)X \tag{2.8}$$

With $B(X)$ defined as

$$b_{ij} = \begin{cases} -(w_{ij}\delta_{ij}/||x_i - x_j||) & \text{if } i \neq j \\ \sum_{k \neq i}(w_{ik}\delta_{ik}/||x_i - x_j||) & \text{if } i = j \end{cases} \tag{2.9}$$

And V defined as

$$v_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{k \neq i} w_{ik} & \text{if } i = j \end{cases} \tag{2.10}$$

The first term K is a constant as it only involves D and W, and the second term is quadratic in X as the matrix V does not depend on X. The third term however is more difficult, so the stress function cannot be minimized easily.

The SMACOF algorithm works by majorizing this third term (and thus majorizing the stress overall). Majorizing is a term from convex analysis which refers to creating a function that bounds another function from above, and touches it at exactly one point (called the support point). The stress function can be majorized with support point Z by a pseudo-stress function defined as

$$\gamma(X, Z) = K + \text{tr } X'VX - 2 \text{ tr } X'B(Z)Z \tag{2.11}$$

$\gamma(X, X) = \sigma(X)$, and for all X other than Z, $\gamma(X, Z) > \sigma(X)$. As Z is a constant matrix, the function $\gamma(X, Z)$ is a quadratic in X. Any quadratic of the form $f(x) = ax^2 + bx + c$ has a minima (or maxima) at $x = -\frac{b}{2a}$, so the γ function has a minima at

$$X = -\frac{-2B(Z)Z}{2V} \tag{2.12}$$

$$= V^{-1}B(Z)Z \tag{2.13}$$

As V is not always invertible, the pseudo-inverse is used instead. Because the pseudo-stress function majorizes the stress function, then with a solution $X \neq Z$ that minimizes $\gamma(X, Z)$, it must be the case that $\sigma(X) < \sigma(Z)$, since $\sigma(X) < \gamma(X, Z) < \gamma(Z, Z) = \sigma(Z)$. That is, if one starts with an initial set of coordinates Z, minimizing γ results in a new set of coordinates that must have lower stress.

The SMACOF algorithm is then to start with X_0 being an initial solution, and to repeatedly set $X_i = \min_X \gamma(X, X_{i-1})$ until $\sigma(X_{i-1}) - \sigma(X_i) < \epsilon$ for

some desired cutoff point ϵ. This deterministic algorithm has been shown to decrease σ monotonically, and is very fast. This algorithm gives a good trade-off between speed and accuracy [57], particularly if the initial solution X_0 is reasonable to begin with, as it is in this case. It has the added benefit of being very simple and easy to program as opposed to genetic algorithms or simulated annealing. As LoopWeaver begins with fairly good solutions already (indeed solutions that are typically good enough to use "as-is" in other database loop modelers) and speed is a major consideration due to the sheer number of database matches to be dealt with, SMACOF is the best choice.

The Pseudocode for SMACOF can be found in Algorithm 4.

Algorithm 4 SMACOF Algorithm

```
function SMACOF(Matrix D,Matrix W,Point X[])
    V = CalcV(D,W)
    V+ = PseudoInverse(V)
    oldstress = ∞
    newstress = σ(D,W,X)
    while (oldstress - newstress) > ε do
        oldstress = newstress
        X = V+ * B(D,W,X) * X
        newstress = σ(D,W,X)
    end while
    return X
end function
```

Computing the pseudo-inverse is a cubic running time operation, and the remainder of the algorithm is quadratic per iteration. For typical loop lengths this is very fast.

2.2.3 Clashes

One issue is that the problem formulation does not address collisions. The WMDS formulation is based around desired distances, rather than inequalities, so it is not simple to add clash terms that ensure the solution avoids the local environment.

The solution I adopted for resolving clashes was to keep track of all solutions that clash with the rest of the protein. Here, clash is defined as an being within 2.4Å of one in another residue that is separated by at least two residues in the backbone. After using SMACOF to close the database matches, LoopWeaver has a list of residues that contain one or more atoms that clashed with the database matches. For each of these residues, it evaluates the pairwise DFIRE energy potential between the clashing residues and the loop candidate residues, and keeps only those clashing residues whose average contribution to the energy potential is positive over all of the loop candidates. After discarding these residues, LoopWeaver is left with residues that clash with, or at least are unfavourably close to, many of the candidates, rather than residues that clashed with only a few database matches. The C_α atoms from the central residue of each remaining clashing residue (if any) are then included in the WMDS equations.

Algorithm 5 gives the pseudocode for obtaining the list of problematic residues.

As these atoms are from the input structure, they are fixed just like the stem atoms, and so treated in the same fashion. The desired distance d between clash atom a and loop atom b is computed by taking the average of $d_{a,b}$ for all loop candidates where the total DFIRE potential between a and the loop is negative (more favourable), as long as the candidate did not clash with a. If the candidate clashed with another atom it is still included in the average, so that these distances can be computed even if there were no clash free candidates. The weights for the new distances are set to the desired distance to the power of negative 2.

Algorithm 6 shows the pseudocode for adding the additional atoms into the D and W matrices.

Algorithm 5 Finding Problematic Residues

```
function FindProblemRes(Protein P,int Start,int End,Loop Loops[])
    Set ClashRes = []
    for all Residue R ∈ P except indices Start-2 to End+2 do
        for all Loop L ∈ Loops do
            if Clash(R,L) then ClashRes.add(R)
            end if
        end for
    end for
    for all Residue R ∈ ClashRes do
        ScoreSum[R] = 0.0
        for all Loop L ∈ Loops do
            for all Atom A ∈ L.Solution do
                ScoreSum[R] += DFIRE(R,A)
            end for
        end for
    end for
    FinalClashRes = []
    for all Residue R ∈ ClashRes do
        if ScoreSum[R] > 0 then FinalClashRes.add(R)
        end if
    end for
    return FinalClashRes
end function
```

Algorithm 6 LoopWeaver Clash Avoidance Matrix Building

```
function AugmentWMDSW(int N,Residue ProblemRes[],Loop Loops[])
    M = Size(ProblemRes)
    NewD = Matrix(M,N)
    NewW = Matrix(M,N)
    for I = 1 to M do
        Residue R = ProblemRes[I]
        Atom A = R.Cα
        for J = 1 to N do
            Counter = 0
            for all Loop L ∈ Loops do
                Atom B = L.FittedLoop[J]
                if NoClash(R,B) and DFIRE(A,B) < 0 then
                    Counter++
                    NewD[I,J] += Distance(A,B)
                end if
            end for
            NewD[I,J] = NewD[I,J] / Counter
        end for
    end for
    for all Loop L ∈ Loops do
        L.D.Resize(N+M,N+M)
        L.W.Resize(N+M,N+M)
        for I = 1 to N+M do
            for J = 1 to M do
                if I is a stem atom or I > N then
                    L.D[I,N+J] = L.D[N+J,I] = NativeDist(I,J)
                    L.W[I,N+J] = L.W[N+J,I] = 10,000
                else
                    L.D[I,N+J] = L.D[N+J,I] = NewD[I-N,J]
                    L.W[I,N+J] = L.W[N+J,I] = 1.0 / (NewD[I-N,J])^2
                end if
            end for
        end for
    end for
end function
```

2.2.4 LoopWeaver Pseudocode

Algorithm 7 shows the pseudocode for the LoopWeaver algorithm, which uses the pseudocode functions defined previously.

Algorithm 7 LoopWeaver Main

```
function LoopWeaver(Protein Query,int Start,int End)
    L = End-Start+1, N = 2 * (L + 2)
    S = {1,2,3,4,N-3,N-2,N-1,N}
    Gap = BackboneAtoms(Query.Residues[Start-1:End+1])
    loops = []
    for all Chains C ∈ Database do
        for all Length L+2 segments R ∈ C do
            dbLoop = MakeScoredMatch(Gap,R)
            if dbLoop then loops.insert(dbLoop)
            end if
        end for
    end for
    loops.SortByStemRMSD()
    for all Loop L ∈ loops[:500] do
        L.D = BuildDistanceMatrix(Gap,L.dbAtoms,N,S)
        L.W = BuildWeightMatrix(L.D,N,S)
        L.Solution = SMACOF(L.D,L.W,L.dbAtoms)
    end for
    if FixClashes then
        ProblemAtoms = FindProblemAtoms(loops)
        AugmentWMDSM(N,ProblemAtoms,newLoops)
        for all Loop L ∈ loops do
            L.Solution2 = SMACOF(L.D,L.W,L.Solution)
        end for
    end if
    OutputClashfreeLoops(loops)
end function
```

2.2.5 Ranking and Selection

After obtaining closed loops through solving the WMDS instances, LoopWeaver must determine the most likely loop to fill the target gap. I selected the DFIRE (Distance-scale Finite Ideal-gas Reference)[40] energy function for picking a candidate, as others have had good results using this function to rank loop candidates [58, 47]. Specifically I used DFIRE with an updated energy table, as used in the LoopBuilder protocol[38].

The DFIRE energy potential function is an empirical log-odds score. The DFIRE energy table is populated by observing actual inter-atomic distances, and the energy function is calculated by taking the log of the probability of having two atoms at a given distance (based on the table) minus the log of the probability of seeing two atoms at that distance in a finite ideal gas.

DFIRE is an all-heavy-atom energy function, so it achieves the best results when all sidechains are present in the model. It is reasonably effective at making selections when the loop has only backbone and C_β atoms for the loop, though it is better if all atoms are available. In part this is because without sidechain atoms, the DFIRE cannot function "notice" that some residues may have steric clashes other residues. For that reason it is necessary to add sidechains. Database sidechains cannot be used as is, unless the database loop has the same sequence, the sidechains cannot be transferred between the two. For that reason only the C_β atom was retained (unless either query or database loop has a glycine in which case there is no such atom, or unless one or the other but not both is a proline in which case the C_β was discarded.

Sidechains were added using the TreePack program [52, 53]. TreePack formulates the sidechain packing problem as a *residue interaction graph*, where each residue is a node in the graph, and an edge exists between two nodes if there is any combination of rotamers for these two residues that results in conflicting atoms. For each edge there is a pairwise interaction score, based on the SCWRL 3.0 paper [43]. Any node i with degree 1 and neighbor j can be removed from the graph, as for all possible rotamers for j, one can compute the optimal rotamer for node i. This allows the energies for these combinations to be summed together and node i to be merged into node j without impacting the optimal energy computation. As with other approaches, TreePack works by decomposing the full graph into overlapping connected components.

If a component A overlaps with only one other component B, then one can generate a table of optimal rotamer assignments for all possible rotamer assignments to $A \cap B$, and A can then be removed. This process can be repeated until only only component Z remains. Z can then have its optimal assignment determined exhaustively. All components that intersected Z will then have their own optimal assignments determined according to the previously generated tables. This can be repeated until all components have their optimal assignments.

TreePack generates the set of connected components by way of tree decomposition. Generating a decomposition of minimal width is NP-Hard, but because the graphs involved are geometric, it is possible to generate a reasonable decomposition quickly. Thus, TreePack is able to generate optimal assignment for the SCWRL 3.0 energy function in much less time. While the running time for this algorithm is exponential in the size of the largest component, these are typically small. TreePack is on average five times as fast as SCWRL 3.0.

When I examined both tools I found this to remain true when dealing with short loops rather than full proteins. TreePack is able to add sidechains to 1000 loop candidates in under 5 minutes, while SCWRL takes an hour or more. There were very few differences between the final results other than the time required to obtain them.

Chapter 3

Results

In this Chapter I present LoopWeaver's average RMSD scores, along with the scores of several popular loop modeling tools. All RMSD scores were calculated using the heavy backbone atoms (N, C_α, C, and O).

3.1 Test Sets

As LoopWeaver is a database driven tool, it could not be benchmarked using the same target proteins used in older loop modeling papers. Doing so would either put the database tool at substantial advantage by using a current database (because even if the exact matches are excluded, there still may be many similar matches in the database), or substantial disadvantage by using a database from the same date as the test set, which negates the advantage of rapidly expanding coverage in the PDB. Therefore, I tested the tool against others using more recent test sets. Specifically, I selected all loop regions (identified by DSSP [6]) of length 6 through 11 from the x-ray targets presented at the CASP8 and CASP9 experiments, while excluding NMR models. The target proteins' native structures were obtained from Zhang Lab at the University of Michigan (http://zhanglab.ccmb.med.umich.edu/). To ensure fairness, the database consisted of only protein structures released to the PDB prior to the start of the CASP8 experiments. Section 2.2 (Method Details) contains the specifics of the database composition and selection.

Although the CASP targets usually have full domain template matches available (since the purpose of the CASP experiments is to test template

based protein modeling) they are selected by hand to be difficult. Here difficult means that although there is a full domain template match, it is not identical to the actual tertiary structure. Since most of these deviations occur in the flexible loop regions, few of the test loops can be modeled using the full domain template match.

Because short loops are the most commonly observed loops, there were too few loops of length 12 or longer to justify inclusion. Loops of 5 or fewer residues are not included as no tools have difficulty modeling such loops. Length 1-5 loops have only a small number of possible conformation types, which simplifies sampling and results in them being well represented in databases of known structures, making their prediction simple for both categories of loop modeling approaches. Loops of 3 or fewer residues have no degrees of freedom left after fixing their endpoints (if the ω torsion angle, the bond distances, and the bond angles are assumed to be their average values) and so have exactly one solution.

3.2 Details of Other Methods Used

I compared LoopWeaver's results to those of several top loop modeling applications. The approaches used by these methods are described in Section 1.3 (Previous Work). Here I describe the specific parameters used in order to test these methods.

3.2.1 ModLoop

ModLoop, the loop modeler from the MODELLER package (version 9v5), was run with refinement set to "fast" and used to generate 50 loops. The candidates returned from ModLoop were then re-ranked according to their DFIRE [40] energy potential, which improves the accuracy when compared to the default DOPE energy function. This is consistent with others' results, where the best MODELLER results have been obtained after re-ranking by DFIRE.

3.2.2 RAPPER

When evaluating RAPPER version 0.5, I generated 1000 candidates as described in the RAPPER paper[42]. As with MODELLER, the RAPPER results were re-ranked by the DFIRE energy potential, since this substantially improves their accuracy for all test sets when compared with the default RAPDF energy function.

3.2.3 Loopy

I also compared with the Loopy program (no version number) with all settings left to default, and with the number of initial models set to 2000 for loops shorter than length 10, and 4000 otherwise. These are the recommended number of candidates for loops of these lengths[36]. While RAPPER and MODELLER take an average of 3 hours (on a 2.2 GHz Opteron) per length 9 loop modeled, Loopy takes around 20 minutes. Loops were removed from the input file and their sequence passed using the -r parameter. This is because Loopy will use the backbone atoms up to C_α from the first and last residue of the loop, if they are present, giving it essentially two half-residues less to model. Or, alternately, it can be viewed as remodeling the gap-facing backbone atoms of the stems, in which case it is solving a harder problem.

LoopyMod/LoopBuilder

Additionally, I compared LoopWeaver with the LoopyMod (no version number) variant of Loopy included in LoopBuilder package[38]. This version generates loops and does quick torsional optimization but does not have the iterative merging procedure used in the full version of Loopy. This allows the user to use an alternate refinement approach while still using Loopy's fast loop sampling. I used the same parameters as described in the LoopBuilder, 1000 candidates for loops shorter than 9 residues, 2000 for length 9 loops, and 5000 for lengths 10 and 11. LoopyMod has the same problem as Loopy with regards to retaining some atomic coordinates from the loop being remodeled, and unlike Loopy this cannot be avoided by removing them as LoopyMod requires these atoms to be present. For this reason I passed LoopyMod different start and end coordinates so that it remodeled the stems,

rather than keep part of the native loop structure (which should not be available). This puts it at a disadvantage, but I compared its results with those of Loopy proper to show that Loopy and LoopyMod perform similarly after being passed to ROSETTA's loop refiner and so it is acceptable to only deal with the Loopy results.

3.2.4 ROSETTA

Finally, I tested version 3.3 of ROSETTA [45]. This version uses the KIC (KInematic Closure) algorithm to generate closed loops where all but 6 torsion angles can be exactly equal to the statistically sampled angle. ROSETTA is often excluded from benchmarks, as although it produces very accurate predictions, the Monte Carlo simulation required to do so is very time consuming. For example, while it may take 20 minutes for Loopy to generate a prediction for a given loop, ROSETTA may take 5 days to model the same loop on the same CPU (if one uses the recommended settings). If the number of candidates generated is lowered drastically, ROSETTA can complete in time comparable with other tools, but is no longer accurate, at least when using the CCD algorithm for loop closure. However, while the new KIC loop closing algorithm is slower than the CCD algorithm used previously, the ROSETTA team's publications show a much tighter clustering of results around the native loop conformations. This suggests that the tool should be able to generate an acceptably accurate prediction with much less total coverage required, so I tested ROSETTA set to generate 10 candidates (so that its running time remains on the order of 2-3 hours, comparable to the running time of MODELLER and RAPPER). The ROSETTA KIC tests used the parameters as described in their online guide [59], and generated 10 candidate structures rather than the recommended 1,000-10,000. Additionally, I used the "-loops::fix_natsc" flag to prevent ROSETTA from refolding the native sidechains. By default ROSETTA will refold any native sidechains that lie within a certain distance (14 angstroms by default) of the loop candidate it generates. This is a reasonable step for a predicted model as these sidechains would have been packed without considering the loop, but other tools do not do this so it would result in ROSETTA being at a disadvantage by solving a harder problem. Not using this option resulted in less accurate predictions, as expected.

