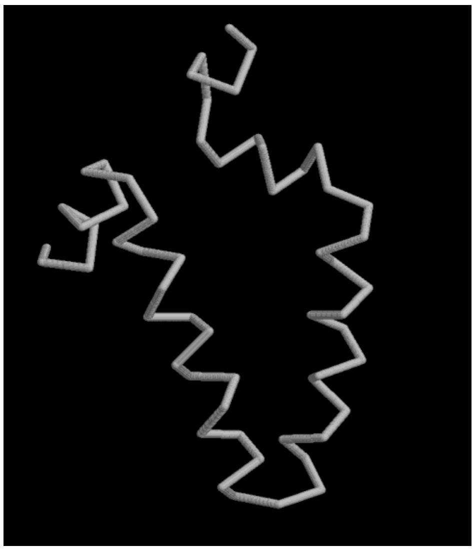
CHALMERS



A System that Builds Decoy Protein Backbones by Assembling Smaller Fragments

Master of Science Thesis in the Programme Bioinformatics and Systems Biology

FARZANA RASHID

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

A System that Builds Decoys Protein Backbone by Assembling Smaller Fragments

FARZANA RASHID

Examiner: GRAHAM J. L. KEMP

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: A RASMOL image of a decoy structure generated using the system built in the project]

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

The project was focused on designing a system that can build protein-like decoy structures using a dynamic programming algorithm. The system first builds three-residue-long polymer fragments, which are then used to build longer fragments, finally leading to large polymer chains of the same length as a given sequence. The decoys formed by the systems are free of any steric clashes.

Acknowledgement

First of all I would like to thank my supervisor, Dr. Graham Kemp whose expert guidance had made this project possible. Special thanks to him for hours of sessions on how to design the algorithm. Special thanks to him for the pieces of code that he had allowed me to use in this project. Special thanks to him for his endless patience, everlasting effort and profound encouragement to steer me through my work.

I would like to thank my husband, Syed Fakhruzzaman for sincerely supporting me all through my master studies. I would like to thank my parents Mr. A. K. Rashiduddin Ahmed and Mrs. Nigar Rashid, my parents-in-law Mr. Syed Md. Hasanuzzaman and Mrs. Ferdousi Zaman, all other family members, and my friends for being there beside me.

I would like to say thanks to my program coordinator, Prof. Olle Nerman for his support and guidance and for giving me the opportunity to acquire this degree.

Table of Contents

1 Introduction

Proteins are essential molecules in living organisms as they are involved in carrying out the vital functions of the cells. Proteins occur as enzymes that are necessary to catalyze the different reactions involved in metabolism. Proteins are also involved in mechanical activities within the cell, including the transfer of other molecules. They are important for providing support to maintain the shapes of the cells.

An extremely important problem in computational biology, which is still to be solved, is prediction of protein structure given its sequence. Many different algorithms exploring the possibilities of solving this problem have been tried out. Given a sequence of protein, most of these methods generate protein like structures called decoys which are considered as candidate structures of the natural form. These candidate structures are then evaluated with some scoring function which evaluates a score for the decoy structure. This evaluated score indicates strength of such a decoy in nature. These scoring functions in some cases are based on the energy state of the structure. It can be simultaneously or independently based on many other factors. One such factor is the positions of different non-polar (hydrophobic) and polar (hydrophilic) residues and the distance among such residues. In some cases, the scoring is simply done by preferring structures which do not have residues which are too close to others in the chain. That is chains without any steric clashes are preferred because during folding, naturally protein molecules avoid steric clashes between the constituent residues. In some cases the scoring function is based on how compact a formed decoy is, that is on the size.

Sometimes, assembling the decoys efficiently using suitable data structures becomes a challenge. This challenge was explored in this thesis, and the resulting decoys were checked for presence of steric clashes.

Figure 1.1 shows the structure of a protein drawn using, RASMOL, a program used for viewing the structures of proteins.

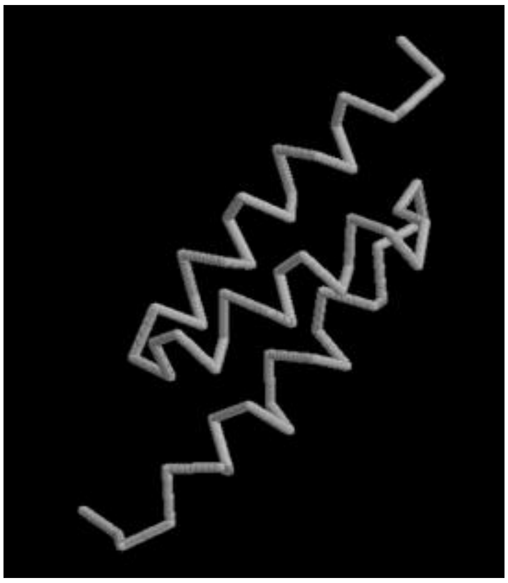


Figure 1.1 The image of a protein structure drawn using RASMOL

1.1 Motivation

Protein carry out many important functions in the cells and their functions can be carried out only if they are folded in a particular three dimensional structure **(**Pietzsch, 2003).

Right after a ribosome decodes a genetic instruction found in the DNA of a cell, and forms an amino acid chain, that chain readily folds or coils into a comfortable conformation in order to reduce the amount of free energy. Although there are a large number of possible conformations, a protein folds into its unique stable structure much faster than it would take to go through all these possible conformation just by trial and error. Scientists have always been fascinated to understand why these proteins fold into their stable structure so rapidly (Lobanov, et al., 2008). They have tried to learn or develop methods that can predict the possible structures of a certain sequence of residues that would instantly produce in the nature. If a sequence of a protein is newly found and its natural structure is still not known, then the structure predicted by an automatic method can help researchers predict the function of the protein. It can help in comparison of functions of proteins with similar structures, and can ultimately help in genome annotation (Hamelryck, et al.,2006).

Drugs work by combining with or by deactivating certain proteins in the cells. Thus, knowing the structure of proteins involved in a certain disease can help drug designers to design certain drugs. By learning how the proteins fold rapidly in their native structure, scientists also learn to engineer new proteins, which can have many varied applications in medical field (Lobanov, et al., 2008).

Inability of a protein to fold in the right form can lead to its malfunction and can thus lead to diseases. So, understanding of the structures of proteins can also put some light on the causes of certain diseases. The causes of deformation of protein in the nature can also be understood better, if methods of predicting correct proteins structures can be designed.

1.2 Aims and Scope

The focus of my project was to build a system that can take a known protein sequence and the information about naturally occurring geometries of that protein, to assemble a number of protein-like structures or decoys, which do not have any steric clashes. The system that I have built uses dynamic programming techniques of breaking down the large problem of building a long decoy into smaller problems of building small fragments, keeping the solutions of these small problems in a matrix structure, and assembling some randomly picked solutions to get to the final solution.

The system builds up the decoys by first breaking up the sequence into fragments of three residues, and then assembling different three-residue-long fragments to form four-residue-long ones, then so on an so fourth until a few structures of the entire length are assembled. At every stage of building the structures, a certain number of fragment solutions are assembled and saved. The constituent of these fragments are smaller fragments which are randomly picked. Each of these fragments has a different structure, as the angles subtended by the consecutive residues in the structures are different. Once each such fragment is formed, it is checked for presence of any steric clash, i.e. the presence of residues that are placed too close together. In case of positive result of this checking, that structure is discarded, and another structure is predicted in its place again by picking the composing fragments randomly. The details of all these fragments being built are saved in a very convenient data structure. The data structure is a triangular array built on a one-dimensional array which will be described later.

The main goal of the project was to end up with a system which can generate some candidate decoy structures that would have geometries similar to naturally found proteins. The system builds decoy structures by using the geometries from known protein structures.

1.3 Thesis Overview

This report first explains some background information of some important concepts that are explored and used in this project work. Discussions on some of these important terms are placed in Chapter 2.

Then in Chapter 3, the discussion moves on to how others have tried to develop systems to build protein-like structures. The chapter explains how the systems built by others work, what are the purposes of those systems, and what algorithms have been used in developing these systems. The chapter also discusses how certain important concepts were learnt from some related works.

In Chapter 4 of the report, I explain how the system that I have built works and what is its purpose. There I discuss the algorithm in detail, and simultaneously try to give a clear view of the data structures that are used.

Discussion on similarities and differences of my work and related works is the main focus of Chapter 5.

I follow with an explanation of the results in Chapter 6.The outputs and the results are explained with figures illustrating the decoys that the system produces.

In Chapter 7, I conclude the report by discussing my achievements through the project. I also briefly describe any expected impact the project might have in future works. I also provide some suggestions about how someone can extend what is done in this project, and briefly touch on what results can later be achieved from possible extensions.

2 Background: Some important terms and concepts

Understanding the aims and approaches of the project required the knowledge of some important concepts and terms. These term and ideas are briefly discussed below.

2.1 Protein Structures

Proteins are compounds that are composed of chains of a combination of 20 different amino acid residues. The peptide bonds between the residues hold them in the chains. The sequence of the residues in the protein is determined by the genetic code in the genes which have given rise to their formation. The amino acid sequence of the protein is termed as its primary structure.