ROSETTA is also useful for refining the results of other tools. ROSETTA

can perform the same refinement steps it uses for its own candidates on loops that have been generated by another program. This is still a substantial amount of the total running time, since ROSETTA's loop closure technique is fairly fast. As in the LoopBuilder protocol, the results from one or more tools are refined using Monte Carlo simulation. While LoopBuilder uses PLOP[41], I elected to use ROSETTA's KIC refinement protocol instead. Refinement was done by refining the top candidate 10 times, rather generating one refined loop from each of the top 10 unrefined candidates. Many times this refinement can make the candidate worse, so it is best to focus ones efforts on the highest ranked candidate. The ROSETTA KIC paper claims superior results to those of molecular mechanics refinement so I used this for refinement instead. The refined LoopyMod results can then be viewed as a new version of the LoopBuilder protocol, while keeping in mind that the LoopyMod program is solving a harder problem by remodeling the stems.

Figure 3.1 shows the results of Loopy and LoopyMod, with and without refinement using ROSETTA's KIC refinement protocol. Loopy results were ranked by Loopy itself, and LoopyMod results were ranked by their all-atom DFIRE energy potential using DFIRE.x, part of the LoopBuilder package. These figures should not be compared to those in the following sections, as for these tests Loopy and LoopyMod were remodeling the stem residues, and this makes the problem more difficult.

A two-tailed paired t-test that compared the two tools yielded $p = 0.079$ for local RMSD and $p = 0.021$ for global RMSD. It appears that Loopy results had better global RMSD than LoopyMod results did, but worse local RMSD. However, these differences are not statistically significant. The Loopy results were similar to those in the LoopBuilder paper (their Table V), indicating that the difficulty of the test set was comparable to that of the LoopBuilder test set. However there was substantial improvement after refining the results of Loopy, where in the LoopBuilder paper, Soto, Fasnacht, *et al.* observed a decrease in accuracy after applying PLOP to Loopy results. The improvement to the score by refining Loopy's results (or LoopyMod's) is greater, when expressed as a percentage, than the improvement achieved between Loopy and LoopBuilder, although it is approximately equal when expressed as an absolute value. The Loopy and LoopyMod scores were also close after alignment, with t-tests yielding $p = 0.15$ for local RMSD scores, and $p = 0.16$ for global RMSD scores.

We can conclude that the Loopy results with KIC refinement were an

acceptable stand-in for the LoopBuilder protocol. It may be interesting to examine in detail why PLOP refinement of Loopy's results did not result in improvement to average scores, while ROSETTA refinement did. This may be because LoopBuilder creates only one refinement of each candidate, and the Monte Carlo refinement is only improving the results on average. Since Loopy uses clustering to ensure it does not return similar candidates, the second candidate will not necessarily be a useful stand-in, even after refinement. If the refinement process makes the first candidate worse, and the first candidate was the most accurate one, refining the other candidates will not yield accurate results so these steps are wasted.

3.2.5 Other Tools

I do not have tables comparing LoopWeaver results with those of SuperLooper or FREAD, as neither tool returns results for all loops and it is meaningless to compare averages for different sets. For example, out of the 60 length 10 loops, FREAD (using the same database as LoopWeaver) returned matches for only 6 loops, and SuperLooper (using LIP from 2007) returned 45. For the six results returned by FREAD, the average score was 0.94Å local RMSD. Over the same six loops, LoopWeaver returned an average score of 0.44 Å. With so few results there is no significance to this difference, which is the result of one single error in FREAD's selection of database matches. For the other five loops, both tools made predictions based on the same database matches and the WMDS fitting did not significantly alter the database loop. Also for length 10 loops, SuperLooper's DFIRE ranked results are essentially the same as the LoopWeaver loops when the WMDS scaling is disabled. This is to be expected, since there are only minor implementation differences prior to this step. Thus, the unfitted LoopWeaver result table can be viewed as essentially the same results one could expect from other database tools.

There are more recent loop modeling tools, but their results often have difficulty surpassing the accuracy of existing tools. An example of this is the FALC server[60], a recent fragment assembly loop modeling server. As with the CCD closure protocol from ROSETTA, FALC is a statistical technique that samples angles from a position-specific φ/ψ distribution, built using fragment matches. Their results are much better than those of RAPPER and ModLoop, two tools I also tested against. I submitted the CASP8 portion of the length 10 test set to the FALC server, and observed an average score

Table 3.1: Average RMSD (local/global) scores for Loopy and LoopyMod with remodeling of stem atoms.

Length	Loopy	Loopy Refined	LoopyMod	LoopyMod Refined
6	1.35/2.08	0.76/1.19	1.32/2.16	0.77/1.28
7	1.59/2.44	1.06/1.64	1.61/2.72	1.03/1.72
8	2.00/3.24	1.28/2.09	2.01/3.68	1.40/2.26
9	2.23/3.69	1.54/2.67	2.00/3.56	1.62/2.77
10	2.36/3.73	1.68/2.82	2.23/4.31	1.76/3.18
11	3.00/4.88	2.15/3.94	2.69/4.58	2.15/3.67

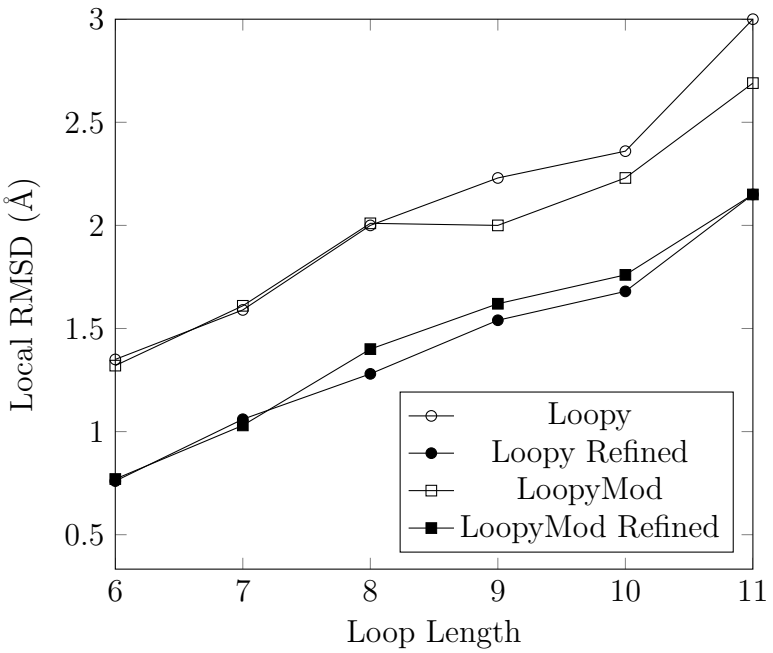


Figure 3.1: Local RMSD scores from Table 3.1 plotted vs. loop length.

of 2.36 local RMSD, and 4.55 global RMSD. This is less accurate than all other tools on this same subset. However, this is roughly the score that they showed in their paper for loops of this length. It is because I have re-ranked the ModLoop and RAPPER results using the DFIRE energy function that these tools perform better than FALC, while in the FALC paper they did not.

3.3 Scores

In Table 3.2 are the average local and global RMSD scores for LoopWeaver with and without the WMDS fitting step. For each test loop, the candidate with the lowest local RMSD was selected. When compared to the DFIRE ranking, this table shows the amount of improvement that is possible. On average, both local and global RMSD scores were improved by using DFIRE fitting. A two-tailed paired t-test returned $p = 4.4 \times 10^{-5}$ for the local RMSD scores, and $p = 3.7 \times 10^{-4}$ for the global RMSD scores, so this improvement was statistically significant.

Table 3.2: Average RMSD (local/global) scores for LoopWeaver with ranking by local RMSD, with and without WMDS fitting.

Length	WMDS	No WMDS
6	0.56/1.04	0.57/1.10
7	0.74/1.37	0.83/1.60
8	0.97/1.97	1.04/2.12
9	1.21/2.49	1.31/2.37
10	1.46/2.83	1.51/2.85
11	1.69/2.98	1.62/2.81

In Table 3.3 I compare the LoopWeaver results with fitting to those without, using DFIRE ranking. In this table we can see that the difference made by WMDS fitting was larger than the difference shown in Table 3.3. The lowest local RMSD scores were approximately 0.05Å lower after WMDS fitting, while the top ranked candidates by DFIRE energy had local RMSD scores approximately 0.15Å lower after WMDS fitting. If the RMSD scores are compared with those of the other tools, the database approach seems to perform well even without fitting. However, the stem RMSD fit results

in less accurate energy potentials because of inaccurate geometry where the loop joins the two stems. For example, the peptide bond that connects the loop backbone to the backbone of the rest of the protein is often very unusual as a result of RMSD fitting. For the LoopWeaver results without WMDS fitting, the average peptide bond length differs from the typical bond length (approximately 1.33Å depending on the type of residue) by up to 1Å, with averages ranging from 0.2Å to 0.4Å. The average over all test sets was 0.20Å, or approximately 13 standard deviations. After WMDS fitting, the average peptide bond length deviation was 0.03Å, or approximately 2 standard deviations, although in a small number of cases the deviation was still unusual (up 10 standard deviations). By obtaining more reasonable geometry, WMDS fitting allowed DFIRE to make more accurate predictions. This explains why WMDS fitting resulted in a larger average score improvement for the DFIRE ranked scores than it did for the minimized scores.

Table 3.3: Average RMSD (local/global) scores for LoopWeaver with and without WMDS fitting.

Length	Loop-Weaver	Loop-Weaver, No Fitting	Average Peptide Bond Deviation
6	0.87/1.37	1.02/1.66	0.204
7	1.18/1.86	1.28/2.15	0.261
8	1.54/2.59	1.80/2.90	0.317
9	1.85/2.91	1.95/3.10	0.312
10	2.02/3.34	2.17/3.59	0.382
11	2.20/3.37	2.24/3.55	0.285

Table 3.4 (and accompanying plot) shows the results of running LoopWeaver, as well as Loopy, ModLoop, RAPPER, and ROSETTA on the various test sets. LoopWeaver performed quite well when compared with Loopy, ModLoop, and RAPPER. With the exception of the length 8 targets where all tools scored approximately the same, LoopWeaver produced lower average RMSD scores, both local and global. Table 3.5 shows the p-values for paired two-tailed t-tests comparing the scores for all pairs of tools shown in Table 3.4. As shown in Table 3.4, the most similar tools by local RMSD score were Loopy and ModLoop. Accordingly, this pair of tools had the highest p value for both for local and global RMSD.

Table 3.4: Average RMSD (local/global) scores for tested tools.

Length	Loop-Weaver	ModLoop	Loopy	Rapper	ROSETTA
6	0.87/1.37	0.97/1.48	1.17/1.71	1.24/2.01	0.66/1.05
7	1.18/1.86	1.36/2.12	1.39/2.10	1.47/2.35	0.93/1.53
8	1.54/2.59	1.57/2.63	1.61/2.27	1.83/2.92	1.23/1.99
9	1.85/2.91	1.97/3.32	2.02/3.03	2.12/3.46	1.56/2.71
10	2.02/3.34	2.39/3.99	2.10/3.29	2.14/3.33	1.65/3.09
11	2.20/3.37	2.41/3.90	2.59/3.76	2.59/4.17	2.04/3.38

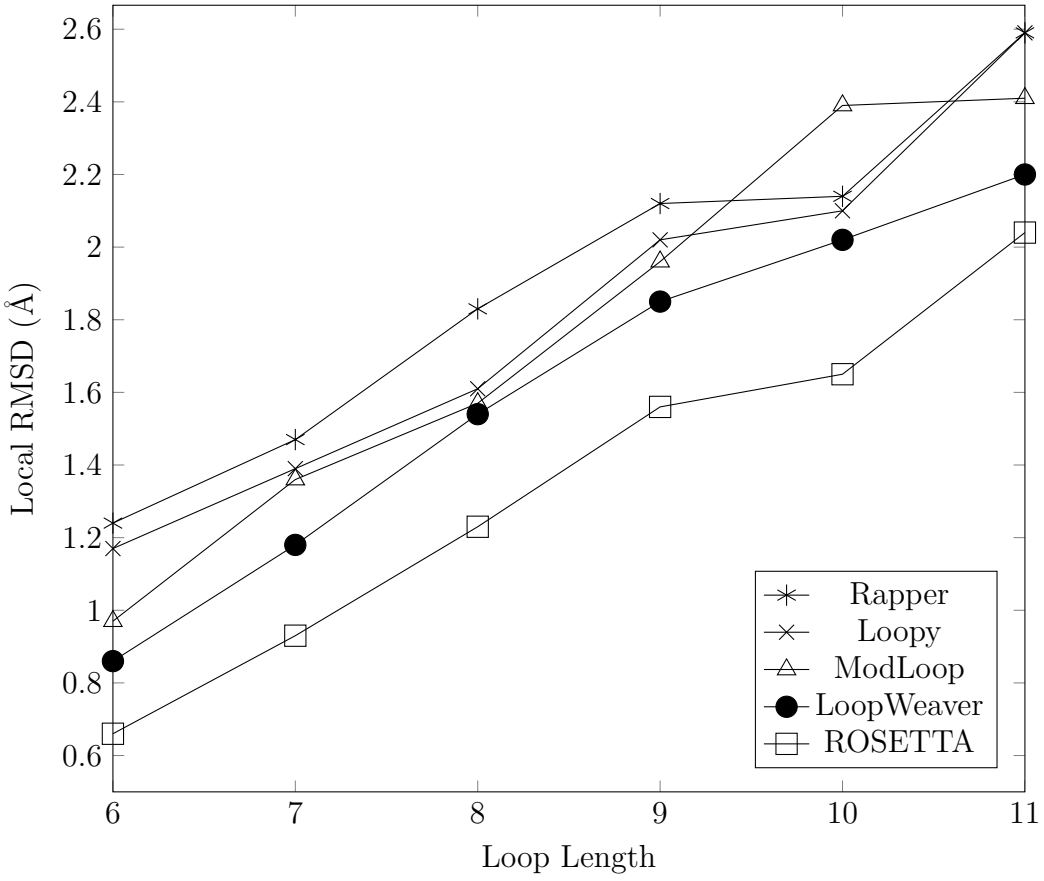


Figure 3.2: Local RMSD scores from Table 3.4 plotted vs. loop length.

Table 3.5: p-values obtained from two-tailed, paired t-tests for local and global RMSD scores in Table 3.4

RMSD Metric	Tool 1	Tool 2 ModLoop	Loopy	Rapper	Rosetta
Local	LoopWeaver	8.8×10^{-5}	4.1×10^{-8}	4.0×10^{-19}	6.3×10^{-11}
	ModLoop		0.038	7.7×10^{-9}	3.4×10^{-28}
	Loopy			5.0×10^{-4}	2.4×10^{-37}
	Rapper				6.2×10^{-50}
Global	LoopWeaver	1.6×10^{-4}	0.040	2.1×10^{-15}	3.6×10^{-6}
	ModLoop		0.081	1.6×10^{-5}	3.1×10^{-79}
	Loopy			4.2×10^{-5}	1.7×10^{-87}
	Rapper				2.4×10^{-106}

ROSETTA was the best stand-alone loop modeling tool, even though it was only generating 10 instead of the recommended 1000 candidates. This does not necessarily indicate that ROSETTA's loop closure approach is superior, only that as a whole, including the extensive refinement step, it yielded the best results. ROSETTA's modular design allows this thought to be explored easily, as it is trivial to replace ROSETTA's loop closure with the results of an external tool, while still taking full advantage of ROSETTA's refinement and perturbation algorithms.

In Table 3.6 are the results of the same tools from Table 3.4, using ROSETTA KIC refinement. Refinement was done by using ROSETTA to generate 10 refined candidates from the top ranked candidate produced by each tool. An alternative would have been to generate 1 refinement for each of the top 10 candidates, similar to what is done with the LoopBuilder protocol using PLOP refinement. However, the nature of the Monte Carlo simulations used means that there is a real possibility of refining a loop into a much worse candidate. Since the other tools tend to create reasonable rankings, the cost of ruining the top 1 result with a bad refinement outweighs the additional coverage of refining the other loop candidates. I speculate that this is the reason that, in the LoopBuilder paper[38], Loopy's results were worse after PLOP refinement than they were before it.

All tools reported substantial improvement from ROSETTA's refinement computation. RAPPER, MODELLER, and Loopy scores were more improved than LoopWeaver scores. Loopy, LoopWeaver, and ModLoop yielded

Table 3.6: Average RMSD (local/global) scores for tested tools after ROSETTA refinement.

Length	Loop-Weaver	ModLoop	Loopy	Rapper	ROSETTA
6	0.57/0.87	0.64/0.94	0.59/0.88	0.73/1.12	0.66/1.05
7	0.84/1.35	0.84/1.27	0.78/1.24	0.86/1.68	0.93/1.53
8	1.14/1.75	1.06/1.51	1.05/1.64	1.38/2.27	1.23/1.99
9	1.34/2.24	1.51/2.33	1.32/2.13	1.65/2.77	1.56/2.71
10	1.52/2.42	1.53/2.44	1.44/2.26	1.70/2.69	1.65/3.09
11	1.87/2.94	1.87/2.88	1.75/2.70	2.20/3.69	2.04/3.38

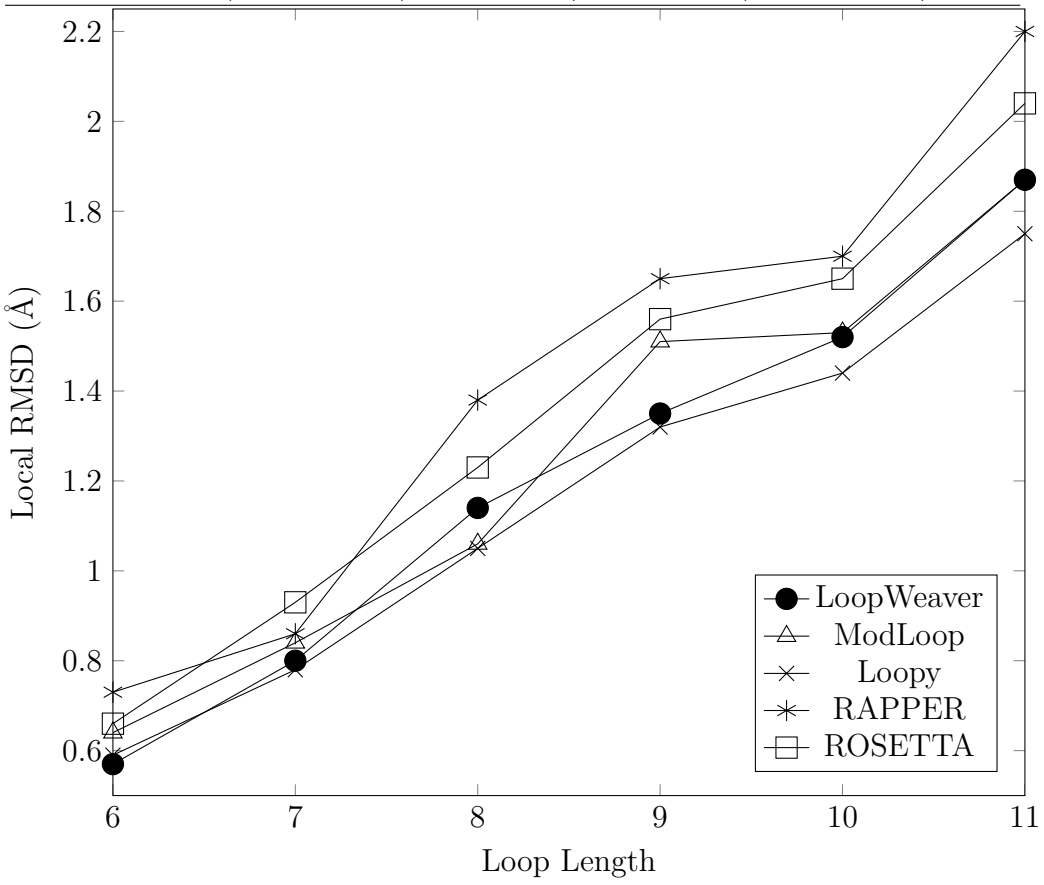


Figure 3.3: Average Local RMSD scores from tested tools after ROSETTA refinement, plotted vs. loop length.

Table 3.7: p-values obtained from two-tailed, paired t-tests for local and global RMSD scores, after ROSETTA refinement (Table 3.6).

RMSD Metric	Tool 1	Tool 2			
		ModLoop	Loopy	Rapper	Rosetta
Local	LoopWeaver	0.31	0.23	5.0×10^{-9}	4.7×10^{-4}
	ModLoop		8.4×10^{-3}	2.6×10^{-7}	2.0×10^{-8}
	Loopy			1.0×10^{-13}	3.5×10^{-3}
	Rapper				0.011
Global	LoopWeaver	0.65	0.17	5.2×10^{-8}	1.7×10^{-4}
	ModLoop		0.23	8.2×10^{-11}	3.1×10^{-9}
	Loopy			2.0×10^{-13}	1.8×10^{-6}
	Rapper				0.13

refined results superior to those from the default ROSETTA candidates. While LoopWeaver was initially the most accurate tool other than ROSETTA, after refinement it appeared to lie between Loopy and ModLoop. Paired t-tests, shown in Table 3.7 did not give confidence in this ranking so we can only say that all three tools appear approximately equal. However, all three are significantly more accurate than the ROSETTA result alone.

One explanation for why LoopWeaver's results were less improved by refinement is that accurate predictions can only achieve a small amount of improvement, and greatly inaccurate predictions can cannot be improved substantially if the maximum change permitted is low. A database method has more accurate predictions, but also has more inaccurate predictions so this could explain why LoopWeaver did not gain as much accuracy from refinement as the other tools did.

3.4 Combined Scores

Because database tools often have not been able to produce results for all gaps, they have been used as a complement to statistical methods. FREAD for example yields very good sub-angstrom results regardless of the length of the loop[48] but is often not able to produce results at all. The FREAD authors therefore use FREAD to supplement predictions obtained by other means. LoopWeaver is able to make usable predictions in all attempted cases,

but this is no reason to abandon the idea of combining its results with those from other approaches. Statistical and database approaches have different strengths and weaknesses. A database approach is very accurate if there is a very similar loop available in the database, and otherwise their performance depends on how well they can manage with dissimilar database candidates. A statistical approach is very accurate if the native loop comes from a dense part of the angle search space, but if the true structure involves uncommon angles it is unlikely to be sampled and so may not be found.

In Figure 3.8 we can see the results of combining the results of two tools. All combinations were obtained by taking the top candidate from each tool according to that tool's ranking, and then using DFIRE to pick between these two candidates. In general the combination approach did not seem to work well. In some cases there was improvement, but for each combination there were some lengths for which the combined score fell between the scores for the two methods being combined. The only exception was the Loopy and LoopWeaver combination, which consistently scored better than either tool did alone. This is convenient as these are also the two fastest methods.

Table 3.9 shows the t-test results that compare all pairs of combined scores. This shows that while most tool combinations were significantly different from each other, the Weaver plus ModLoop combination was not significantly different than the Weaver plus Rapper combination. Table 3.10 shows p-values for t-tests between each combination of two tools, and its constituent tools. Only the LoopWeaver plus Loopy scores differed significantly from the scores of both of components methods.

These results seems to have been caused by idiosyncrasies within the various methods. ModLoop, for example, appears to generate candidates with better DFIRE energy, on average. That means that for loops with approximately the same score, the ModLoop loop is more likely to be selected according to its DFIRE energy. This lead to many cases where the competing result had a lower RMSD, but was not selected due to the energy biases of the two methods. For example, when ModLoop and Loopy were combined, the ModLoop result was selected 94% of the time (657 of 697 loops) while the ModLoop candidate only had a lower local RMSD score for 54% (367 of 697) loops. A similar pattern held when the LoopWeaver and ModLoop results were combined. The ModLoop result was selected 75% of the time (527 of 697), but only had a lower RMSD score for 44% of the test loops (306 of 697).

Table 3.8: Average RMSD (local/global) scores for combined results. W for Weaver, L for Loopy, M for ModLoop, R for RAPPER

Len	W+M	W+L	W+R	M+L	M+R	L+R
6	0.91/1.43	0.84/1.30	0.94/1.46	1.00/1.55	1.01/1.57	1.19/1.87
7	1.18/1.84	1.12/1.76	1.18/1.87	1.34/2.09	1.36/2.12	1.43/2.23
8	1.51/2.53	1.39/2.19	1.54/2.47	1.51/2.51	1.58/2.65	1.71/2.66
9	1.72/2.90	1.79/2.72	1.84/3.00	1.91/3.19	1.97/3.38	2.01/3.22
10	2.18/3.59	1.96/3.11	1.94/3.13	2.31/3.81	2.33/3.92	2.15/3.52
11	2.19/3.53	2.11/3.18	2.27/3.68	2.33/3.65	2.37/3.89	2.60/4.24

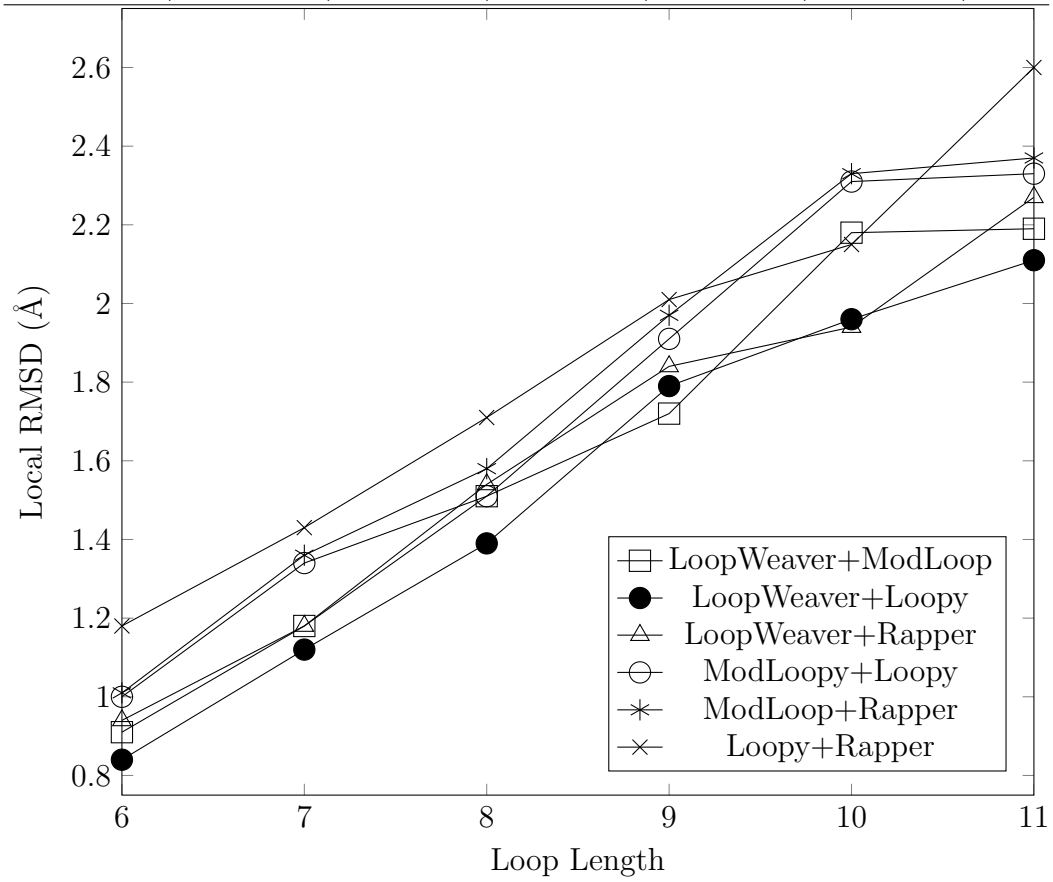


Figure 3.4: Average Local RMSD scores for combined results, plotted vs. loop length.

Table 3.9: p-values for two-tailed paired t-tests comparing all pairs of combined tools by local and global RMSD scores.

Local RMSD

	Pair 2 [a]				
Pair 1[a]	W+L	W+R	M+L	M+R	L+R
W+M	0.016	0.71	8.1×10^{-6}	1.9×10^{-11}	3.2×10^{-13}
W+L		1.9×10^{-4}	8.4×10^{-7}	6.2×10^{-10}	2.0×10^{-22}
W+R			0.011	5.3×10^{-5}	4.1×10^{-16}
M+L				2.1×10^{-3}	1.2×10^{-6}
M+R					3.9×10^{-4}

Global RMSD

	Pair 2 [a]				
Pair 1 [a]	W+L	W+R	M+L	M+R	L+R
W+M	1.2×10^{-4}	0.92	4.9×10^{-4}	1.7×10^{-10}	7.5×10^{-8}
W+L		2.8×10^{-6}	1.6×10^{-8}	1.4×10^{-13}	1.5×10^{-22}
W+R			0.031	8.5×10^{-6}	1.2×10^{-12}
M+L				2.8×10^{-4}	4.4×10^{-4}
M+R					0.14

[a]W for LoopWeaver,M for ModLoop, L for Loopy, R for RAPPER

Table 3.10: p-values for two-tailed paired t-tests comparing combined tool scores to the individual scores of each tool.

Tool	Tool 1		Tool 2	
Combination[a]	Local	Global	Local	Global
W+M	0.85	0.61	1.8×10^{-10}	8.1×10^{-9}
W+L	6.7×10^{-4}	2.9×10^{-6}	6.5×10^{-17}	3.8×10^{-8}
W+R	0.46	0.61	7.8×10^{-27}	1.0×10^{-23}
M+L	3.6×10^{-5}	4.0×10^{-4}	$3.9 \times^{-4}$	0.72
M+R	0.97	0.36	7.8×10^{-27}	1.0×10^{-23}
L+R	0.18	4.4×10^{-5}	1.4×10^{-7}	2.1×10^{-6}

[a]As "Tool1 + Tool2", with tools abbreviated using W for LoopWeaver,M for ModLoop, L for Loopy, R for RAPPER

Just as ROSETTA refinement improved the stand-alone scores for each tool, it also improved the combined scores for all of the tools I tested. In Figure 3.11 I show the results of applying KIC refinement to the combined results. In this case I used ROSETTA to produce 5 candidates from one tool's top result, and 5 from the other's. This was ROSETTA was still generating a total of 10 refined loops, and the total running time would remain approximately the same. Unlike the unrefined combinations, all refined combinations reported scores superior to either tool alone. This may be because all of the candidates were refined under the ROSETTA energy function, and the same function is being used to pick the best refined candidate.

Table 3.9 shows the p-values obtained by performing paired t-tests between all pairs of tool combinations. From these numbers we see that although the LoopWeaver plus ModLoop, and LoopWeaver plus Loopy combinations had the lowest average scores, they was not enough difference to claim with confidence that the LoopWeaver plus Loopy combination was more accurate than the LoopWeaver plus ModLoop combination was.

Table 3.13 shows the results of paired t-tests comparing the combined refined scores with the refined scores for each of the two individual methods involved. For example, the "M+L" line shows the p-values for t-tests between the ModLoop and Loopy combined result and both ModLoop and Loopy alone. Here we can see that other than the ModLoop and Rapper and the Loopy and Rapper pairs, all tool combinations show significant improvement over either of their constituent tools alone.

ROSETTA's KIC refinement made substantial improvements to the average scores for all of the test sets, resulting in much lower scores than ROSETTA alone achieved with the same approximate execution time. These results held even when I allowed the ROSETTA *ab initio* execution substantially more time. I obtained numbers for length 9 only, due to the large amount of CPU time required. For this set I generated the recommended 1000 candidates, and the average ROSETTA score was 1.15Å for the local RMSD, and 1.82Å for the global RMSD. The refined Loopy and LoopWeaver combined score for the same test set was 1.13Å local and 1.74Å global RMSD. So, the Loopy plus LoopWeaver protocol was able to obtain results comparable to ROSETTA was able to use 100 times as much CPU time (approximately 30,000 hours compared to approximately 300 hours).

Table 3.11: Average RMSD (local/global) for combined tools[1], after ROSETTA refinement.