The protein chains naturally fold into characteristic secondary and tertiary structures soon after they are formed by ribosomes in the cells. This natural folded structure is called the protein's native structure. Secondary structures are regular sub-structures alpha helix (cylinders), and beta sheets or strands (ribbons) which can be found at many different locations in the whole protein structure. In a simplified model structure of protein, it can be thought of as a series of these cylinders and ribbons connected together in a defined way (Phillips, et al., 2009). Figure 2.1 shows cartoon representation of a protein with of alpha helices and beta sheets.

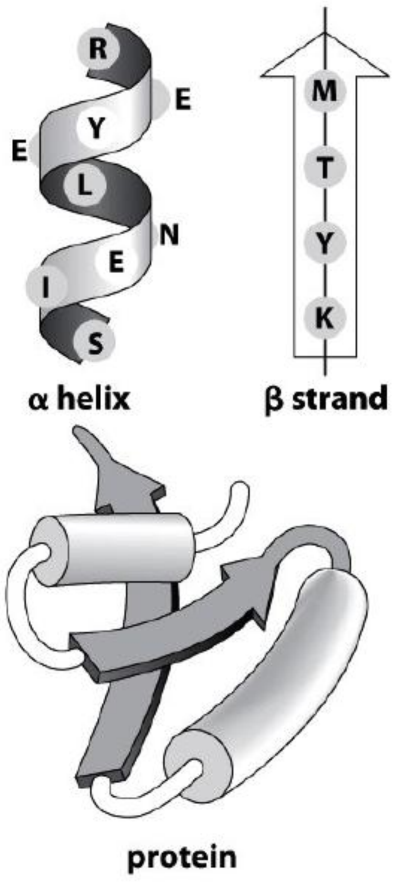


Figure 2.1 Cartoon representations of secondary structures of protein, with alpha helix and beta sheets.
(from Phillips, et al., 2009,
© Garland Science)

The protein chain folds readily in nature and gives rise to a completely folded globular and compact structure termed as its tertiary structure (Phillips, et al., 2009). These globular structures, which any protein typically has, form automatically in nature by ensuring minimization of free energy in the chain. The globular structure in the protein is very important for it to remain stable in the most comfortable position, and is also very important for its function and activities.

2.2 *Ab initio* protein structure prediction and decoy building

In *ab initio* protein structure prediction, the native structure of a protein is predicted once its amino acid sequence is given (Bonneau and Baker, 2001). *Ab initio* protein structure prediction involves two step (1) predicting some candidate structures also known as decoys and (2) formulating an energy potential or scoring function that helps to evaluate the candidate structures' closeness to the native or near native structures (Kolodny and Levitt, 2002).

A decoy is a model chain of alpha carbons i.e. the central carbons of the amino acids of a particular sequence. In other words, decoys are candidate structure of the alpha carbon backbone of a protein structure. Decoys can thus be a close representation of the native structure of the protein or its fragments but is not necessarily the actual native structure of the protein (Hamerlyck, et al., 2006).

It is possible to predict numerous structures from a given sequence, but only one which has the minimum free energy is possible in the nature, and that is the native one for the protein of that particular structure. In order to find this native structure computationally, an exhaustive search among all the possible structures may be necessary, which can sometimes be quite time consuming to carry out. In order to avoid considering of all the numerous possible structure, a relatively smaller number of candidate structures, or decoys are considered in computational protein prediction methods, just to make the work a little easier (Keasar and Levitt, 2003). Building decoys can be done by exhaustively searching all the possible structures and by randomly picking up some from there. Another way is by randomly choosing from only the ones that qualify on the basis of some scoring function.

2.3 The usage of Protein Data Bank (PDB) files

Ab initio protein structure prediction requires the usage of protein sequence information. Most such protein structure prediction methods use the sequence information available in the Protein Data Bank (PDB) (Berman, et al., 2003). PDB is an electronic archival data bank for structures of macromolecules like protein. The sequence and structural information of these macromolecules, which are derived from crystallographic studies are stored in the data bank in uniform format (Bernstein, et al., 1977) known as the PDB format. The entries of this data bank are freely available via the PDB website.

2.4 Dynamic programming and its usage in protein structure prediction

A dynamic programming algorithm breaks down a complex problem into smaller sub-problems, the solutions to which are found first. The solutions to the smaller problems are then joined together to get to the solution of the larger problems. Dynamic programming was first formalized by mathematician Richard Bellman, in the 1950's (Eddy, 2004). Usually dynamic programming approaches or algorithm consists of these steps—(1) Recursively define a complex problem, i.e. define it in terms of smaller sub problems. (2) From a matrix for storing the solutions of the smaller sub problems, (3) a bottom-up approach of filling out the matrix with solutions of the smaller sub problems and (4) solving the complex problem and linking up the smaller solutions which lead to this final solution (Eddy, 2004). Computational biology heavily uses systems built on dynamic programming algorithms. Many programs used for sequence alignment, gene finding and folding uses dynamic programming approaches (Eddy, 2004). The zipping and assembly approach for *ab initio* protein structure prediction software is also based on dynamic programming (Dill, et al., 2007; Hockenmaier, et al., 2006)

2.5 Lattice, random walk and HP models usage in protein structure representation

There are many useful models that are used to represent complex protein folding in a simpler way. One such model is the lattice model where the amino acids of a protein are only allowed to occupy regularly arranged slots in the three dimensional space (Phillips, et al., 2009). Lattice models are combined with the random-walk model representation of macromolecules, where a macromolecule is imagined as being composed of cylindrical segments of equal length arranged randomly in a pattern in the three dimensional space such that segments do not overlap each other. But keeping the fact in mind that proteins take up a compact structure in its native form, the lattice model is incorporated with the random walk model to ensure compactness (Phillips, et al., 2009). This model thus says that, these amino acids composing a protein chain are the random walkers which are arranged in the different slots of the lattice space. And a compact random walking, which ensures that all the slots of the lattice are occupied, ensures the compactness of the structure (Phillips, et al., 2009). Figure 2.2 shows a compact lattice model of protein.

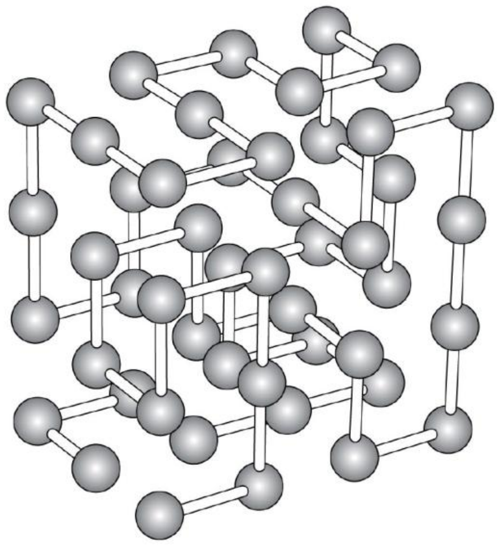


Figure 2.2 Compact lattice model for representing protein structure. The balls represent the amino acids (from Phillips, et al., 2009, © Garland Science).

Another model which is very popularly used for the representation of protein structure is the HP model. When a protein folds to its native form, the hydrophobic or non-polar domains hide themselves in the inside parts, and the hydrophilic or polar domains occupy the parts that are exposed. As this force of hydrophobicity plays an important role in the folding of protein, this HP model is developed where each of the 20 naturally occurring amino acids are categorized as either being hydrophobic (H) or polar (P) (Phillips, et al., 2009). HP model, incorporated with lattice model, is often used in the *ab initio* protein prediction to help explain the relationship between amino acid sequence space and the three-dimensional structure space. The two-letter representation of amino-acids makes the problem predicting the structure of a protein chain much easier to solve (Phillips, et al., 2009). Compact random walking in lattice space makes it much easier to keep track of the possible structures. If two hydrophobic amino acids in the lattice space come close together they form non-covalent bonds that help in folding and minimize the free energy (Hockenmaier, et al., 2006). Thus the energy level of the different configurations of the chains that can be formed in the lattice space can be calculated and scored to be used in protein structure prediction algorithms.

2.6 Similarities to Natural Language Processing (NLP)

Using computational methods to understand the meaning of a sequence of natural language words is known as natural language processing or computational linguistics (Bates, 1995). Natural languages are composed of sequence or strings of words, and similarly protein chains are composed of strings of protein monomers or residues. In natural language processing, the meaning of a given sequence of words is being predicted, and in *ab initio* protein structure prediction, the structure of a given sequence of protein residues is being predicted. These problems aim towards predicting a structure (semantic structure in the former case, and physical structure in the latter case) based on a given sequence. Because of these similar goals, variants of methods and algorithms used in natural language processing can often be used to solve the problem of protein structure prediction (Dill, et al., 2007).

3 Related Works

There had been a lot of work on *ab initio* prediction of protein structures. Besides other approaches, many use dynamic programming approaches and algorithms that are used in natural language processing field. There have been many different ways of representing the structures also. Some works which were studied closely to get ideas about these different approaches are discussed below.

3.1 Usage of dynamic programming with natural language processing algorithms

Computational linguistics is the field where computational tools are used in parsing and understanding natural languages. As a biopolymer chain or protein chain is just a string of monomers or protein residues, it resembles natural language sentences, which are also strings, but of words. Depending on a grammar rule, every sentence of a natural language has its meaning encoded in it. And similarly, the 3-dimensional structure of a protein is also encoded in its one dimensional sequence information, depending on what gives the minimum global free energy.