Len	W+M	W+L	W+R	M+L	M+R	L+R
6	0.54/0.81	0.53/0.79	0.54/0.84	0.57/0.83	0.61/0.92	0.58/0.86
7	0.70/1.05	0.75/1.13	0.76/1.19	0.75/1.14	0.81/1.22	0.80/1.27
8	1.00/1.44	0.93/1.36	1.08/1.60	0.90/1.31	1.06/1.59	1.03/1.57
9	1.26/1.98	1.13/1.74	1.31/2.09	1.35/2.13	1.40/2.15	1.34/2.12
10	1.29/2.11	1.22/1.91	1.33/2.33	1.37/2.20	1.51/2.43	1.47/2.43
11	1.64/2.65	1.53/2.42	1.64/2.63	1.82/2.79	2.00/3.14	1.90/2.98

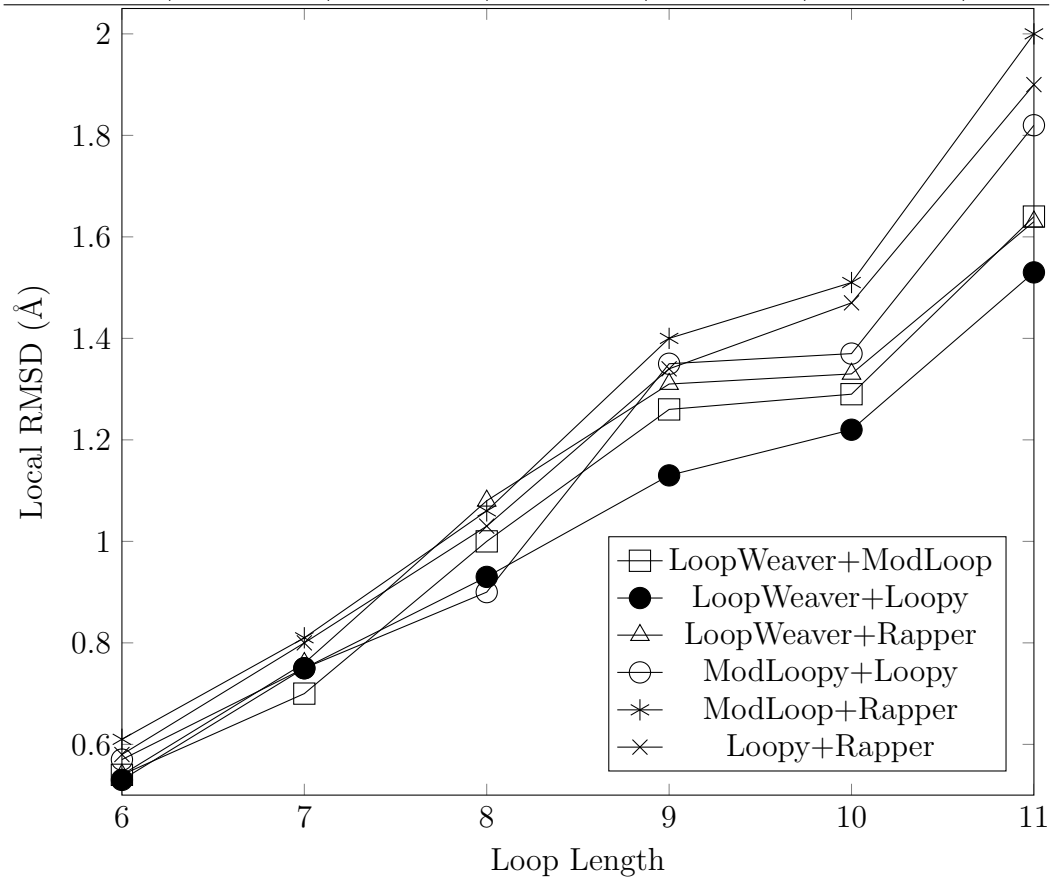


Figure 3.5: Average Local RMSD scores for combined results after ROSETTA refinement, plotted vs. loop length.

Table 3.12: p-values for two-tailed paired t-tests comparing all pairs of combined tools by local and global RMSD scores, after ROSETTA refinement.

Local RMSD

	Pair 2 [a]				
Pair 1[a]	W+L	W+R	M+L	M+R	L+R
W+M	0.090	0.084	0.194	1.8×10^{-7}	5.3×10^{-3}
W+L		8.5×10^{-4}	2.4×10^{-3}	7.8×10^{-8}	3.1×10^{-7}
W+R			0.82	5.0×10^{-3}	0.12
M+L				3.5×10^{-5}	0.015
M+R					0.18

Global RMSD

	Pair 2 [a]				
Pair 1 [a]	W+L	W+R	M+L	M+R	L+R
W+M	0.088	0.011	0.30	2.1×10^{-6}	4.5×10^{-3}
W+L		2.4×10^{-5}	5.2×10^{-3}	5.2×10^{-7}	2.4×10^{-7}
W+R			0.24	0.12	0.31
M+L				3.3×10^{-4}	4.3×10^{-3}
M+R					0.64

[a]W for LoopWeaver,M for ModLoop, L for Loopy, R for RAPPER

Table 3.13: p-values for two-tailed paired t-tests comparing combined tool scores to the individual scores of each tool, after ROSETTA refinement.

Tool	Tool 1		Tool 2	
Combination[a]	Local	Global	Local	Global
W+M	3.1×10^{-6}	2.3×10^{-6}	7.8×10^{-10}	9.6×10^{-7}
W+L	1.2×10^{-9}	8.2×10^{-10}	8.2×10^{-6}	5.5×10^{-6}
W+R	4.0×10^{-4}	5.6×10^{-3}	5.5×10^{-21}	3.9×10^{-19}
M+L	1.3×10^{-7}	9.6×10^{-5}	0.065	0.028
M+R	0.10	0.53	4.6×10^{-12}	2.5×10^{-14}
L+R	0.55	0.55	8.9×10^{-17}	4.1×10^{-16}

[a]As "Tool1 + Tool2", with tools abbreviated using W for LoopWeaver,M for ModLoop, L for Loopy, R for RAPPER

3.5 Database Accuracy over Time

One major advantage of database techniques is that the Protein Databank (PDB) is constantly expanding as scientists around the world determine new protein structures. As time passes, database based loop modeling techniques will have more and more loops available to use as templates. When designing LoopWeaver I saw large improvement in accuracy when I changed from a non-redundant database to a more complete snapshot of the PDB. While homologous proteins may be very similar in overall structure, it does not take a very large $\text{RMSD}_{\text{stem}}$ before a loop template will not fit into the corresponding gap in a very similar homologous protein. The CASP targets used for testing LoopWeaver typically have one or more full domain template matches, and yet very few of the loops have native-like loops available in the database.

To demonstrate the accuracy improvements over time, I ran LoopWeaver (without ROSETTA KIC refinement) using yearly snapshots of the PDB from 2000 through 2008. In Figure 3.6 we can see the effect of an older (and thus smaller) loop set on the quality of all of the loop test sets. In all cases there is a noticeable downward trend (though, especially in the global RMSD scores, this is not a monotonic trend), and depending on test set the final 2008 score is between 10 and 20 percent lower than the 2000 score. These trends all appear approximately linear. Table 3.14 shows the p-values for t-tests comparing the LoopWeaver results for a given database year to those for later database years. We can see that although the difference from one year to the next is generally not significant (meaning $p < 0.05$), the difference from one year to two years later is statistically significant (other than for the 2003 to 2005 comparison).

Much of the noise in these graphs appears to be caused by the difficulty of ranking poor results relative to each other. That is, loops that are not particularly good (say 1.5Å local RMSD or higher) are usually ranked as worse than a native-like loop, but are not consistently ranked among themselves, leading to many cases where a 1.5Å loop might be replaced by a 1.9Å loop when it becomes available.

Very little of LoopWeaver's total running time is a due to the database search. It takes approximately 10 seconds to load the database and perform a complete search. The WMDS solver, sidechain packing, and DFIRE energy evaluation make up the majority of the running time requirement, at

Table 3.14: Paired t-tests testing LoopWeaver's average local and global RMSD scores when using a database from a given year against the same scores when using a database from one or two years later.

Year	Year + 1[a]		Year + 2[b]	
	Local	Global	Local	Global
2000	0.03	0.329	0.0073	0.0045
2001	0.24	0.010	1.6×10^{-4}	1.0×10^{-6}
2002	8.2×10^{-4}	3.5×10^{-4}	7.5×10^{-4}	8.0×10^{-3}
2003	0.75	0.60	0.12	0.19
2004	0.39	0.15	0.0059	0.0079
2005	0.023	0.14	0.0084	0.046
2006	0.27	0.33	0.0093	0.0030
2007	0.037	0.0053		

[a]For example, in the 2005 row this column represents t-tests between the 2005 tests and the 2006 tests

[b]For example, in the 2005 row this column represents t-tests between the 2005 tests and the 2007 tests

approximately 10-15 minutes, depending on the size of the loop involved and the complexity of its local environment. For a database with 100 times as many proteins as my current database, the running time due to the database search should still be less than half of the total running time required by LoopWeaver. If one is also using ROSETTA to refine the LoopWeaver results, the database search is an even smaller portion of the total running time.

The runtime difference between the 2200 chain database from 2000 and the 14,400 chain database from 2008 was indistinguishable during these tests. As an example, all 9 trials for the set of length 6 loops took approximately 50 hours of CPU time on the Sharcnet Saw cluster, or about 15 minutes per individual loop. This value includes TreePack and DFIRE time but does not include ROSETTA refinement.

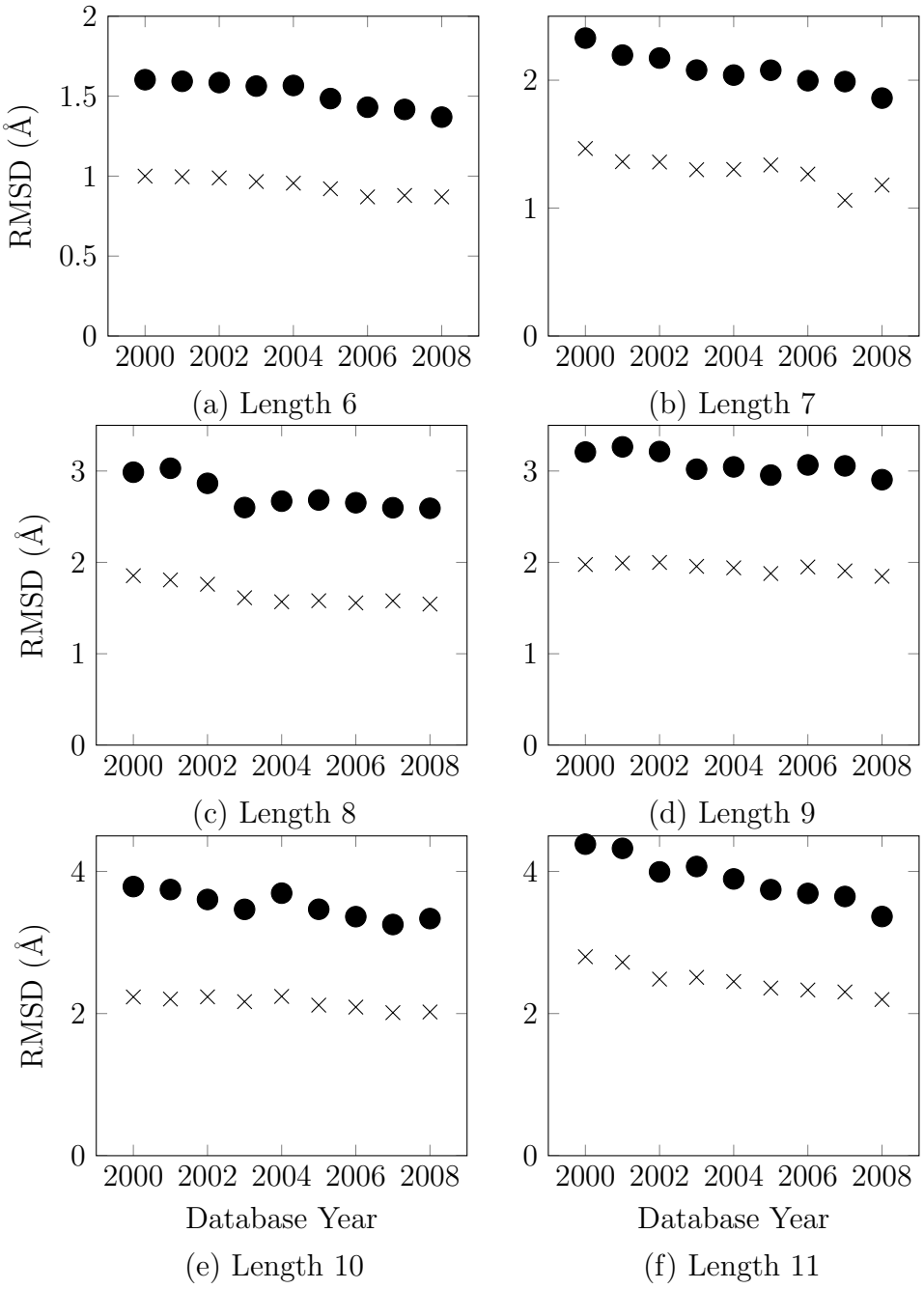


Figure 3.6: Plots of LoopWeaver's average local (×) and global (•) RMSD scores vs. database cutoff date.

3.6 Example Loops

3.6.1 LoopWeaver Compared with Other Tools

While LoopWeaver demonstrates good results on average, there are cases where it does well and cases where it does poorly, as with any other tool.

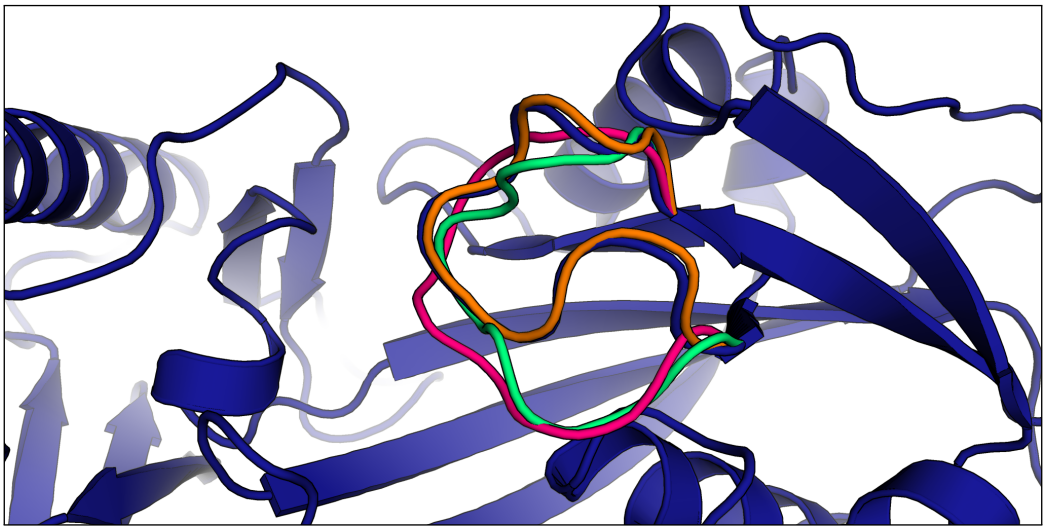
T0513 residues 139-148

An example situation where LoopWeaver does well can be seen in the loop in target T0513 from residue 139 to 148, as shown in Figure 3.7a. For this loop, Loopy had a global RMSD of 3.01Å and ModLoop had a global RMSD of 4.44Å. LoopWeaver's candidate had a local RMSD of 0.27Å and a global RMSD of 0.67Å, which is a very accurate prediction. This is because a homolog exists in the database (see Figure 3.7b), and LoopWeaver was able to find and select this loop. This is a situation where any database method should yield good results, as LoopWeaver's fitting algorithm is not needed for loops that already fit well. Before WMDS fitting this loop had local and global RMSDs of 0.27Å and 0.55Å, so the scaling in this case negatively impacted the orientation of the loop though not substantially. However, the WMDS fitting improved the DFIRE energy very slightly from -608.851 to -612.196. Without WMDS fitting, this candidate was ranked second and the first candidate had a very poor score (6.29Å global RMSD).

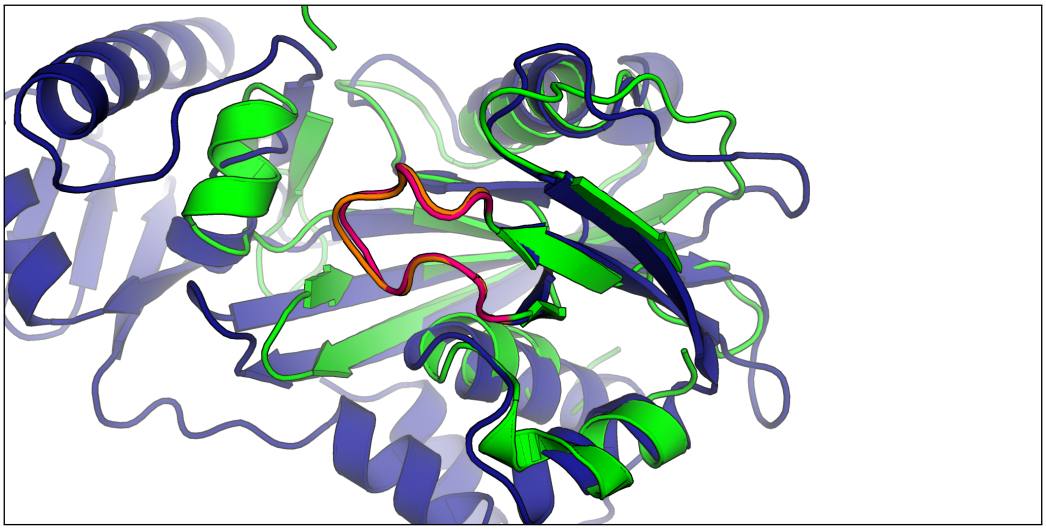
As mentioned previously, homolog matches are not common in this test set, even though full domain homologs are available for almost all of the target proteins. Generally the homologs available at the time of the CASP experiments were not similar enough for homologous loops to fit, even if the loop itself remained the same length. Fitting a loop requires similarity on a much finer scale. If we look at Figure 3.7b once more, we can see that while the two proteins are structurally similar, the majority of the loops differ significantly.

T0532 residues 303-331

Database methods such as LoopWeaver are able to make predictions with sub-angstrom accuracy even if a homologous template is not available , as



(a) Native Chain (Blue) and Loop (A) compared with LoopWeaver (B,Orange), Loopy (C,Pale Green), and ModLoop (D,Magenta). Loop starts at the upper strand, residue 139.



(b) Target T0513 (blue, loop in orange) and template 2KFB chain A (green, loop in magenta) with superposition of homologous structures. Loop starts at the upper strand, residue 139 in target, 57 in template.

Figure 3.7: Candidates and LoopWeaver Template used for target T0513, length 10 loop at residues 139 through 148.

there are often similar loops available in the database. For example, for the length 9 loop in target T0532, residues 303 through 311, the prediction made by LoopWeaver has a local RMSD of 0.80Å, and a global RMSD of 1.84Å. Figure 3.8 shows the template used (PDB ID 2C9Y, chain A). Although the helix at one end of the loop (indicated by an arrow in Figure 3.8) aligns with a similar helix in the template protein, overall the query and template proteins are not similar. The majority of the loop region is a β-strand in the template protein. The template itself has 0.90Å local RMSD with the native loop. LoopWeaver required unusually large changes during the WMDS step because of a 0.7Å stem RMSD. Normally matches with large stem RMSDs are not used, but this loop is unusually extended and there were few database loops with a lower stem RMSD. Prior to fitting, the peptide bond lengths used to connect this template loop to the stems were 0.74Å (or 50 standard deviations) greater than the average bond length. After WMDS fitting, all bond lengths were within 2 standard deviations of the mean values for the appropriate bond types. As the gap has a large span, the conformation space is quite limited by the closure requirements. Statistical methods work well when dealing with a constrained search space, so this is an ideal situation for them. ModLoop, Loopy, and Rapper all achieved excellent results, with local RMSDs of 0.44Å,0.76Å, and 0.62Å respectively. While LoopWeaver did not do as well as these tools, all four predictions were quite accurate. This is especially promising for LoopWeaver considering the lack of similar database matches. For this target no database loops would fit without WMDS fitting.

T0508, residues 82-88

For the length 7 loop in target T0508, residues 82 through 88, shown in Figure 3.9, the LoopWeaver prediction has a local RMSD of 0.67Å and a global RMSD of 1.23Å. Sub-Figure 3.9b shows the native loop and the template loop, superimposed using the loop and stems regions of the native and template structures only. Although the template loop is very similar to the native loop(0.14Å local RMSD) the template is not otherwise similar to the query protein. Both end (residue 88 and 172 for query and template respectively) with a helix, labeled with arrow 1 in Sub-Figure 3.9b, but these two helices do not line up well if the loops are aligned. The template loop starts at another helix (labeled with arrow 2), while the query loop starts at a β-strand (labeled with arrow 3). In this particular case the prediction before WMDS

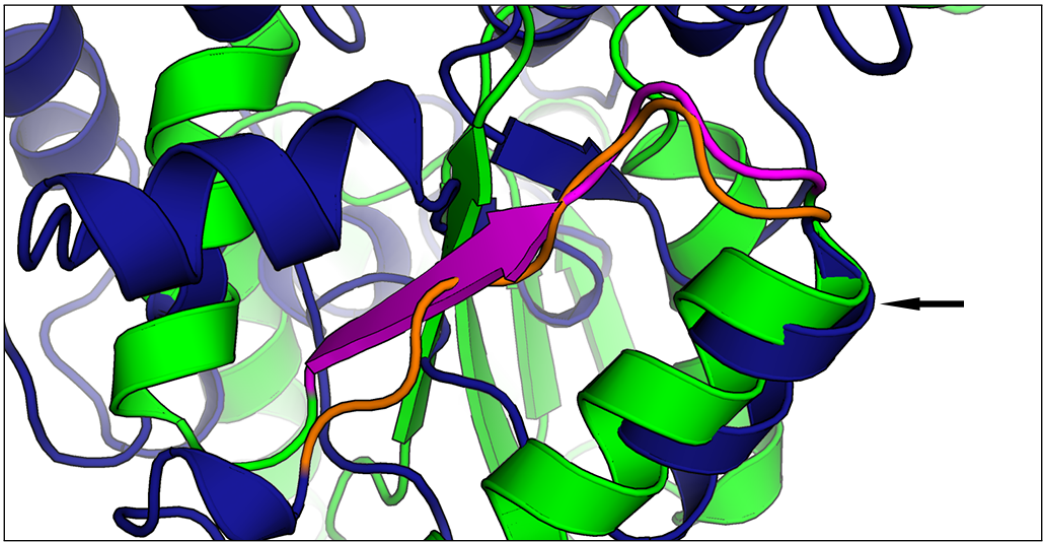
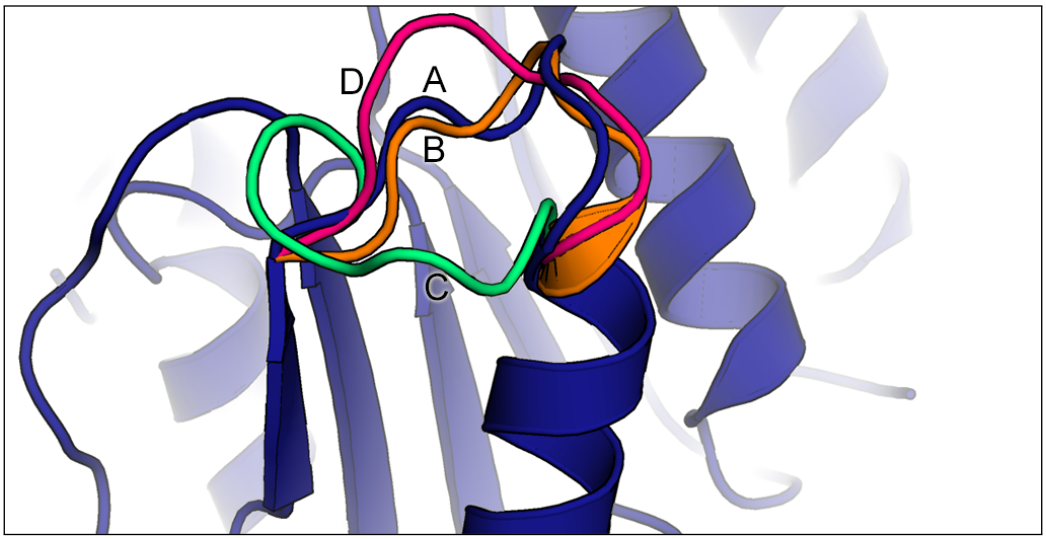


Figure 3.8: T0532 (blue) and loop from residue 303 to 311 (orange), superimposed with LoopWeaver Template protein 2C9Y-A (green) residues 207-215 (magenta). Superposition was calculated using only loop and stem main chain atoms.

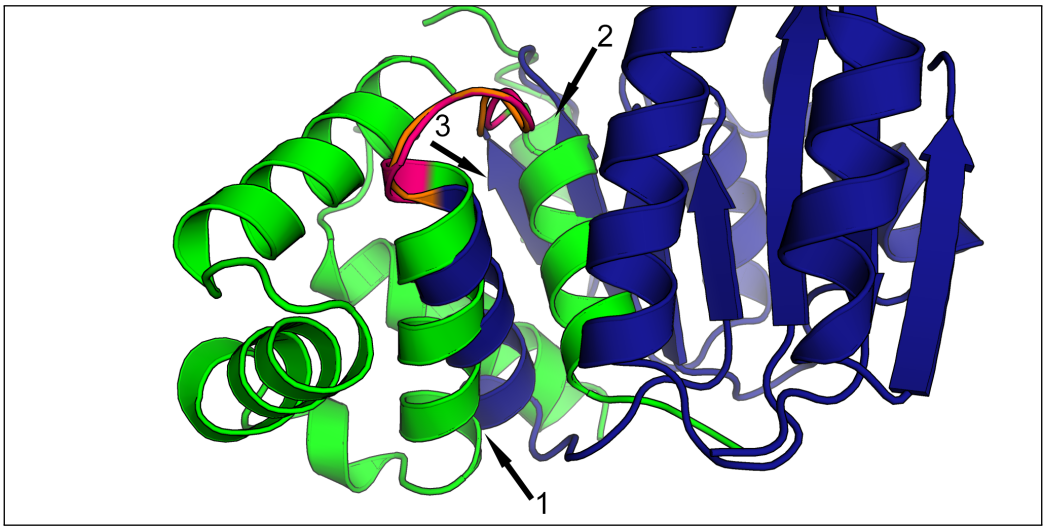
fitting had RMSD scores of 0.71Å and 1.93Å so again the WMDS fitting bent the loop slightly, increasing the local RMSD, while achieving a better orientation thereby lowering the global RMSD. Additionally, the DFIRE energy function did not select this loop without WMDS fitting. Instead, it selected a different loop with 0.65Å local RMSD and 1.69Å global RMSD. So, both scores were improved by WMDS fitting due to improved candidate selection. Sub-Figure 3.9a shows the LoopWeaver loop (B, orange) compared to the native loop (A, blue), the Loopy (C, green) and ModLoop (D,magenta) loops. The Loopy loop has scores of 2.94Å local and 7.79Å global because it was bent in the wrong direction. The ModLoop loop followed the correct general path but bulged outward toward the stem at residue 82, which resulted in RMSD scores of 1.65Å local and 2.37Å global.

T0533 residue 189-199

Another example that specifically highlights LoopWeaver's advantage over other database techniques can be seen in the length 11 loop from target



(a) Native loop (A, Blue), compared with LoopWeaver (B, Orange) Loopy (C, Pale Green) and ModLoop (D,Magenta) candidates. Loop is from residue 82-88, with 82 connected to the strand and 88 to the helix.



(b) Template loop 1LQ1, chain A (green), loop at residues 166 through 172 (magenta). Superimposed with native structure of T0508 (blue) with loop at residues 82-88 (orange).

Figure 3.9: Candidates and LoopWeaver Template for target T0513, length 10 loop at residues 82 through 88.

T0533 (see Figure 3.10). LoopWeaver did not find any similar loops in its database. The lowest energy candidate prior to WMDS fitting had a local RMSD of 2.27Å and global RMSD score of 4.06Å, which is an inaccurate prediction. The lowest energy candidate after WMDS fitting had a local RMSD score of 1.46 and a global RMSD score of 1.66Å, which is a medium quality prediction. This difference was due to an improvement to the positioning of all of the loops allowing for more accurate discernment based on their DFIRE energy. For reference, the best Loopy candidate for this loop has local and global RMSDs of 1.77Å and 2.10Å respectively, and ModLoop's prediction has scores of 1.63Å local and 2.12Å global. These numbers are all lower than the respective tools' averages for the length 11 test set. Although no tool achieved sub-angstrom accuracy for the local RMSD, LoopWeaver came close, while a database approach without WMDS fitting made a poor prediction.

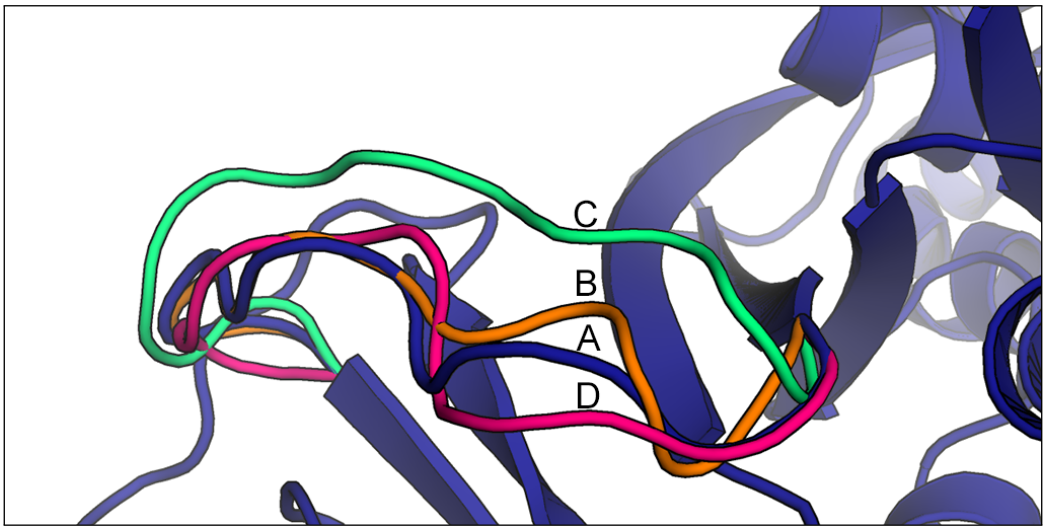


Figure 3.10: T0533 length 11 loop from residue 189 to 199. Native in blue (A), LoopWeaver in orange (B), LoopWeaver without fitting in green (C), ModLoop in magenta (D).

T0625 residue 89-97

It was not always the case that LoopWeaver performed well when compared to existing tools. In target T0625, length 9 loop from residue 89 to 97, as

seen in Figure 3.11a we can see that the native loop connecting the helix to the beta strand is tucked inward, running roughly parallel to other loops but not close enough to form hydrogen bonds. Loopy's prediction was close at the strand end, and followed the same general path, although its connection at the helix end differed. LoopWeaver's result however was not similar at all, and had a local RMSD of 3.75Å and a global RMSD of 9.04Å. LoopWeaver placed the loop on the wrong side, where it extended out from the core of the protein. Additionally, it included a small helix in the middle. This was because the template used came from the helix region of protein 2H12 chain A (See Figure 3.11b for an image of the loop template used). LoopWeaver does not assume all gaps occur in loop regions and so it does not filter according to secondary structure (this is the case with the FREAD and SuperLooper database methods as well). In general the DFIRE energy function should filter inappropriate secondary structures due to steric clashes. However, in this case the helix was energetically favourable in terms of its dihedral angles. It is nevertheless quite unusual as there are two proline residues placed into the helix from the template. While the proline residues placed in this helix take on very favourable angles, proline is unique in that it has two groups bound to the amine nitrogen, making it unable to donate the hydrogen bond required to form a helix structure. While proline residues can occur in a helix they result in a less stable structure due to this missing bond, making these occurrences quite rare. Typically, knowledge-based tools use a proline filter because of proline's more limited geometry. Such a filter would not have helped with this case, as the small angle basins for proline fit well within the norm for $\alpha-$helix structures.