3.1.1 Parsing and the CKY algorithm

When the meaning of a natural language sentence is to be determined, it has to be parsed. Parsing is the computational job of finding the correct syntactical structure of a string of words in a natural language sentence according to a grammar that defines the all possible syntactic structures. Parse trees or phrase structure trees generated using the given context-free grammars represent the different possible parsing structures of the given natural language sentence. Figure 3.1 shows the phrase structure trees showing the syntax of the two sentences 'I eat sushi with tuna' and 'I eat sushi with chopsticks' (Hockenmaier, et al., 2006).

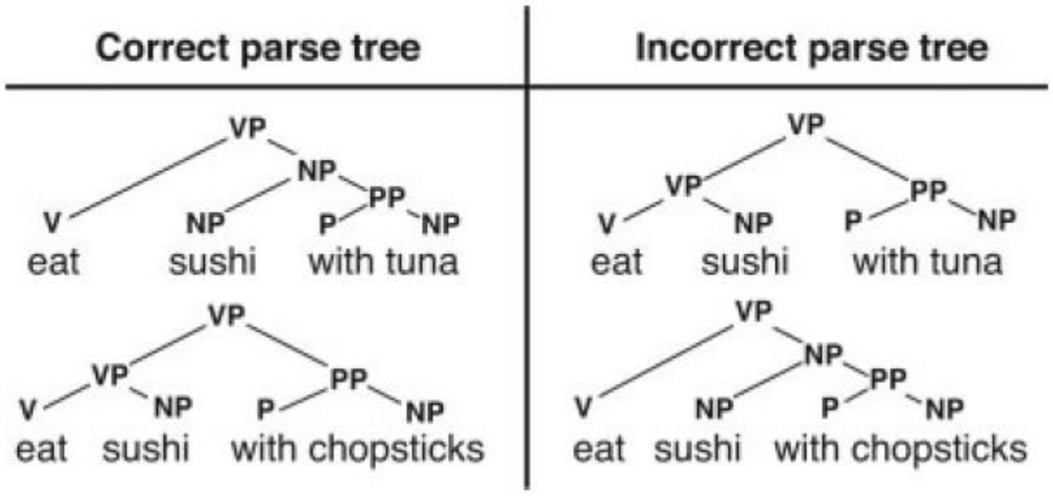


Figure 3.1 Parse trees showing grammatical structures of sentences (from Hockenmaier, et al., 2006).

Hockenmaier, et al. (2006) have also said that just the way trees can be used to represent grammatical structures of natural language sentences, they can be used to represent protein folding also, because protein folding is also hierarchical and folding routes have a tree-like structure. In the mentioned paper, the authors described in detail how Cocke Kasami Younger (CKY) algorithm, a chart parsing dynamic programming algorithm that is used in parsing natural language sentences, can also be used to give insight about the folding process in protein leading to the prediction process of a protein structure.

CKY maintains a chart known as a parse chart, which stores information about all the different grammatical parsing (syntactic structure) of a given sentence. The parse chart is a table, each cell of which (chart [i][j]), represents the part of the sentence starting from word i to word j. CKY fills out the table in the bottom-up form by first filling out the cells in the main diagonal with individual words, and then moves to the next diagonal level, and fills out the cells chart[i][i+1], and then the next level, i.e. cells chart[i][i+2], so on and so forth until it reaches the corner most cell, or the topmost cell at chart[1][n].

3.1.2 ZAMDP variant of the CKY algorithm

Dill, et al. (2007) also discusses how methods normally used in computational linguistics can help in assembling protein-like structures. They explained how they used dynamic programming algorithms to build protein models of native structures. They have devised an efficient algorithm for predicting the structures of a protein chain, and an algorithm for computing the partition functions and stabilities of helix bundle protein. In my project, I extensively took ideas from how the first algorithm discussed in this paper works. I have taken similar approach in building the data structures, and assembling different fragments.

As discussed in the paper, Dill, et al. (2007) have said that prediction of a structure of protein involves considering all the possible topologies (conformation) of a possible native structure from a given string information, and choosing the one that has the lowest free energy. They have used dynamic programming methods, i.e. they have divided the problem into smaller solvable problems, and have used and combined the solutions of these small problems to get to those of larger problems. In other words, they have broken down the whole protein sequence in smaller fragments, and have got the lowest energy structures for those fragments, and have built larger structures assembling these smaller ones (Dill, et al., 2007).

Dill, et al. (2007), have actually devised a variant of the CKY (Cocke Kasami Younger) algorithm. Their variant is named ZAMDP (Zipping and Assembly Mechanism by Dynamic Programming). This algorithm uses recursion and dynamic programming to build protein decoys (native like protein structure).

3.1.3 Incorporation of the HP model in protein structure prediction algorithms

In the ZAMDP variant of the CKY algorithm, Dill, et al. (2007) used the simple HP model, to represent the context-free-grammar-like rules for protein chain. Hockenmaier, et al. (2006) is where the idea of adapting the CKY algorithm to the HP model was introduced before Dill, et al. (2007) built this variant. In the HP model, a protein chain is represented by a short sequence of two different kinds of monomers. A monomer unit is either hydrophobic (H), or polar (P).These monomers are placed in two or three dimensional lattice space. A monomer or amino acid is represented by a bead in the lattice space. If two hydrophobic monomers come close, they form non-covalent bonds that help in folding and minimize the free energy. So the energy functions, which indicate how much energy minimization has occurred, are based on the contacts between two adjacent hydrophobic monomers (Hockenmaier, et al. 2006; Dill, et al., 2007).

The free energy contained in the protein depends on how many HH bonds are there, and each such bond contributes to a -1 in the energy function. ZAMDP uses the chart parsing method with the adaptation of the HP model of protein representation, and predicts the structure of a protein. The structure of the protein is predicted by first finding structures for smaller fragments, storing them in a look-up table like the parse chart in CKY algorithm, and then assembling adjacent pairs of these fragment structures. The chart or the lookup table in the ZAMDP contains the structures with monomers represented by either an H or a P (Dill, et al., 2007).

3.2 Usage of other dynamic programming and divide and conquer approaches

Hamelryck, et al. (2006) also discusses how dynamic programming or divide and conquer method, is used in predicting protein structures. In such a method again, the larger problem of determining the structure of a long protein chain, is broken into smaller solvable problem of generating much smaller fragments. Finally structures of these smaller fragments are again combined to generate the full structure. In their paper they give the name decoy to the protein-like structures that they are generating. The decoys that are assembled are either accepted or rejected based on some energy functions.

They focus on generating decoys based on local sequence or structure preference (local structural bias, as termed by them). That is, they use fragments of sequences with secondary structures from the fragment library to build the whole structure. First fragments from the libraries, that match with parts of the given sequences are picked and they are combined with computationally built structures from the other parts of the sequences to give the final structures. The local structures of a sequence are picked using complex probabilistic methods.

Hamelryck, et al. (2006) used a complex stochastic sampling method to construct an alpha-carbon backbone iteratively by removing steric clashes and re-sampling. They generated compact decoys by first initializing with candidate fragment structures, then

removing steric clashes and then collapsing the structures. They used 4Å to be the critical distance for steric clash. They accepted a structure only after the steric clashes are minimized, or totally removed. The lengths of the segments are chosen randomly. Then the structure is collapsed iteratively. Also, the structure is only accepted if the *radius of gyration* of the structure is only lower or equal than the previous structure. Radius of gyration, R_G is a measure that gives an idea about the size of the protein and also about its compactness. Lobanov, et al., (2008) describes this quantity as a parameter that gives the equilibrium conformation of the structure. They suggest the computation of this value for protein using the formula:
$R_G{}^2 \sim \Sigma(r_i - R_C)^2/N$, where r_i is the coordinate of the i^th^ residue, R_C is the coordinates of the center of mass and N is the number of atoms other than hydrogens in a protein. A small radius of gyration would mean a compact protein. In Hamelryck, et al., (2006), decoys are not accepted if there is a steric clash, and collapsing is stopped once the radius of gyration falls below a certain threshold or a maximum number of iterations take place.

3.3 Other methods for building decoys

(Kolodny and Levitt, 2002) uses assembling of small fragment taken from a fragment library to build larger decoys. The fragment library is composed of 20 fragments each having the alpha carbons of five residues. These 20 fragments are used as the building blocks of the decoys, as they are picked from the library and added to each other to lengthen the decoy chain. Each added fragment is placed on another by overlapping the first three residues of the fragment on the last three residues of the growing chain. The orientation of a fragment is determined by the first three residues composing it. Self-avoidance is enforced by making sure that any two alpha carbons are separated by at least 2.5 Å. They enforce compactness by requiring the decoys not to exceed a range of sizes. Figure 3.2 shows how Kolodny and Levitt (2002) uses fragments from a library and overlaps them with each other to build decoys. While fragments with four residues are shown in this figure, fragments of five residues were used in the method implemented by Kolodny and Levitt.