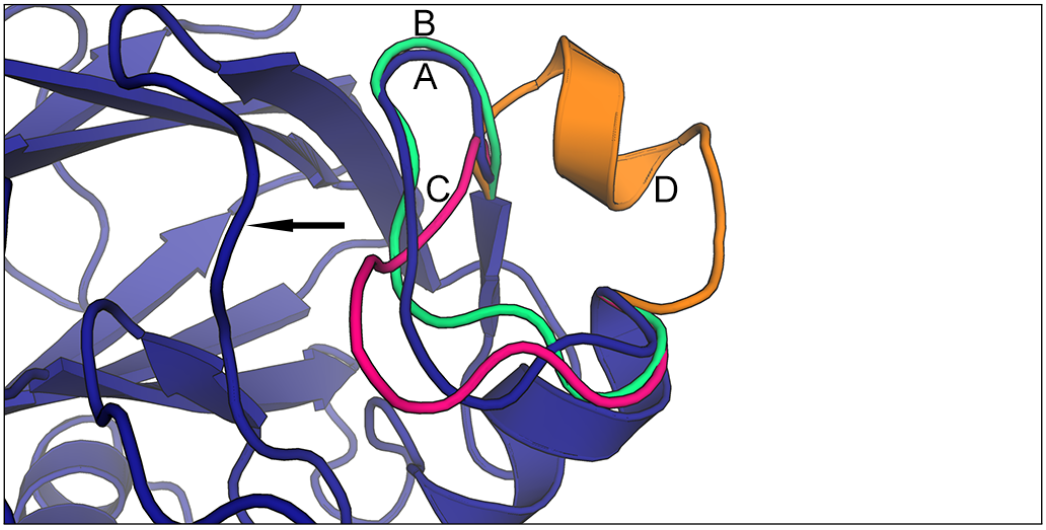
The next four templates for this target all involved extending the helix by one or two residues before transitioning to a loop region. It was not until the sixth model that LoopWeaver produced a more realistic loop prediction. This candidate came from a loop region, and was correctly oriented relative to a nearby loop (indicated by the arrow in Sub-Figure a). This prediction scores 1.99Å local, and 2.38Å global, compared 1.40Å and 1.50Å for Loopy, and 2.84Å local and 4.24Å global for ModLoop. The LoopWeaver result shown has RMSD scores of 3.46Å local and 9.12Å global. ROSETTA refinement improves the energy of the loop and unwinds the helix, but the LoopWeaver candidate was too far from the correct path to result in an accurate refined candidate. The local RMSD was improved by ROSETTA to 2.55Å but the global RMSD degraded to 10.84Å. For comparison the Loopy loop was greatly

improved by ROSETTA, with RMSD scores of 0.40Å local and 0.47Å global, and the ModLoop candidate was also improved, with RMSD scores of 1.12Å local and 1.42Å global.

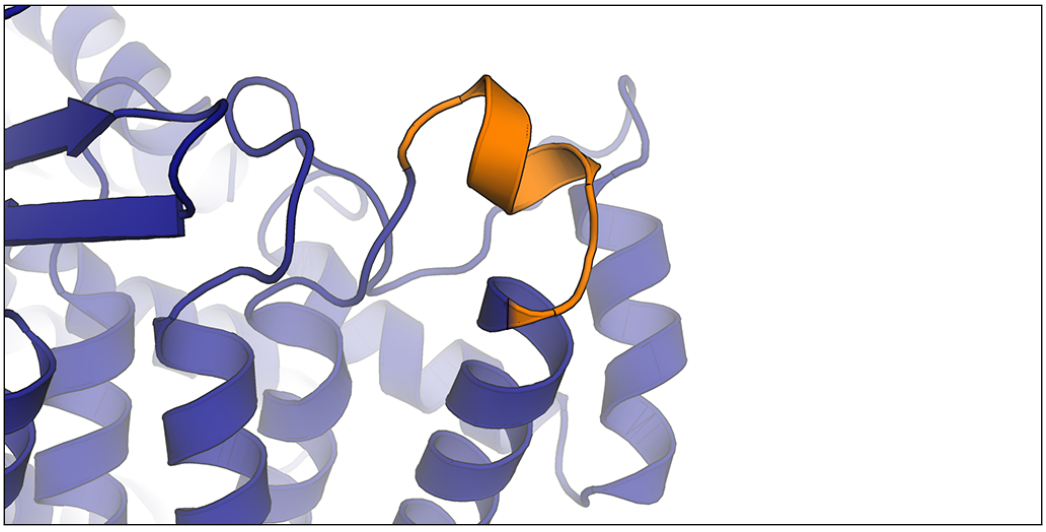
While this case could be improved by filtering based on secondary structures, I have not done this for two reasons. First, the other tools do not assume a purely loop region being modeled, so there would be some risk to giving LoopWeaver an unfair advantage when working with test sets that only contain gaps that correspond to loop regions. Secondly, a strict filter causes poor performance as there are many cases where a good loop candidate has one or two residues of secondary structure at one or both edges, because the division between structures is not always clear cut. DSSP assigns secondary structure type according to specific hydrogen bond energy thresholds. Different cutoffs result in different assignments. Even if one assumes that the gap should only be filled with loop residues, it is not clear how many residues on either end should be allowed to be non-loop to avoid issues with assignment ambiguity.

T0513 residues 79-88

Another example of poor performance can be seen in the length 10 loop from target T0513. Figure 3.12a shows the LoopWeaver and Loopy candidates compared with the native loop. The template that LoopWeaver used (Figure 3.12b) was between a helix and a strand, just as the gap was in the query protein. However, the template involves one additional residue in the helix, and 3 additional residues in the strand. This resulted in a more extended loop, and high RMSD scores of 2.70Å local and 4.54Å global. For reference, the Loopy loop had 0.80Å local RMSD and 1.32Å global RMSD. This may have been resolved by the use of secondary structure filters, but in this case there were no secondary structures occurring in the middle of the template loop, but only as a short continuation of the secondary structures in the stems (although technically the strand was not continued by using this template because there was not a second strand with which to form hydrogen bonds). Because there is no single accepted method of classifying secondary structures it is unreasonable to have a hard cutoff when determining transition points, so it's not certain that any reasonable filters would have worked in this case. Fortunately ROSETTA refinement in this case correctly pulls the extra residue back out of the helix, and the refined loop was very



(a) Native loop (A, Blue), compared with Loopy (B, Pale Green), ModLoop (C, Magenta), and LoopWeaver (D, Orange) candidates. Loop starts with residue 89, the helix and ends with residue 97, at the strand.



(b) Template loop 2H12, chain A, residues 357 through 365 (Orange). Template starts at the helix.

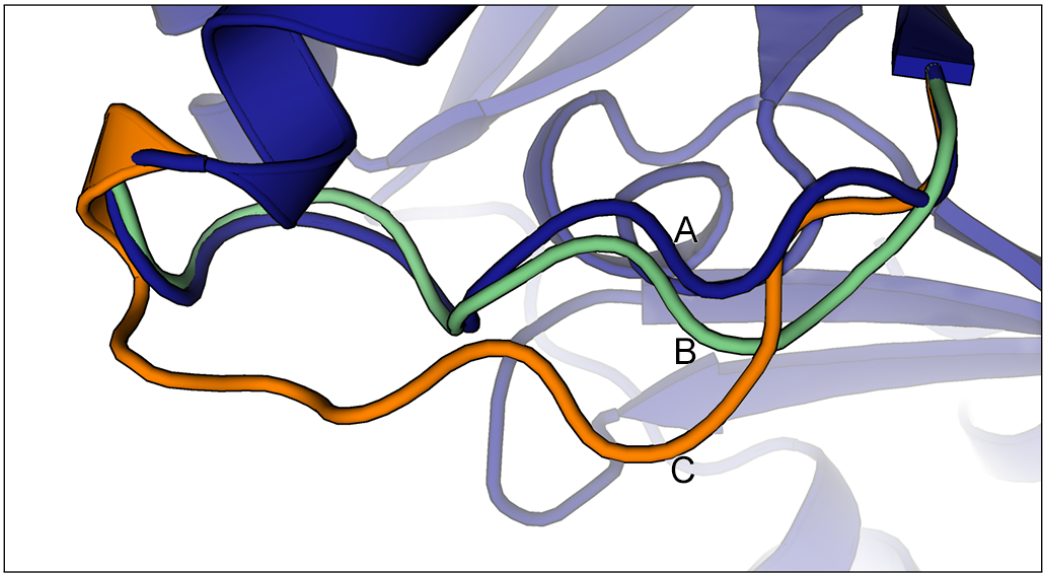
Figure 3.11: Candidates and LoopWeaver template for target T0625, length 9 loop at residues 89 through 97.

accurate, with a local RMSD of 0.36Å and a global RMSD of 0.44Å. The refined Loopy candidate had local and global RMSDs of 0.36Å and 0.47Å respectively. So while Loopy's unrefined prediction was much more accurate than LoopWeaver's, after refinement both candidates were approximately the same structure.

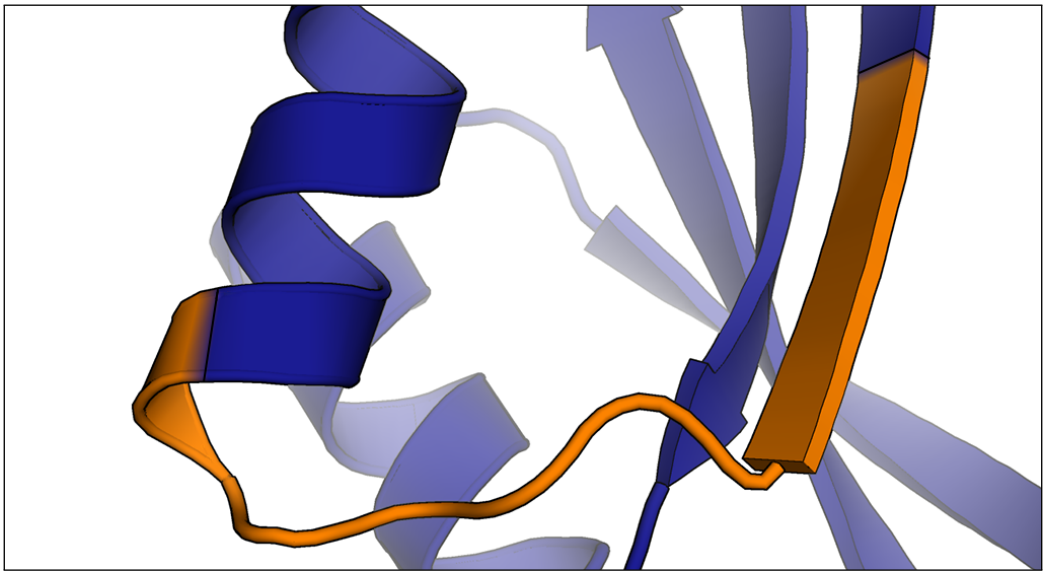
T0490 residues 52-61

A major problem with database methods, including LoopWeaver, is dealing with a constraining environment. Many of the test loops where LoopWeaver made bad predictions involved a dense environment near the loop. This often occurred where one or both stems were part of a β-sheet, where the other strands, and the loops connecting them, were very close to the loop being modeled. Even if a very similar loop existed in the database, it could be used if the small differences cause a clash with the rest of the protein. This means that placement and orientation is very important. LoopWeaver results clashed less often than the same database loops did without WMDS fitting, but for constrained environments many of the candidates still clashed. The second pass WMDS, which attempts to resolve clashes, was able to resolve many such clashes, but at the cost of more changes to the template loop and a worse DFIRE potential energy. In many cases where there were other loops or secondary structures near to the native loop, LoopWeaver did poorly while Loopy did very well. An example of this can be seen in Figure 3.13 In Sub-Figure 3.13a the Loopy loop in green was very close to the native loop in blue. The magenta ModLoop candidate was not as accurate, and the orange LoopWeaver loop was the least accurate. In this example the loop must avoid being too close to the helix on the left (arrow 1) and to the sheet connected to it (arrow 2), as well as to the sheet to the right (arrow 3) and to the loops that connect these structures together (arrows 4 and 5). Many of the database loops were too close to one or more of these features, and were not used. Sub-Figure 3.13b shows the template that LoopWeaver used. Although the template ended at a helix just as in the query, at the other end it ended with another helix rather than a strand, and the template included several turns of this helix. This is not a very probable template, but it is the best scoring candidate that fits properly.

Loopy appears to do very well with a constraining environment, while ModLoop and RAPPER do not appear to get nearly as much benefit from

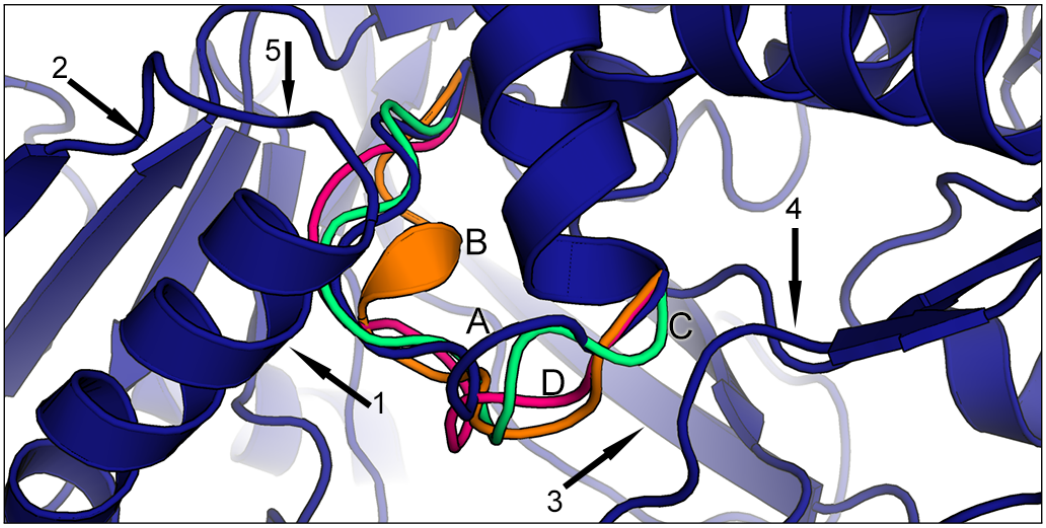


(a) Native loop (A, Blue), compared with Loopy (B, Pale Green) and LoopWeaver (C, Orange) candidates.

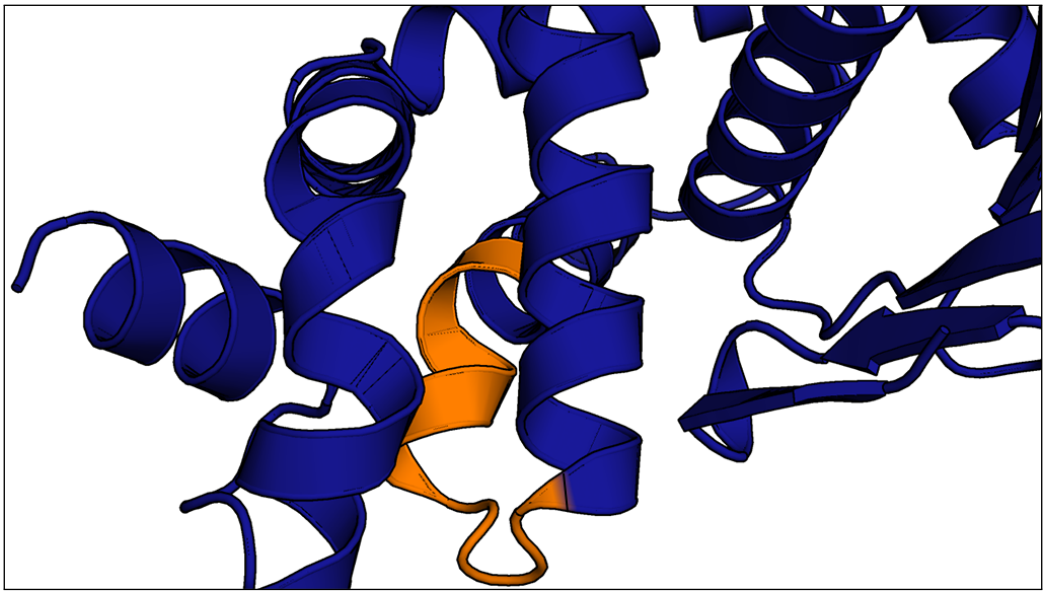


(b) Template loop 2F06, chain A, residues 26 through 35 (Orange). The template loop starts with residue 26 at the helix on the left, and ends with residue 35 most of the way up the rightmost strand.

Figure 3.12: Candidates and LoopWeaver template for target T0513, length 10 loop at residues 79 through 88.



(a) Native loop (A, Blue), compared with Loopy (B, Pale Green) and LoopWeaver (C, Orange) candidates.



(b) Template loop 2GA9, chain D, residues 82 through 91 (Orange). The loop starts at residue 52, two turns back from the end of the rear helix, and ends at residue 61, the first residue of the front and center helix.

Figure 3.13: Candidates and LoopWeaver template for target T0490, length 10 loop at residues 52 through 61.

this situation. The Loopy algorithm involves creating a number of clash free closed loops, and iteratively splicing these loops together to create variants. It appears that in cases where there were fewer possible closed paths, there was a more thorough coverage of the conformation space, and thus the splicing stage achieved a much more complete exploration of the conformation space, and the colony energy function can use this coverage to make a good selection.

This helps explain why Loopy and LoopWeaver formed the most complementary combination of tools. While a restrictive environment makes it difficult to place a suitable database loop, it makes it easier for Loopy to explore the conformation space, and because of these cases there is some correlation between low LoopWeaver accuracy and high Loopy accuracy.

3.6.2 LoopWeaver's Performance over Time

In general there were small improvements to LoopWeaver over time as the result of higher quality matches being available. Additionally, if a close loop homolog was present in the database, it was easily found by its stem RMSD, and selected by its very low energy potential. However, there were cases where the larger database resulted in a worse prediction. This often occurred when a medium or poor quality prediction was replaced by another. This was usually only a small decrease in quality as measured by RMSD, but in a few instances the degradation was substantial.