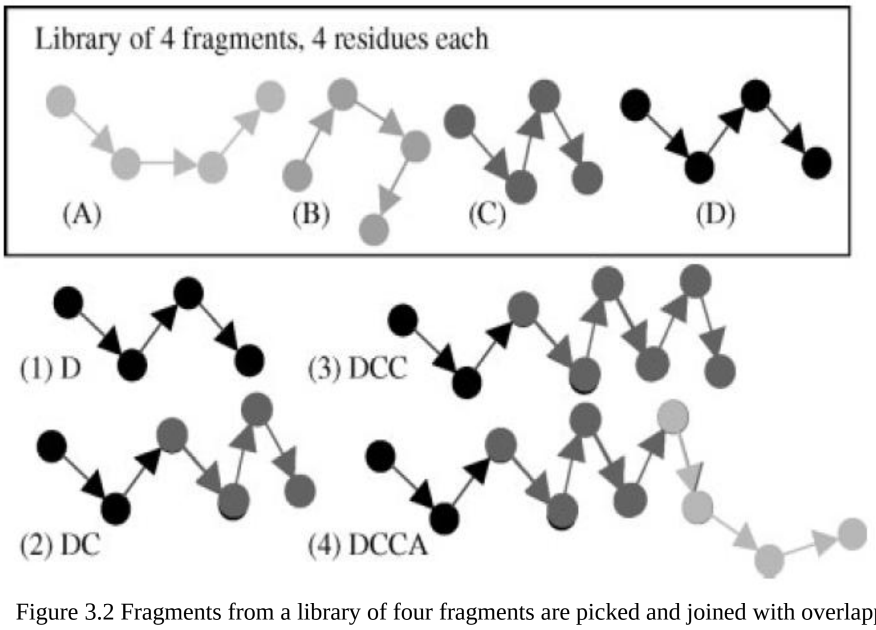


Figure 3.2 Fragments from a library of four fragments are picked and joined with overlapping of residues to build decoys (from Kolodny, Levitt, 2002).

3.4 Representation of the decoy structures

In their method, Hamelryck, et al. (2006) consider only the alpha-carbon backbone to represent the protein chain. The alpha-carbon backbone is a string the alpha-carbon atoms of the amino-acids. Each of the alpha-carbons is on average 3.8 Å away from the next one. As shown in Figure 3.3, a sequence of pseudo angles ‘Theta’, and dihedral angles termed as ‘Tao’ in the paper are used to describe conformation. Dihedral or torsion angle is the angle between two planes that contains two sets of points. For example, in a three dimensional chain of atoms A-B-C-D, the angle between the plane where the atoms A,B and C lie, and the plane where B,C and D lie is their torsion or dihedral angle. Many other papers have denoted dihedral angles of protein chains by ‘Tao’ or by ‘Alpha’. In this project, I have referred to the pseudo angles by theta, and the dihedral angles by alpha. These pseudo angles usually in nature range between 80° and 150° and dihedral angles range between -180° and 180°.

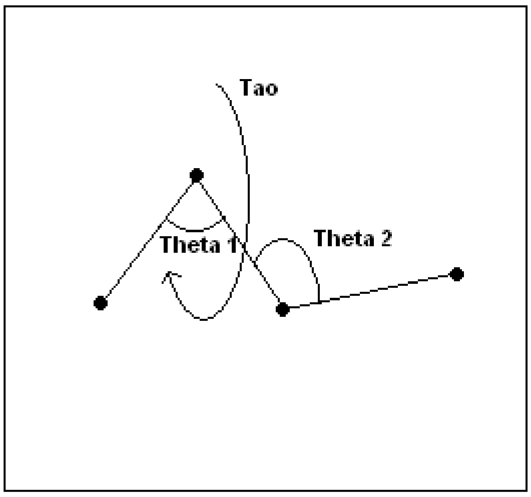


Figure 3.3 Theta 1 and Theta 2 represent the pseudo angles, and Tao represents the dihedral angle in an alpha carbon backbone.

4 A system that assembles protein-like decoys

Just like others have tried to design methods and systems for building protein structure, I also tried to build a system that potentially can form protein-like structures or decoys. The main aim of the project was to build a program that can assemble small fragments with real protein geometries to build three dimensional protein-like structures (decoys) using a convenient data structure.

4.1 Purpose of the system and the overview of the algorithm

The purpose of the system was to take the structural information of a protein in PDB format as its input, use the geometries in that structure, and generate a number of three dimensional structures of the same length. Below is a brief description of what the system does.

The system is a C program that can read in a PDB file, which contains the information of only the alpha-carbons of the different residues of a protein with known structure. Once it reads in the sequence information, it saves up the information about the geometries of the protein, i.e. the angles among the residues. Then it uses this information to form small fragments containing only three residues. It makes up several three-residue-long fragments, i.e. for example makes five fragments of the first, second and third residues, five of the second, third and the fourth residues etc. The number of fragments in a cell which is now set to be five, can be easily changed. Each of these fragments have different structures and positions in three dimensional space, reflected by the coordinates of the different residues, and the angles subtended by the three residues. The angles subtended by the three residues are assigned after they are randomly picked from a list of typical angles found in the given protein whose structure is known. All these five suggested structures for each of three consecutive residues are then saved in a cell of the third diagonal of a one dimensional array that can be visualized as a 2 dimensional triangular matrix as shown in Figure 4.1.

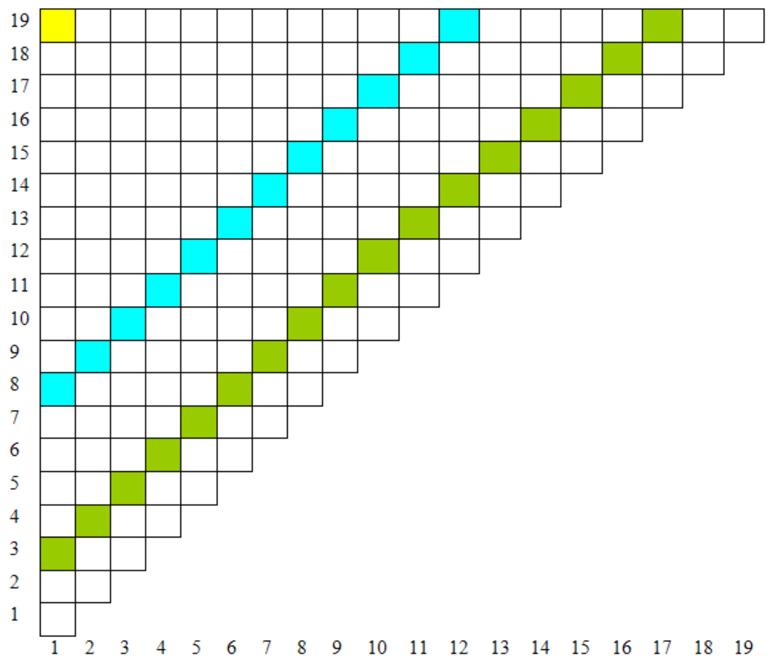


Figure 4.1 This is how the triangular matrix would look like if it were to carry structures of 19 residues. The green cells represent the 3rd diagonal where all the structures of length three are stored. The cells in blue contain structures of length eight as that is the 8th diagonal. The Nth diagonal (19th in this case) which has only one cell, and is the topmost corner has information of structures of length N.

The coordinate information, along with some more information of groups of five three-residue-long fragments are saved in each of the cells of the third diagonal of the triangle as shown in Figure 4.1. As each of the fragments are of length three, the third diagonal is used. Fragments of length n, are to occupy the nth diagonal of the triangle. If I were to make fragments of length one residue and fragments of length two residues, they would have occupied the 1st and the 2nd diagonals of the triangular array respectively. But this is not necessary for this project. As mentioned earlier, in each of the cells of the triangular matrix, information of five fragments is saved. This number can easily be altered in the program, but five used for now as this number was enough to fulfill the purpose. The systems task is then to build longer fragments of size four residues by picking up three-residue-long fragments randomly, and filling up the fourth diagonal layer. Every time a fragment is assembled, it is checked that no two residues in that are too close together, that is there is no steric clash in such a fragment. If they are too close(less than 3Å) then that fragment structure is not saved anymore, rather another structure is formed again by randomly picking up fragments from the lower diagonal layer(s).

Similarly the fifth diagonal layer is filled with five-residue-long fragments built by randomly picking up the fragments of size four residues, and assembling them to those of size three residues and vice versa. Building the fragments here actually means, combining the coordinate information of each of the two individual fragments in such a way so that it formed chains by overlapping just the last two residues of the first fragment with the first two residues of the second fragment. Once this 5th diagonal layer is filled, the system iteratively fills up the 6th, then the 7th and so on and so forth. It knows what lengths of the fragments should be used to result in a longer fragment of a certain length and also knows where it can find them. (nth diagonal has n residue long fragments.) So it searches in the appropriate diagonal layer of the triangular matrix, and randomly picks up appropriate fragments to build the required fragment. Iteratively all the layers are filled until the top-most corner cell of the triangular matrix (the cell colored yellow in figure 4.1) is reached were resides five complete chains of the given sequence. Every time a cell is filled with any fragment information, it is checked to avoid steric clash. Finally the system ends up in storing five chains of the entire length in the topmost cell of the triangular matrix.

The five final chains in the topmost corner cell of the triangular matrix are the five final solutions given by the system. The information of each of the solution structures are printed out on screen in the PDB format. This information then can be used to view the structure in a visual tool like RASMOL.

4.2 The main tasks of the program and the data structures

The entire task carried out by a number of C functions. The main task can be divided into two main parts:

- Using the geometries of naturally occurring proteins
- Populating the triangular structure with information of fragments which can be subdivided in the two following parts
  - Populating the third diagonal
  - Populating the rest of the diagonals

These different parts of the task and the data structures involved are described below with the help of flow-charts or pseudo-code and illustrations of the data structures.

4.2.1 Using the geometries of naturally occurring proteins

The first part of the task was to save geometries of naturally occurring protein. The task is illustrated in a flowchart given in Figure 4.2. In this part of the program, data from a given PDB file was read to calculate and save different angles that can be subtended by sets of three consecutive alpha carbons of a protein chain, and the different dihedral angles that are found naturally in between planes of alpha carbons. These values were stored so that they can be used later on to be assigned to the chains to be generated.

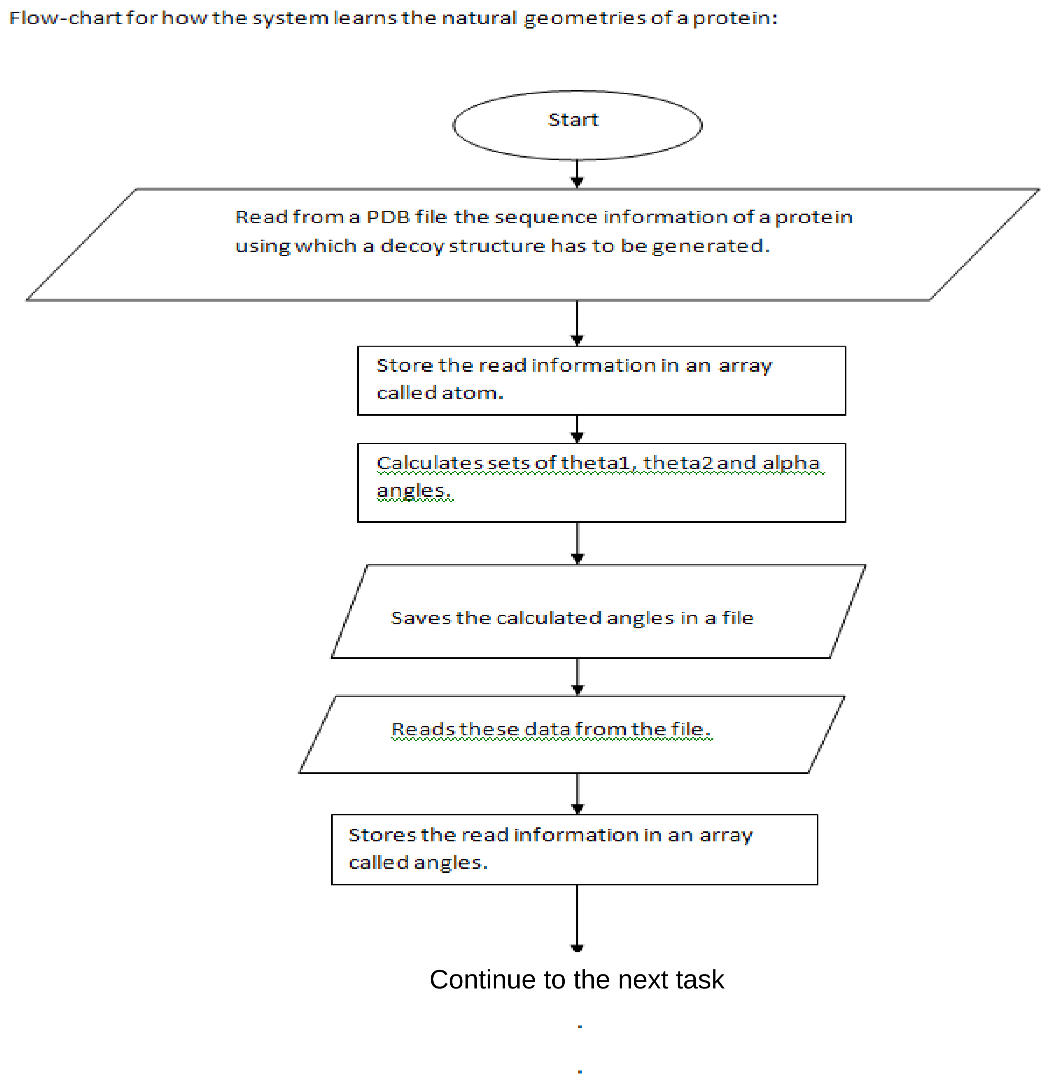

Flow-chart for how the system learns the natural geometries of a protein:
Start
Read from a PDB file the sequence information of a protein using which a decoy structure has to be generated.
Store the read information in an array called atom.
Calculates sets of theta1, theta2 and alpha angles.
Saves the calculated angles in a file
Reads these data from the file.
Stores the read information in an array called angles.
Continue to the next task

Figure 4.2 The flowchart of the part where the system learns the values the angles naturally formed by residues of protein chains. This angle values will be randomly assigned to the decoy structures to be generated.

Reading information from a PDB file, storing them in appropriate data structures, and calculating and storing the angles subtended by the alpha-carbon of the chain were done using C functions and codes that were written by incorporating functions written and provided by my supervisor Dr. Graham Kemp.

This part of the program reads in a PDB file with the sequence information of the protein whose information would help in assign angle values to the decoy structures to be produced. While the program does the reading, it stores the read information in an array called **atom**. The array atom has a size of a constant value MAX_ATOM, which is set to a quite big number like 10000 in order to be able to accumulate large number of atoms that can be present in a typical protein chain. Figure 4.3 shows the structure of the array atom and its constituent.

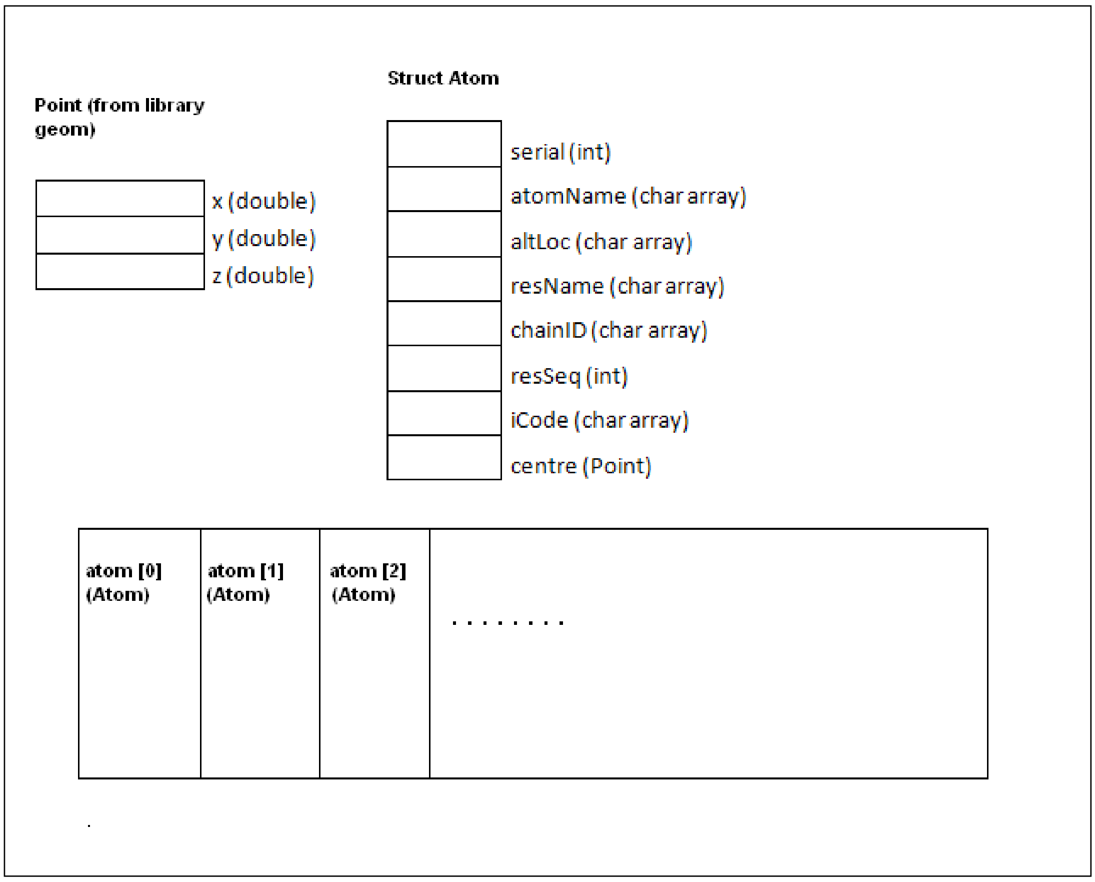


Figure 4.3 The array atom is composed of struct Atom and is used to store data read from a PDB file. The information read from this PDB file is used by the system to save the natural geometries of a protein.

The **angles** array is where sample pairs of theta angles, and alpha angles extracted from already known protein structures are stored. Figure 4.4 shows the structure in detail. This is an array of size MAX_ANGLES which should at least as big as the number of residues in the known protein sequence which is given as the input for collecting the sample angles.