T0393 residues 62-67

An example of the worst case of LoopWeaver doing poorly with a larger database can be seen in length 6 loop in target T0393, from residue 62 to residue 67. This loop connects a short helix to a beta strand, and runs parallel to two other loops involved in this beta sheet. Using the 2000 database, LoopWeaver selected a loop from protein 1CJC, chain A, starting at residue number 141. Because of the restrictive environment around this loop, few candidates were clash free so LoopWeaver selected a pass 2 candidate. After clash avoidance this database loop had a local and global RMSD of 0.62Å and 0.90Å respectively.. However, when given the 2002, 2003, or 2004 database, LoopWeaver instead selected protein 1JJ2 chain Y at residue number 47. After clash avoidance this loop has local and global RMSDs of 1.36Å and

1.57Å respectively. Using a 2005 or newer database, LoopWeaver again made a worse selection by picking 1LEH chain A, residue number 350. After clash avoidance this loop scored approximately 1.72Å and 2.12Å. These loops can be seen in Figure 3.14. The 2000 loop (orange) follows the native loop closely (blue). The 2002-4 and 2005-2008 loops (pale green, magenta) were not as precise. However, they still follow the path quite closely. Table 3.15 shows the local and global RMSD scores for each year. The clash avoidance algorithm derives the second iteration WMDS instance from the set of non-clashing loop candidates from the first iteration, which is why the same template had different RMSD scores depending on the database date. This also partially explains the inconsistency in selecting candidates, as these changes also altered the DFIRE energy of the candidates.

Another source of noise is that energy functions are only an approximate way of selecting the best loop, and loops do not always have the actual lowest energy conformation[61]. Additionally, local and global RMSD values are only an approximate measure of similarity, so it is not a certainty that selecting a loop with a lower energy score but higher RMSD score is a mistake. As shown in Figure 3.14 the different LoopWeaver solutions follow almost the same path. In any event ROSETTA KIC refinement refined the 1LEHA loop to 0.49Å local and 0.83Å global RMSD, supporting the idea that the most important thing is following a reasonable path. The older matches are likewise refined to a near native conformation.

T0431 residues 453-463

A second example of worsening performance over time can be seen in target T0431, loop from residue 453 to 463 (length 11). For 2000-2006 LoopWeaver selected database protein 1DML chain A, starting at residue 259. After fitting, this loop had local and global RMSD values of 1.62Å and 2.38Å respectively. This is shown as the orange loop in Figure 3.15, where it can be seen following the same general path as the native (blue) loop. When using the 2006 database (and again when using the 2007 database), LoopWeaver instead selected 2GB3 chain A, at residue 35, as its database loop. This loop scored 2.51Å and 4.50Å, and is shown in pale green, where it can be seen extending much further out into solvent. This is a case where the performance of LoopWeaver dropped substantially as a direct result of a larger database. Unlike the first example, this was not caused by a loop with better energy but

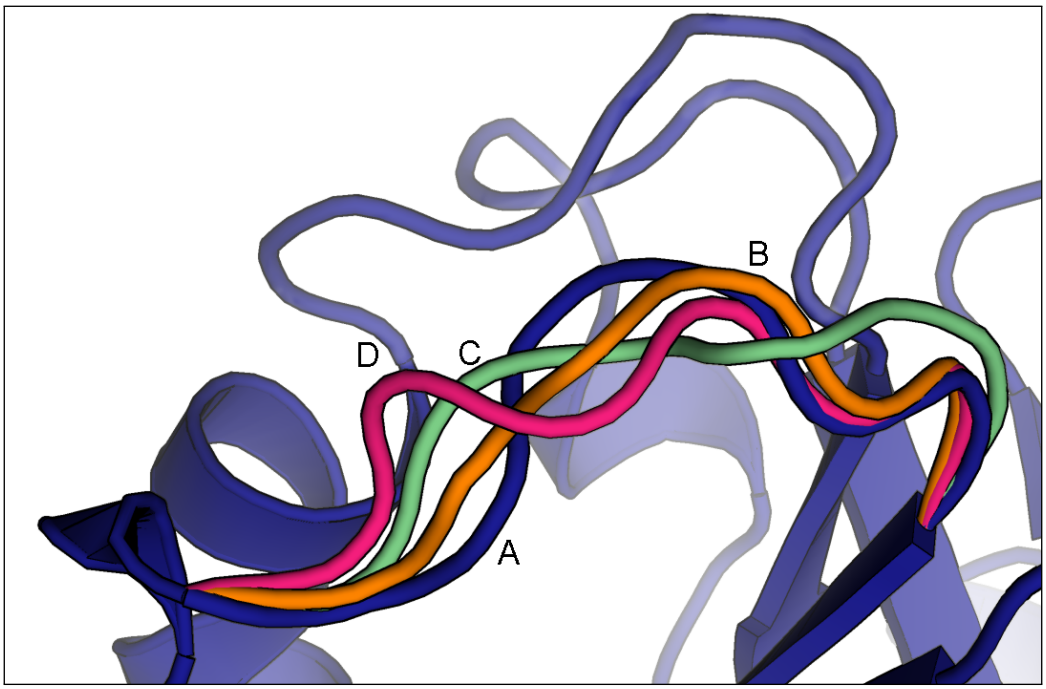


Figure 3.14: LoopWeaver results for CASP8 target T0393, loop from residue 62 (left) to 67 (right)

Native structure, including native loop (A). In orange is the LoopWeaver 2000 loop (B). In pale green is the LoopWeaver 2002 loop (C). In magenta is the LoopWeaver 2005 loop (D).

Table 3.15: LoopWeaver results for target T0393, loop at residues 62-67 vs database year.

Year	Local RMSD	Global RMSD	Template [a]		
2000	0.63	0.90	1CJC	A	141-146
2001	1.14	1.48	2TNF	A	142-147
2002	1.36	1.57	1JJ2	Y	47-52
2003	1.35	1.54	1JJ2	Y	47-52
2004	1.36	1.57	1JJ2	Y	47-52
2005	1.72	2.12	1LEH	A	350-355
2006	1.72	2.16	1LEH	A	350-355
2007	1.72	2.15	1LEH	A	350-355
2008	1.72	2.14	1LEH	A	350-355

[a]PDB ID, Chain ID, Residue IDs

worse RMSD being introduced. The 1DML loop after fitting has a DFIRE energy potential of -1002.4, while the 2GBA loop had a worse score of -909.4. The reason the higher scoring loop was no longer used is that it was not in the top 500 by $\text{RMSD}_{\text{stem}}$. That is, a larger database eventually resulted in too many medium and poor results to identify this particular medium candidate. This does not appear to occur often in shorter loops. It may be necessary to increase the number of matches selected when dealing with longer loops, just as it is in other approaches. However, the 2008 loop used is from protein 3B6H chain A, and it has loop scores of 0.64Å local and 1.24Å global RMSD. The DFIRE energy of this new loop is -930.232, higher than the -1002.4 of the 1DML loop. In this case, for the full (2008) database, increasing to the number of matches used would have resulted in selecting the less accurate 1DML database loop.

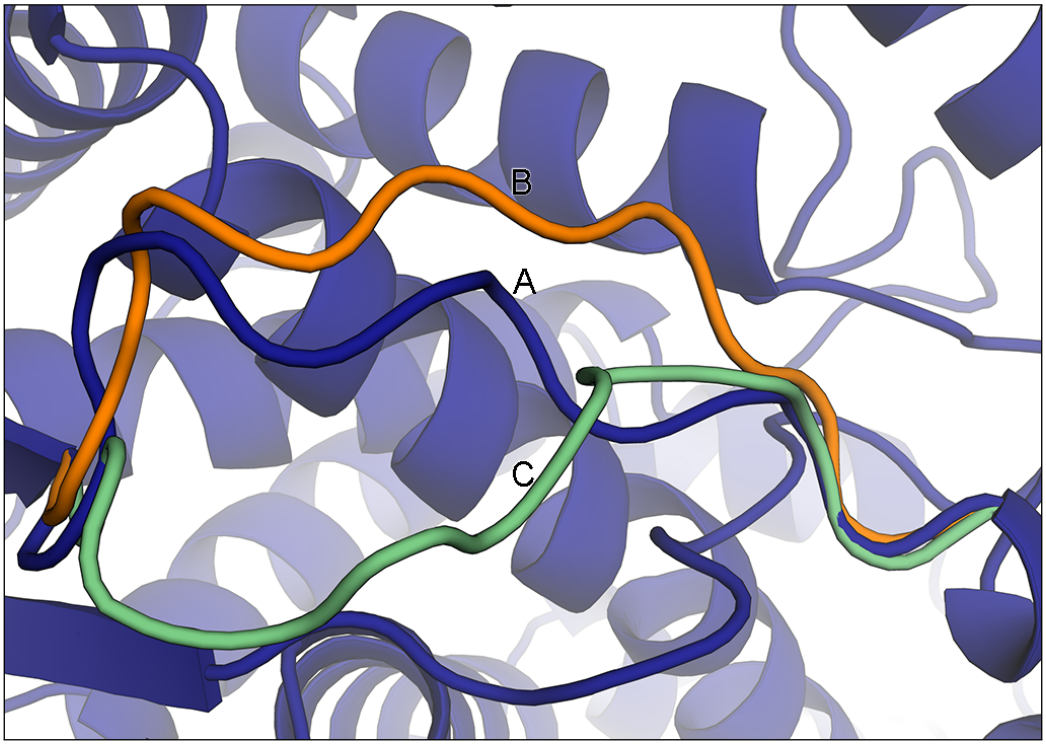


Figure 3.15: LoopWeaver Results for CASP8 target T0431, loop from residue 453 (left) to 463 (right)

Native structure in dark blue, including native loop, labeled (A). The 2000-2003 database loop in orange and labeled (B). The 2006-2007 loop in pale green and labeled (C).

Table 3.16: LoopWeaver results for target T0431, loop from residue 453 to 463, over time.

Year	Local RMSD	Global RMSD	Template [a]		
2000	1.62	2.38	1DML	A	259-269
2001	1.62	2.38	1DML	A	259-269
2002	1.62	2.38	1DML	A	259-269
2003	1.62	2.38	1DML	A	259-269
2004	2.01	2.95	1EGJ	H	131-141
2005	2.15	3.55	1FP5	A	532-532
2006	2.51	4.50	2GB3	A	35-45
2007	2.51	4.61[b]	2GB3	A	35-45
2008	0.66	1.23	3B6H	A	467-477

[a] PDB ID, Chain ID, Residue IDs

[b] this is different from the above value due to the clash avoidance step

3.7 Clash Avoidance

Although database matches were only selected if the initial placement did not clash with the surrounding environment, sometimes many of them clashed after the WMDS fitting. LoopWeaver performed a second round of WMDS fitting in an to attempt to avoid these clashes (Section 2.2 details this step) and although it did eliminate clashes, it was always possible to do so for all candidates. Table 3.17 details the clash statistics for the different test sets. We can see that on average at least 400 of the 500 matches were clash free, and the median was even higher, suggesting that there were only a few very difficult cases. This is reflected by the minimum values being very low. However, a large number of clashes was not always a sign of poor predictions. A database loop that is very similar and comes from a similar environment will not clash, so difficulty fitting loops in a clash-free manner does not always imply that the few loops that do fit will be of low quality. For the set of length 7 loops, the most difficult loop had only had only 90 matches (out of 500) that LoopWeaver fitted without clashes. However, the protein that contains this loop has a close homolog in the database, and because of this, one of the 90 matches was almost identical to the native loop (0.14Å local RMSD, 0.19Å global RMSD). Although this one result was the most dramatic, overall there was no correlation between the number of clashes and the accuracy of the

Table 3.17: Number of clashing candidates before and after clash avoidance.

	Before			After		
Length	Min	Mean	Median	Min	Mean	Median
6	90	413.31	445.0	123	475.02	498.0
7	90	407.08	442.5	114	462.70	494.5
8	39	367.99	424.0	102	459.77	495.0
9	27	384.56	421.0	84	467.65	494.0
10	42	342.22	362.5	179	452.48	490.5
11	9	286.00	289.0	40	407.52	480.5

prediction. The most difficult loop was from CASP9 target T0585 between residues 83 to 93 (length 11), which had only 9 matches. Clash avoidance improved this to 230, so almost half of the clashes were resolved. However, none of second group of loop candidates had lower energy than the candidate that was selected. This match had RMSD scores of 2.18Å local and 2.18Å global, close to the average local RMSD score and better than the average global RMSD score.

The second pass WMDS to avoid clashes consistently increased the number of usable matches. When we look at the median value, the number of clash-free matches was almost 500, so in most situations the clash avoidance step was able to resolve almost all clashes. However, there was no significant change to the testing scores if this step was skipped, as the pass two candidates were worse as often as they were better. It is possible that with further modifications this step could avoid clashes with less negative impact on the quality of the loop, by my current method does not appear useful outside of cases where there are few usable matches without it.

Chapter 4

Summary and Outlook

I have demonstrated a new way of generating closed loop candidates using a database of proteins with known structure. By the application of the weighted multi-dimensional scaling problem, LoopWeaver is able to fit database loops into the target protein in a way that is much more energetically favourable than previous approaches. This means that energy functions are much more able to accurately rank the resulting loop candidates. These small changes have little effect on the shape of the loop itself, only the bond and dihedral angles where it is placed into the gap, so it remains as effective as other database techniques for selecting high quality candidates when they are available. While the angles and bond distances are still not entirely realistic, and so in any event refinement is required for truly realistic predictions, LoopWeaver's rapid refinement process allows for a superior initial selection, so that only one candidate must be refined through more time intensive approaches such as ROSETTA's KIC refinement.

I have also demonstrated that while database loops are best used as a complement to statistical results instead of a replacement, with proper fitting and refinement by LoopWeaver these loops are an adequate sampling of the conformational space. LoopWeaver's predictions are superior to most methods of loop modeling, and are approximately ten times as fast as all tools but Loopy. When using ROSETTA to do final refinement of loop candidates LoopWeaver's results are no longer the best, but are still comparable. However, these database loops remain highly complementary to the statistical results and so the combined LoopWeaver and statistical results are superior to either method alone, both before and after refinement.

Although LoopWeaver's current implementation is producing acceptable results, these results are not as accurate as they could be. In all of the cases where LoopWeaver produces medium or worse quality predictions, there are a number of loop candidates generated with better scores, but which were not selected by DFIRE. It is not clear if other energy functions would improve this situation. I have examined several energy functions besides DFIRE, and found that they all perform the same or worse. For example, although the RWP paper[62] shows large improvements over the DFIRE function for full protein structure prediction, I do not observe the same improvement when comparing loops. The Dipolar DFIRE (DDFIRE)[63] also showed improved accuracy for proteins in general, but did not work as well for loop selection. It is often the case that loops do not always take on the most energetically favourable conformation[33], so it is not entirely unexpected that the most effective energy function for whole proteins may not be the most effective for loops.

This ranking problem may also be the result of the sidechain predictions used. DFIRE is sensitive to the accuracy of the sidechain predictions made. RWP also is quite dependent on sidechain accuracy. Although TreePack is the best tool that I have tried in terms of both speed and the accuracy of the final loop candidate ranking, it is not in general regarded as the best method of sidechain prediction. So all combinations of sidechain packing and energy functions must be tested in order to obtain a complete analysis. We saw earlier that there is substantial difference between the minimum average RMSD scores, and the average RMSD scores obtained by DFIRE selection. Thus there is room for improvement with respect to the energy function used.

As LoopWeaver's database used x-ray models from the PDB, it may be possible to improve its matching by incorporating the B-Value from these models. The B-Value measures both the disorder of the region and the error in the model. As such, it may be useful to rank matches based on a combination of stem RMSD and B-Value, rather than on stem RMSD alone. However, it is difficult to separate the model error and disorder components from the B-Value, and one would not want to discard matches with low error but high disorder as one may be modeling a loop that has a similarly high disorder. This makes the approach less than straightforward.

The LoopWeaver technique, as a database approach, is limited by the accuracy of the rest of the protein, especially the stems. Because LoopWeaver adjusts the atomic coordinates of the database match, it is less likely to

encounter situations where it cannot place a loop into the gap as a result of incorrect stems, but cannot always do so if the stems are very unusual. If the stems of the gap are either too close together or too far apart, it is difficult or impossible to connect them together. This is true for statistical approaches as well, but statistical tools can sample unusual conformations and database tools can only use conformations in the database. A standard approach, used both in the database based tool FREAD, and the statistical tool ROSETTA, is to extend the gap outward until realistic loop candidates can be generated. This approach works better for database tools than it does for statistical tools; as searching the database is very fast, the tool can quickly tell if an extension of the gap results in sufficient database matches. A statistical tool would rely on heuristics for deciding how much extension is required for realistic loop modeling. FREAD has no trouble with this situation as typically all matches found are highly accurate because of the strong filters used. LoopWeaver will require a more careful examination of the database matches in order to determine whether a given gap extension is usable.