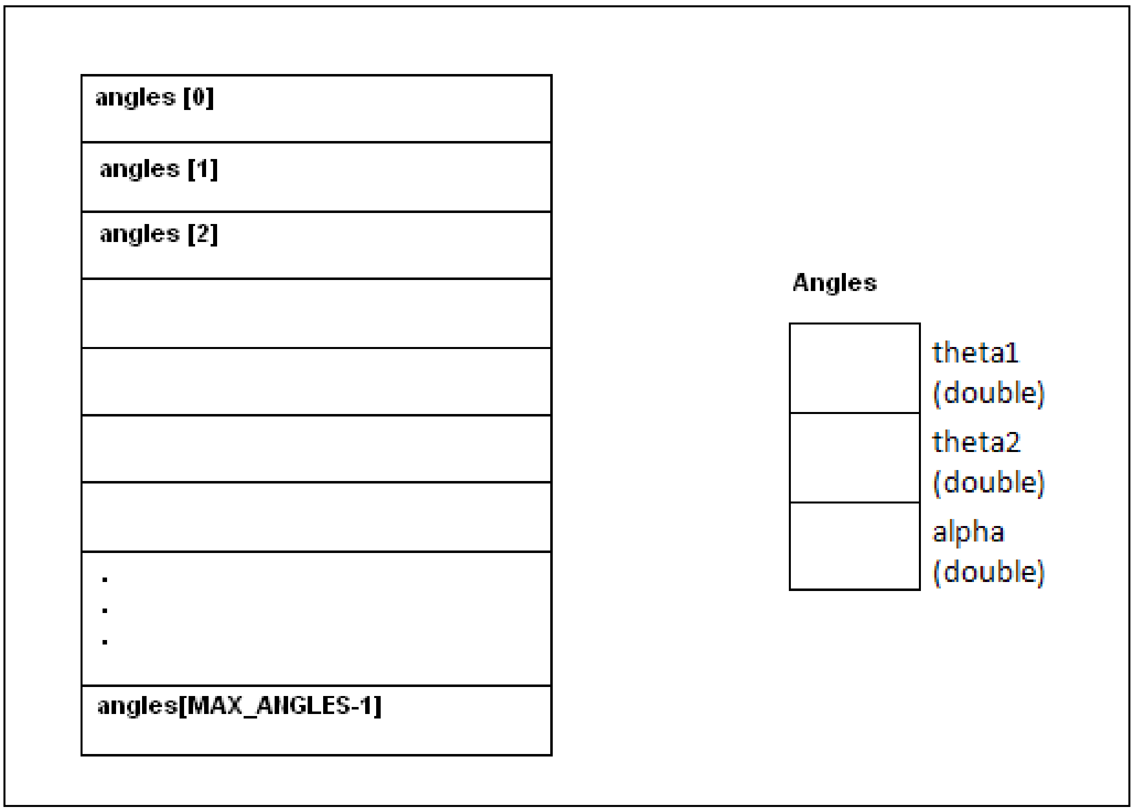


Figure 4.4 The array angles is composed of the structure Angle with the indicated data fields. This array is used to hold the pairs of theta angles and the dihedral angles (alpha) using sequence information of a known protein chain.

This data structure was used to store the groups of angle values of theta angles, and the dihedral angles. The theta angles theta1 and theta2 in the data structure are the pair of angles each formed by set of three consecutive alpha carbons in a series of four alpha carbons as shown in Figure 3.3. The data field alpha in the data structure corresponds to the dihedral angle denoted by 'Tao' in Figure 3.3.

4.2.2 Populating the triangular structure

The next step is to start populating the triangular data structure. The triangular data structure is built of a one dimensional array, that is represented in such a way that the cells of it can be accessed just the way those of a two dimensional array are accessed. This triangular structure referred to as **one_dimensional_array** is composed of more complex data structures like **solution_details, coordinate_info** and **aSolution.**

solution_details is the most important data structure of this program, as it contains all the information of a particular solution decoy fragment produced by the program. It forms the main constituent of our triangular matrix. Each of the cells of the matrix contains a pointer to an array of five (in my program it is fixed to be five but the number can be altered) of solution details. solution_details is composed of information like the number of residues found in the particular fragment represented. The ID or serial number of that particular fragment to distinguish among the different fragments found in a particular cell is also part of the structure. The angle subtended by the last three residues of the first fragment it is composed of and the angle subtended by the first three residues of the second fragment it is composed of are parts of the structure too. The torsion angle formed among the planes containing the residues of the two joining fragments, at the place where they join together are also saved in this data structure. It also contains the location information of the two fragments that build up a particular fragment. That is, it has pointers to a structures which contain information about where each of the two fragments that combine to form a single fragment have come from and the Cartesian coordinates of the alpha carbons of all the residues forming the particular fragment. Figure 4.5 illustrates the structure of this data structure.

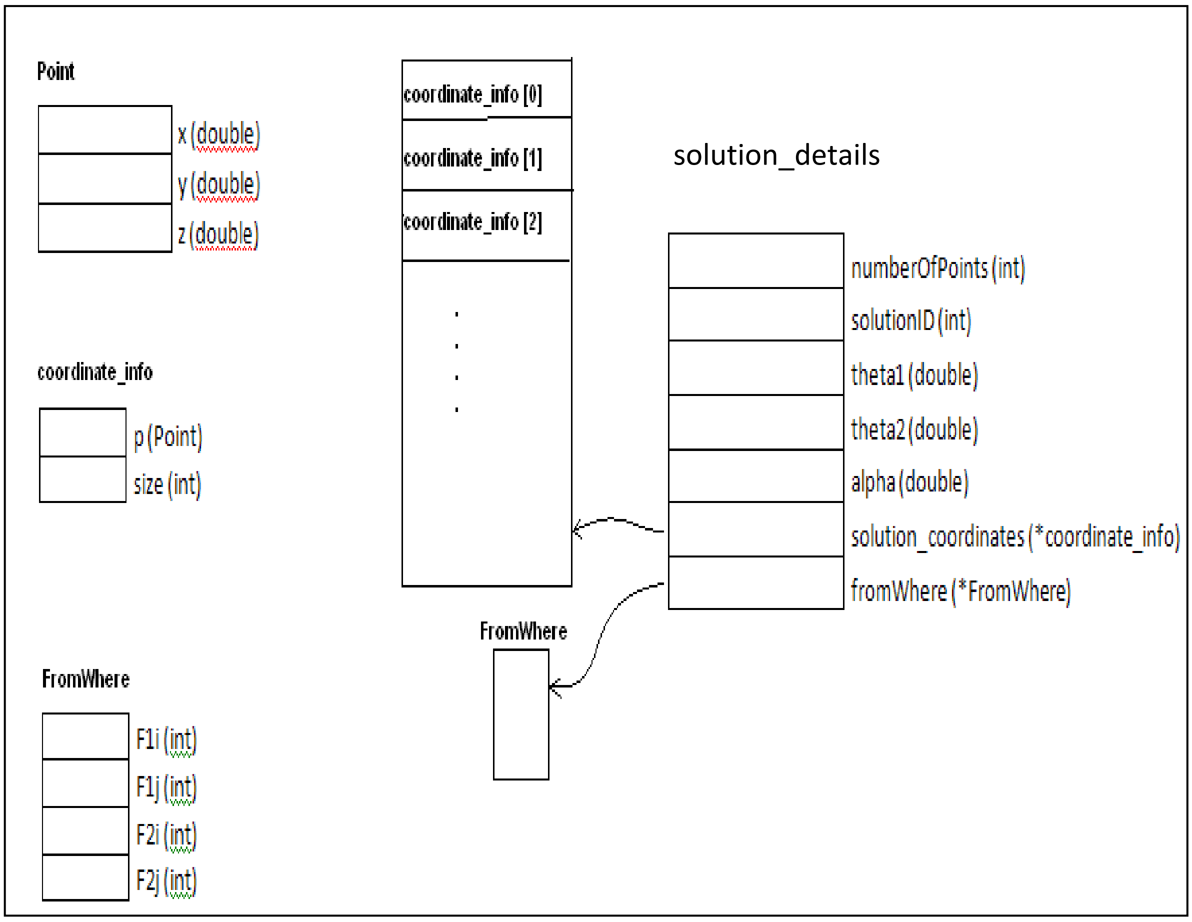


Figure 4.5 solution_details contains information about a fragment formed at any stage. It has a pointer to array of coordinate_info that contains the Cartesian coordinates of a fragment, a pointer to a structure called FromWhere which has information about where each of the constituent fragments have come from, and other information needed to identify a fragment.

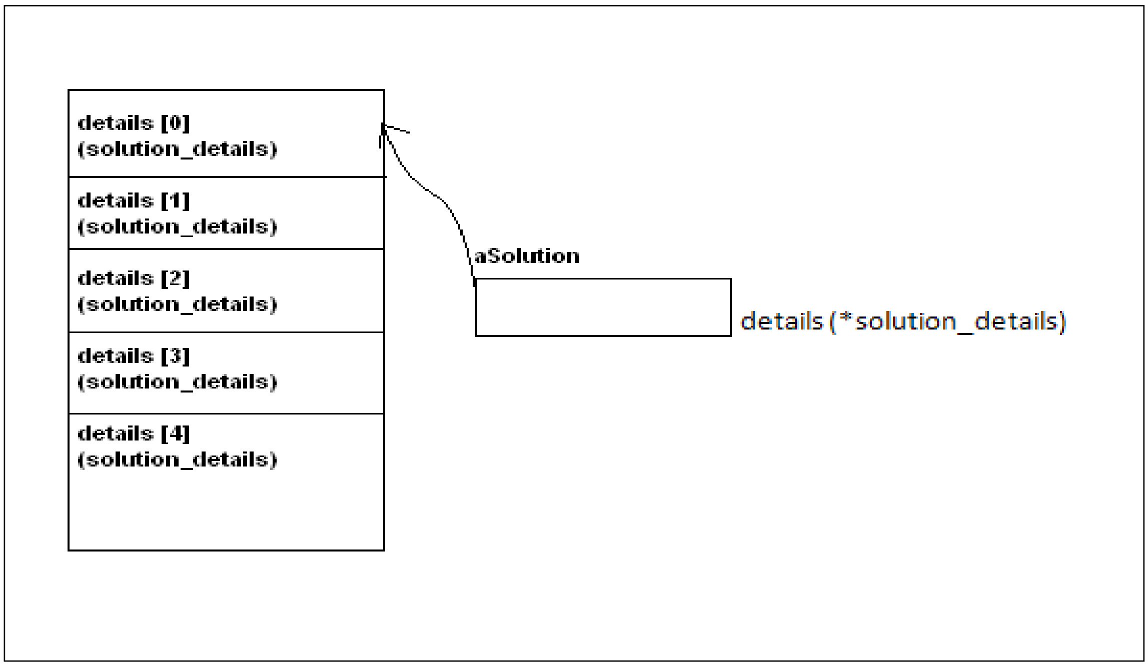


Figure 4.6 aSolution just has a pointer to a five element array of solution_details. Each of the cells in the triangular data structure points to an aSolution as shown in Figure 4.7

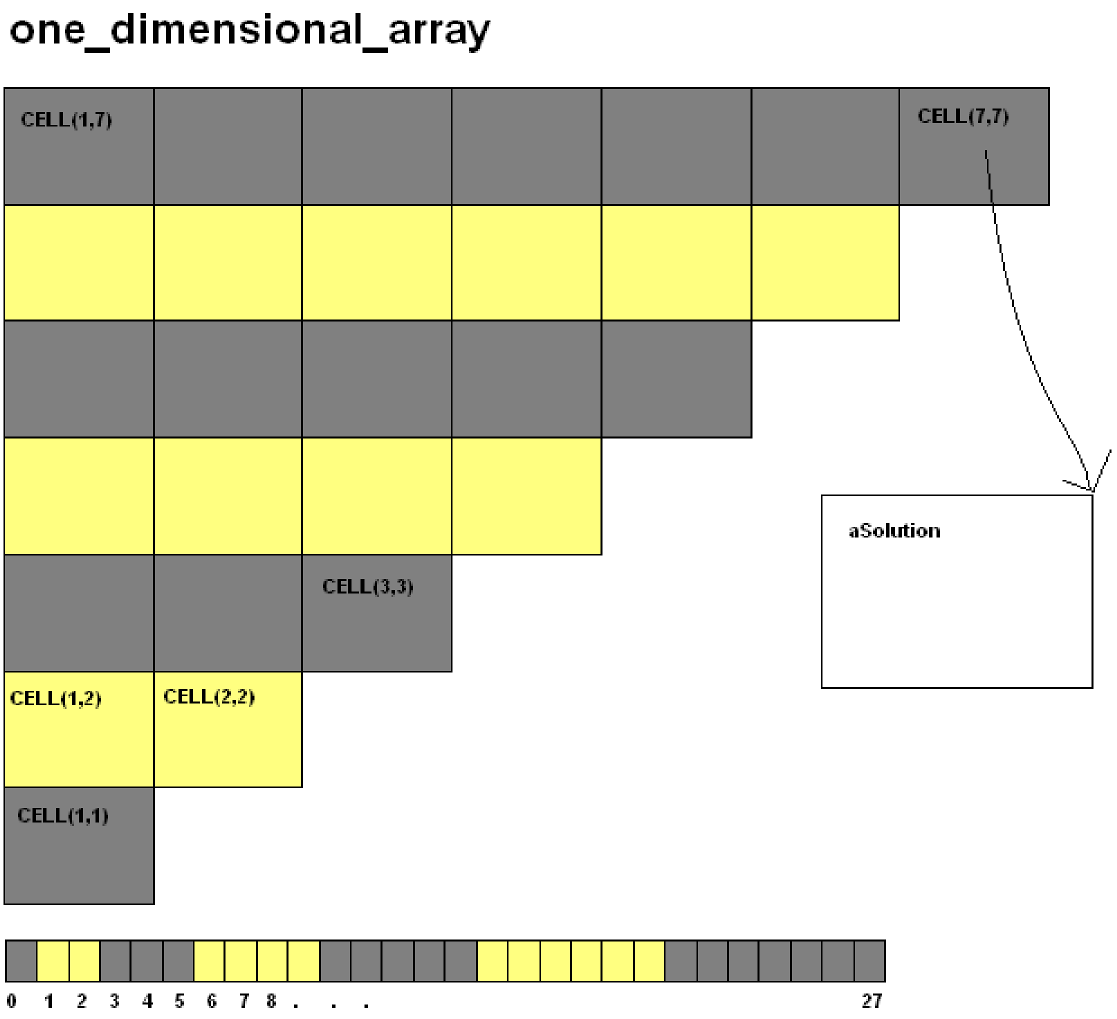


Figure 4.7 How one_dimensional_array can be represented as the triangular structure when N=7. The two-dimensional layout is shown on the top, and the actual one dimensional layout is shown in the bottom. The different colors shows the corresponding cells shown in two different layouts. Each of the cells of the triangular structure points to an instance of aSolution, that is contains details about five fragments.

aSolution contains a pointer to a dynamically allocated array of type solution_details as shown in Figure 4.6. Each of the cells of our triangular matrix, i.e. one_dimensional array points to an instance of aSolution. Each of the cells can be accessed just the way how that of a two-dimensional array is accessed. A cell is accessed through **CELL(i,j)** where i and j are the two indexes, i being the row number and j being the column number of a particular cell. CELL(i,j) accesses the cell one_dimensional_array[((j) * ((j)-1))/2 + ((i)-1)]. Figure 4.7 shows one-dimensional representation of the one_dimensional_array alongside it's two dimensional representation.

4.2.2.1 Populating the third diagonal

As the fragments are built starting from three-residue-long fragments it is necessary to first populate the third diagonal of cells in the triangular matrix by three-residue-long sequences. We do not need chains composed of only one residue, or two residues, so the 1st and the 2nd diagonal layer of cells in the triangular structure will remain empty. First the three-residue-long chains populate the corresponding cells separately as this is a special case. The cells of the one_dimensional array with indexes (1,3), (2,4), (3,5), (4,6) etc are filled with the information of three-residue-long chains. Figure 4.8 illustrates a flowchart of how the 3rd diagonal of the triangular structure is populated.

As illustrated in Figures 4.6 and 4.7, each of the cells in the one_dimensional_array has information of five different solutions. In other words, for a protein sequence of length, N, the third diagonal, that is all the cells with indexes CELL[i][i+2] (where i is a number between 1 and N) starting from CELL[1][3], till CELL[N-2][N], is filled first. Each of these cells has the information of five fragments of length three. CELL[1][3] has info about representation of chain from the residue number 1 to 3. Similarly, CELL [5][7] has information about chain starting from residue number 5 to 7. That is [i][i+2] has information about chain from residue number i to i+2.

As each of the cells in the third diagonal has information of five three-residue-long chains in it, when each of the cell is accessed, dynamic allocation of five solution_details is done to be able to hold five information. As only one angle can be subtended by three residues, only theta1 of the solution_details is assigned with some value. This value of the angle to be assigned is randomly obtained from the *angles* array. The three residues are all set to have their center residue at the origin. The first alpha carbon of the 3 residues is set to have the coordinates (-D,0,0) where D=3.8 (representing 3.8 Å which is the average distance between the alpha carbons in a typical protein chain (Hamelryck, et al., 2006)). The center residue (i.e. the second residue) has the coordinates at the origin, i.e. (0,0,0).The third residue has the x coordinate to be equal to D*radian(cos(180-theta1)), where D=3.8, theta1 is the theta angle (in degrees) subtended by the three residues. The third angle has the y coordinate to be equal to D*radian(sin(180-theta1)). The z coordinate is 0. Assigning the mentioned values ensure that each of these fragments are anchored at the origin of the three dimensional plane.