Further, although multi-dimensional scaling works well in these experiments, it is not the only way to solve sets of distance constraints. For example, semidefinite programming has been used successfully for protein docking prediction [64], although by using database matches, LoopWeaver has a full set of constraints so this approach may not yield many advantages over multi-dimensional scaling. The random and direct tweak algorithms used in Loopy to close and refine statistically generated loops also are based on satisfying sets of distance constraints, so it may also be possible to adapt these algorithms to place adjusted database loops into the gap.

The parameters used for the multi-dimensional scaling step were derived empirically, based on the desire for weights to be inversely proportional to the expected amount of deviation observed between similar loops. Currently no work has been done on confirming that these values yield the best results possible. Indeed, minor changes to the parameters can have a large impact on the final scores. Changes to the parameters have almost no effect on the loops themselves, so their effect on the final score is almost entirely due to the sidechain packings and energy potentials. If these parameters are trained in order to produce the best scores before ranking, it may not translate to improved scores after ranking because of the sidechain and energy function issues.

The method of clash avoidance is also quite *ad hoc* and only marginally effective. For atoms that are often interfering with loop placement, desired distances are estimated by examining the high scoring non-clashing candidates using a backbone-only DFIRE score and taking the average pairwise distances to loop atoms. While this is effective at preventing clashes with these atoms, it results in large changes to the shape and orientation of the loop. Currently the same estimated distance is used for all matches when doing clash avoidance. Some improvement may be had by instead clustering similar non-clashing loops. The pairwise distances for new atoms in the second formulation could then be based on the average distances from the cluster each candidate belongs to, instead of the average over all non-clashing candidates. Although clashes are uncommon for most of the loop regions tested, cases where loops pass close to each other are the most difficult for LoopWeaver to predict because any accurate prediction has a good chance of clashing with the nearby loop. Presently the clash avoidance algorithm is interesting in that it largely avoids clashes, but it is not practical as these loops are not useful unless no alternatives are present. In the test sets I used, there were always alternatives present so the clash avoidance step did not improve the average scores.

Copyright Permissions

Figure 1.1 is authored by Yassine Mrabet[1] who has granted all people the unconditional right to use this image for any purpose. Figure 1.2 is based on this work.

Figure 1.3 is authored by LadyOfHats[2] who has granted all people the unconditional right to use this image for any purpose.

Figure 1.4 is reproduced from Liu and Altman 2009[3] figure 1, in compliance with the Creative Commons Attribution License (`http://creativecommons.org/licenses/by/2.0`)

Bibliography

[1] YassineMrabet, "Structure gnrale d'un acide amin.." `http://upload.wikimedia.org/wikipedia/commons/archive/c/ce/20070812164301!AminoAcidball.svg`, Aug 2007.

[2] LadyOfHats, "Main protein structure levels." `http://en.wikipedia.org/wiki/File:Main_protein_structure_levels_en.svg`, Oct 2008.

[3] T. Liu and R. B. Altman, "Prediction of calcium-binding sites by combining loop-modeling with machine learning," *BMC Struct. Biol.*, vol. 9, p. 72, 2009.

[4] R. A. Engh and R. Huber, "Accurate bond and angle parameters for X-ray protein structure refinement," *Acta Crystallographica Section A*, vol. 47, pp. 392–400, Jul 1991.

[5] C. Brändén and J. Tooze, *Introduction to Protein Structure*. Garland Publishing, Incorporated, 1999.

[6] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers*, vol. 22, no. 12, pp. 2577–2637, 1983.

[7] G. N. RAMACHANDRAN, C. RAMAKRISHNAN, and V. SASISEKHARAN, "Stereochemistry of polypeptide chain configurations," *J. Mol. Biol.*, vol. 7, pp. 95–99, Jul 1963.

[8] K. Wuthrich, "The way to nmr structures of proteins," *Nat Struct Mol Biol*, vol. 8, pp. 923–925, Nov 2001.

[9] T. U. Consortium, "Reorganizing the protein space at the universal protein resource (uniprot)," *Nucleic Acids Research*, vol. 40, no. D1, pp. D71–D75, 2012.

[10] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, 2000.

[11] A. Yonath, "X-ray crystallography at the heart of life science," *Current Opinion in Structural Biology*, vol. 21, no. 5, pp. 622 – 626, 2011.

[12] "Current release statistics < uniprotkb < embl-ebi." Available at: `http://www.ebi.ac.uk/uniprot/TrEMBLstats` Accesed Jun. 7, 2013.

[13] "Rcsb pdb - content growth report." Available at: `http://www.pdb.org/pdb/statistics/contentGrowthChart.do?content=total&seqid=100` Accessed Jun. 7, 2013.

[14] P. E. Leopold, M. Montal, and J. N. Onuchic, "Protein folding funnels: a kinetic approach to the sequence-structure relationship.," *Proceedings of the National Academy of Sciences*, vol. 89, no. 18, pp. 8721–8725, 1992.

[15] P. Bradley, K. M. S. Misura, and D. Baker, "Toward high-resolution de novo structure prediction for small proteins," *Science*, vol. 309, no. 5742, pp. 1868–1871, 2005.

[16] B. Qian, S. Raman, R. Das, P. Bradley, A. J. McCoy, R. J. Read, and D. Baker, "High-resolution structure prediction and the crystallographic phase problem," *Nature*, vol. 450, pp. 259 – 264, Nov 2007.

[17] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia, "Scop: A structural classification of proteins database for the investigation of sequences and structures," *Journal of Molecular Biology*, vol. 247, no. 4, pp. 536 – 540, 1995.

[18] I. Sillitoe, A. L. Cuff, B. H. Dessailly, N. L. Dawson, N. Furnham, D. Lee, J. G. Lees, T. E. Lewis, R. A. Studer, R. Rentzsch, C. Yeats, J. M. Thornton, and C. A. Orengo, "New functional families (FunFams) in CATH to improve the mapping of conserved functional sites to 3D structures," *Nucleic Acids Res.*, vol. 41, pp. D490–498, Jan 2013.

[19] R. H. Lathrop, "The protein threading problem with sequence amino acid interaction preferences is NP-complete," *Protein Eng.*, vol. 7, pp. 1059–1068, Sep 1994.

[20] J. Soding, A. Biegert, and A. N. Lupas, "The HHpred interactive server for protein homology detection and structure prediction," *Nucleic Acids Res.*, vol. 33, pp. W244–248, Jul 2005.

[21] J. Xu, M. Li, D. Kim, and Y. Xu, "RAPTOR: optimal protein threading by linear programming," *J Bioinform Comput Biol*, vol. 1, pp. 95–117, Apr 2003.

[22] M. Kallberg, H. Wang, S. Wang, J. Peng, Z. Wang, H. Lu, and J. Xu, "Template-based protein structure modeling using the RaptorX web server," *Nat Protoc*, vol. 7, pp. 1511–1522, Aug 2012.

[23] A. Fiser and A. Sali, "Modeller: Generation and refinement of homology-based protein structure models," in *Macromolecular Crystallography, Part D* (J. Charles W. Carter and R. M. Sweet, eds.), vol. 374 of *Methods in Enzymology*, pp. 461 – 491, Academic Press, 2003.

[24] A. Kryshtafovych, C. Venclovas, K. Fidelis, and J. Moult, "Progress over the first decade of casp experiments.," *Proteins*, vol. 61 Suppl 7, no. S7, pp. 225–236, 2005.

[25] Y. Zhang and J. Skolnick, "Scoring function for automated assessment of protein structure template quality," *Proteins: Structure, Function, and Bioinformatics*, vol. 57, no. 4, pp. 702–710, 2004.

[26] A. Zemla, C. Venclovas, J. Moult, and K. Fidelis, "Processing and evaluation of predictions in casp4," *Proteins: Structure, Function, and Bioinformatics*, vol. 45, no. S5, pp. 13–21, 2001.

[27] A. Zemla, "Lga: a method for finding 3d similarities in protein structures," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3370–3374, 2003.

[28] J. C. Whisstock and A. M. Lesk, "Prediction of protein function from protein sequence and structure," *Quarterly Reviews of Biophysics*, vol. 36, pp. 307–340, 7 2003.

[29] E. T. Kool, "Active site tightness and substrate fit in dna replication.," *Annual review of biochemistry*, vol. 71, pp. 191 – 219, Nov 200.

[30] Z. Szeltner, T. Juhasz, I. Szamosi, D. Rea, V. Fulop, K. Modos, L. Juliano, and L. Polgar, "The loops facing the active site of prolyl oligopeptidase are crucial components in substrate gating and specificity.," *Biochimica et biophysica acta*, vol. 1834, pp. 98 – 111, Jan 2000.

[31] C. Smith and I. Rayment, "Active site comparisons highlight structural similarities between myosin and other p-loop proteins," *Biophysical Journal*, vol. 70, no. 4, pp. 1590 – 1602, 1996.

[32] J. Greenwald, V. Le, S. L. Butler, F. D. Bushman, and S. Choe, "The mobility of an hiv-1 integrase active site loop is correlated with catalytic activity," *Biochemistry*, vol. 38, no. 28, pp. 8892–8898, 1999.

[33] A. Fiser, R. Do, and A. Sali, "Modeling of loops in protein structures," *Protein Science*, vol. 9, pp. 1753–1773, September 2000.

[34] J. E. Walker, M. Saraste, M. J. Runswick, and N. J. Gay, "Distantly related sequences in the alpha- and beta-subunits of ATP synthase, myosin, kinases and other ATP-requiring enzymes and a common nucleotide binding fold," *EMBO J.*, vol. 1, no. 8, pp. 945–951, 1982.

[35] L. Wei and R. B. Altman, "Recognizing complex, asymmetric functional sites in protein structures using a Bayesian scoring function," *J Bioinform Comput Biol*, vol. 1, pp. 119–138, Apr 2003.

[36] Z. Xiang, C. S. Soto, and B. Honig, "Evaluating conformational free energies: The colony energy and its application to the problem of loop prediction," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 11, pp. 7432–7437, 2002.

[37] P. S. Shenkin, D. L. Yarmush, R. M. Fine, H. Wang, and C. Levinthal, "Predicting antibody hypervariable loop conformation. i. ensembles of random conformations for ringlike structures," *Biopolymers*, vol. 26, no. 12, pp. 2053–2085, 1987.

[38] C. S. S. Soto, M. Fasnacht, J. Zhu, L. Forrest, and B. Honig, "Loop modeling: sampling, filtering, and scoring.," *Proteins*, August 2007.

[39] Z. Xiang and B. Honig, "Extending the accuracy limits of prediction for side-chain conformations," *Journal of Molecular Biology*, vol. 311, no. 2, pp. 421 – 430, 2001.

[40] H. Zhou and Y. Zhou, "Distance-scaled, finite ideal-gas reference state improves structure-derived potentials of mean force for structure selection and stability prediction," *Protein Science*, vol. 11, pp. 2714–2726, November 2002.

[41] K. Zhu, D. L. Pincus, S. Zhao, and R. A. Friesner, "Long loop prediction using the protein local optimization program," *Proteins: Structure, Function, and Bioinformatics*, vol. 65, pp. 438–452, November 2006.

[42] P. I. W. de Bakker, M. A. DePristo, D. F. Burke, and T. L. Blundell, "Ab initio construction of polypeptide fragments: Accuracy of loop decoy discrimination by an all-atom statistical potential and the amber force field with the generalized born solvation model," *Proteins: Structure, Function, and Bioinformatics*, vol. 51, no. 1, pp. 21–40, 2003.

[43] A. A. Canutescu, A. A. Shelenkov, and R. L. Dunbrack, "A graph-theory algorithm for rapid protein side-chain prediction," *Protein Science*, vol. 12, no. 9, pp. 2001–2014, 2003.

[44] A. A. Canutescu and R. L. Dunbrack, "Cyclic coordinate descent: A robotics algorithm for protein loop closure," *Protein Science*, vol. 12, no. 5, pp. 963–972, 2003.

[45] D. J. Mandell, E. A. Coutsias, and T. Kortemme, "Sub-angstrom accuracy in protein loop reconstruction by robotics-inspired conformational sampling," *Nat. Methods*, vol. 6, no. 8, pp. 551–552, 2009.

[46] E. A. Coutsias, C. Seok, M. J. Wester, and K. A. Dill, "Resultants and loop closure," *International Journal of Quantum Chemistry*, vol. 106, no. 1, pp. 176–189, 2006.

[47] C. M. Deane and T. L. Blundell, "Coda: A combined algorithm for predicting the structurally variable regions of protein models," *Protein Science*, vol. 10, pp. 599–612, March 2001.

[48] Y. Choi and C. M. Deane, "Fread revisited: Accurate loop structure prediction using a database search algorithm," *Proteins: Structure, Function, and Bioinformatics*, vol. 78, pp. 1431–1440, May 2010.

[49] E. Michalsky, A. Goede, and R. Preissner, "Loops In Proteins (LIP)-a comprehensive loop database for homology modelling," *Protein Engineering*, vol. 16, no. 12, pp. 979–985, 2003.

[50] P. W. Hildebrand, A. Goede, R. A. Bauer, B. Gruening, J. Ismer, E. Michalsky, and R. Preissner, "SuperLoopera prediction server for the modeling of loops in globular and membrane proteins," *Nucleic Acids Research*, vol. 37, no. suppl 2, pp. W571–W574, 2009.

[51] J. de Leeuw, "Applications of Convex Analysis to Multidimensional Scaling," in *Recent Developments in Statistics* (J. Barra, F. Brodeau, G. Romier, and B. van Cutsem, eds.), pp. 133–146, North Holland Publishing Compant, 1977.

[52] J. Xu, "Rapid protein side-chain packing via tree decomposition," in *Research in Computational Molecular Biology* (S. Miyano, J. Mesirov, S. Kasif, S. Istrail, P. Pevzner, and M. Waterman, eds.), vol. 3500 of *Lecture Notes in Computer Science*, pp. 423–439, Springer Berlin / Heidelberg, 2005.

[53] J. Xu and B. Berger, "Fast and accurate algorithms for protein side-chain packing," *J. ACM*, vol. 53, pp. 533–557, July 2006.

[54] G. Wang and R. L. Dunbrack, "PISCES: a protein sequence culling server," *Bioinformatics*, vol. 19, no. 12, pp. 1589–1591, 2003.

[55] J. Zhang, Q. Wang, B. Barz, Z. He, I. Kosztin, Y. Shang, and D. Xu, "Mufold: A new solution for protein 3d structure prediction," *Proteins: Structure, Function, and Bioinformatics*, vol. 78, pp. 1137–1152, April 2010.

[56] J. D. Cohen, "Drawing graphs to convey proximity: an incremental arrangement method," *ACM Trans. Comput.-Hum. Interact.*, vol. 4, pp. 197–229, September 1997.

[57] W. Basalaj, "Proximity visualization of abstract data," tech. rep., University of Cambridge Computer Laboratory, 2001.

[58] C. Zhang, S. Liu, and Y. Zhou, "Accurate and efficient loop selections by the dfire-based all-atom statistical potential," *Protein Science*, vol. 13, no. 2, pp. 391–399, 2004.

[59] D. J. Mandell and R. A. Pache, "Rosetta projects: Documentation for kinematic loop modeling." Available at: `http://rosettacommons.org/manuals/archive/rosetta3.3_user_guide/app_kinematic_loopmodel.html` Accessed Dec. 1, 2011, October 2011.

[60] J. Lee, D. Lee, H. Park, E. A. Coutsias, and C. Seok, "Protein loop modeling by using fragment assembly and analytical loop closure," *Proteins: Structure, Function, and Bioinformatics*, vol. 78, no. 16, pp. 3428–3436, 2010.

[61] K. C. Smith and B. Honig, "Evaluation of the conformational free energies of loops in proteins," *Proteins: Structure, Function, and Bioinformatics*, vol. 18, no. 2, pp. 119–132, 1994.

[62] J. Zhang and Y. Zhang, "A novel side-chain orientation dependent potential derived from random-walk reference state for protein fold selection and structure prediction," *PLoS ONE*, vol. 5, p. e15386, 10 2010.

[63] Y. Yang and Y. Zhou, "Specific interactions for ab initio folding of protein terminal regions with secondary structures," *Proteins*, vol. 72, pp. 793–803, Aug 2008.

[64] Y. Shen, I. C. Paschalidis, P. Vakili, and S. Vajda, "Protein docking by the underestimation of free energy funnels in the space of encounter complexes," *PLoS Comput Biol*, vol. 4, p. e1000191, 10 2008.