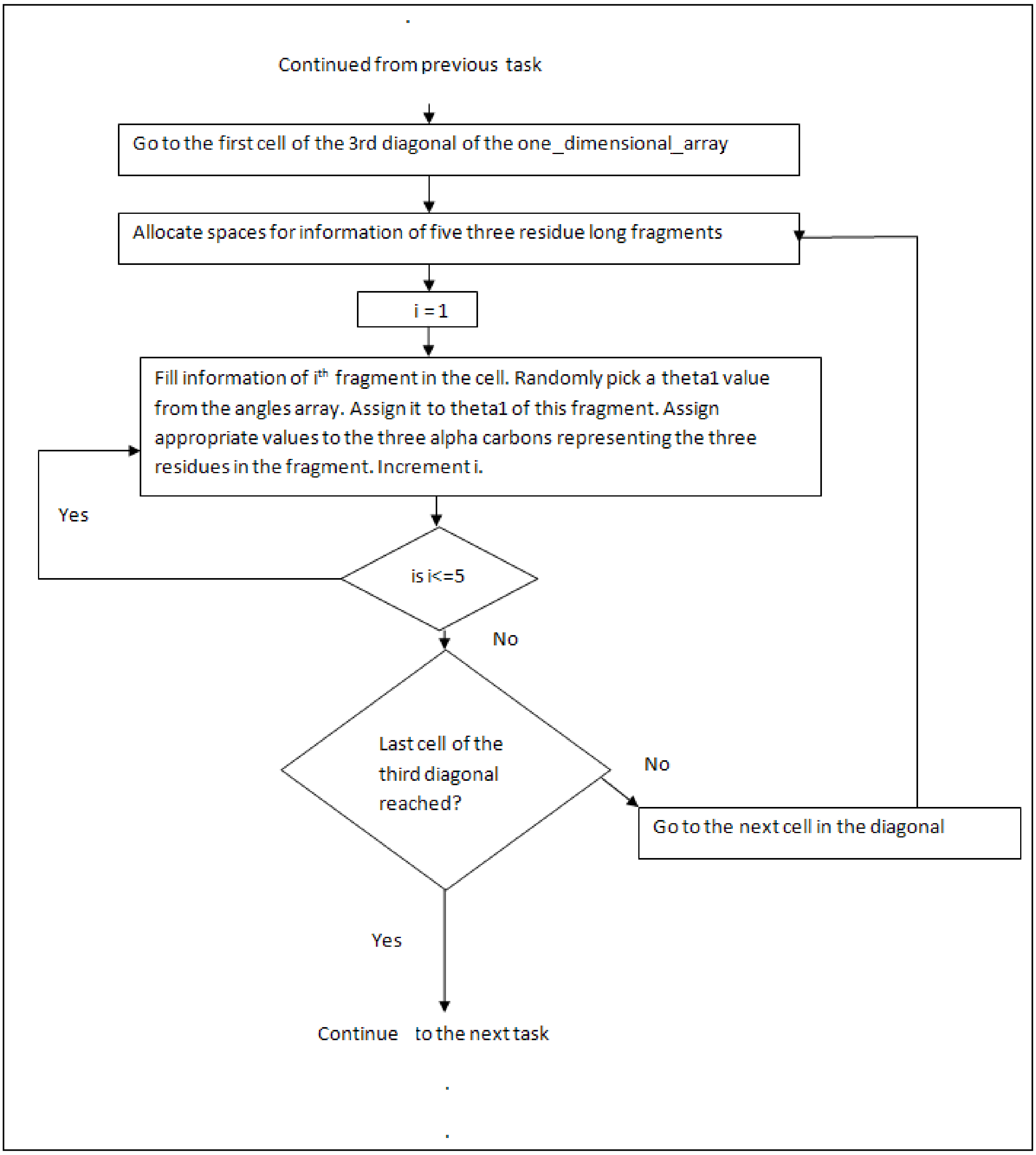

Continued from previous task
Go to the first cell of the 3rd diagonal of the one_dimensional_array
Allocate spaces for information of five three residue long fragments
i = 1
Fill information of ith fragment in the cell. Randomly pick a theta1 value from the angles array. Assign it to theta1 of this fragment. Assign appropriate values to the three alpha carbons representing the three residues in the fragment. Increment i.
Yes
is i<=5
No
Last cell of the third diagonal reached?
No
Go to the next cell in the diagonal
Yes
Continue to the next task

Figure 4.8 A flowchart for how the third diagonal of the triangular structure is filled.

4.2.2.2 Populating the higher diagonals

Then the program iteratively uses the three-residue-long chains to build four-residue-long-chains. Using loops, the program accesses the 4th diagonal of the triangular matrix and populates each of the cells in the layer with information of five four-residue-long chains, only keeping the ones that do not have residues placed too close together (at a distance less than 3Å). Thus CELL [1][4] has information required to represent chain from residue number 1 to 4, and CELL [3][6] has chains from residue 3 to 6. That is all CELL [i][i+2+1] (where i is a number between 1 and N) till CELL[N-2-1][N], are next filled. It then goes to the 5th layer, and similarly populates each of the cells with information of five five-residue-long chains. It iteratively does the same for the 6th layer till the Nth layer, where N is the number of residues in the given sequence. The Nth layer only has only one cell, that is the topmost corner cell of the triangular matrix. Figure 4.9 shows the high-level pseudo-code of how the cells starting from the 4th diagonal till the top-most corner cells are filled. Figure 4.10 shows in which order the different diagonals filled.

```
Set numberOfPoints to be equal to 4
While (numberOfPoints<=N)
    Set x=1
    While(i<= N-2-(numberOfPoint-3) or j<=N){
            Go to the CELL[i][j], where, i= x, and j= x+2+(numberOfPoints-3)
            Allocate 5 spaces for storing solutions
            Set numberOfSolutions=0
            While (numberOfSolutions<5){
          ┌──────►  Pick a random number to choose the two cells from
          │         where the two fragments of appropriates length will be
          │         picked
          │         Pick random numbers to  choose which of the
          │         five fragments from a particular cell to be picked
          │         Find the best dihedral angle that can form in between
          │         the two combing fragments and make adjustments to
          │         what is formed
          │         Combine the two fragments
          │         If there a steric overlap in the fragment
          └──────────────────── Go to
                    Increment numberOfSolutions
            }
            Increment x
        }
        Increment numberOfPoints
}
```

Figure 4.9 Pseudo-code of how the cells starting from the fourthdiagonal uptill the topmost corner cell are populated.

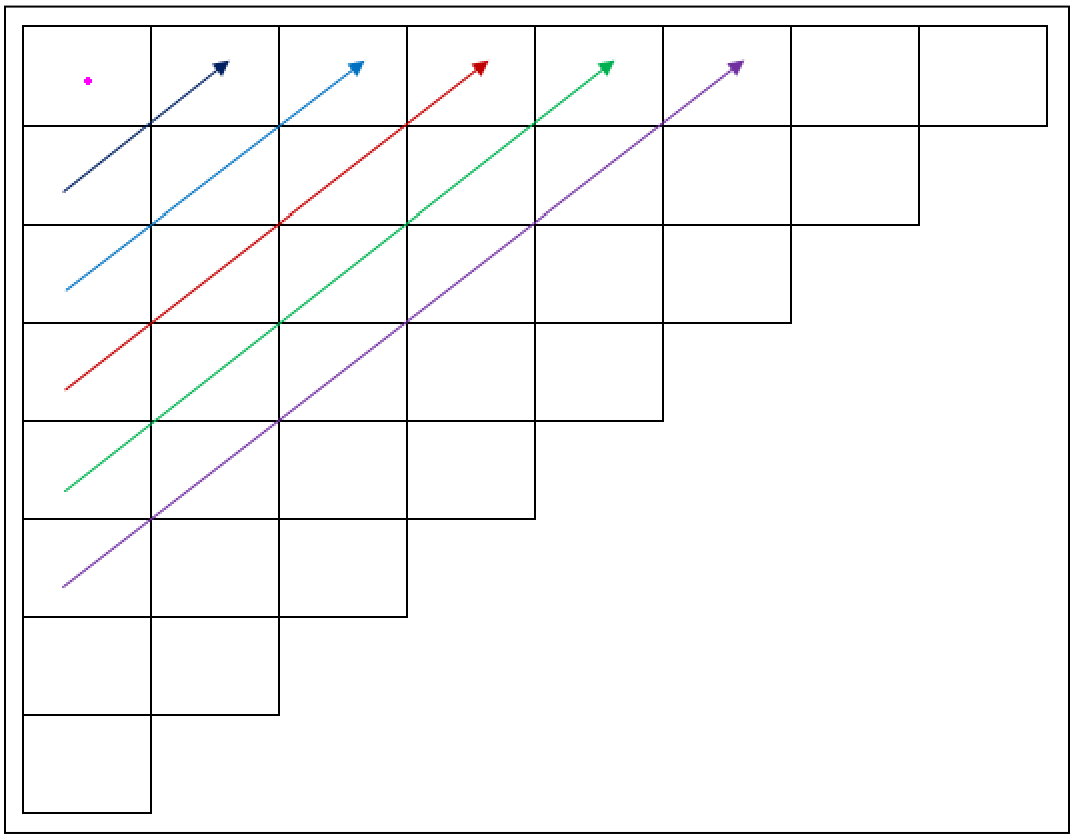


Figure 4.10 The third diagonal is filled first. Then the fourth diagonal gets filled up, and so on and so forth. Each new diagonal is accessed once the filling of the lower one is completed. Finally the last diagonal at the topmost corner which has only one cell is reached and filled.

How the chains are combined

To combine two fragments first the lengths of the two combining fragments have to be determined. Then it is necessary to determines which cells of the triangular matrix, and which slots of these cells should supply these fragments. Then necessary transformations in the 3 dimensional space should be carried out to on the fragments physically put them together to join them in one resulting fragment. These steps are described in detail below.

Determine the lengths of each of the fragments

Each of the chains can be built by combining two fragments. The aim is to overlap the last two residues of the first fragment with the first two residues of the second fragment. The length of the fragments will depend on the fact that two residues of each of the fragment will overlap on each other when they combine together. So if the required length of the resulting chain is z, the length of the first fragment is x, and the length of the second fragment is y, then it should be the case that x+y-2=z. Therefore, for example, a chain of length 8 (a chain at a cell of the 8th diagonal) can be formed from fragments of sizes 3 (a fragment from 3rd diagonal) and 7 (a fragment from 7th diagonal), or from fragments of sizes 4 (a fragment from 4th diagonal) and 6 (a fragment from 6th diagonal), or two fragments of size 5 (fragments from the 5th diagonal). Figure 4.11 shows which cells are accessed to populate a cell in the 8th diagonal. Each of these fragments has to be chosen randomly, which means fragments of what lengths are to be combined is picked randomly. If we are filling out one cell, the random number will tell us which pair of cells from the lower diagonal(s) to access to pick out fragments to be combined. Also for each the two needed fragments, we use a random number to pick out one of the five fragments found in a particular cell that is picked.

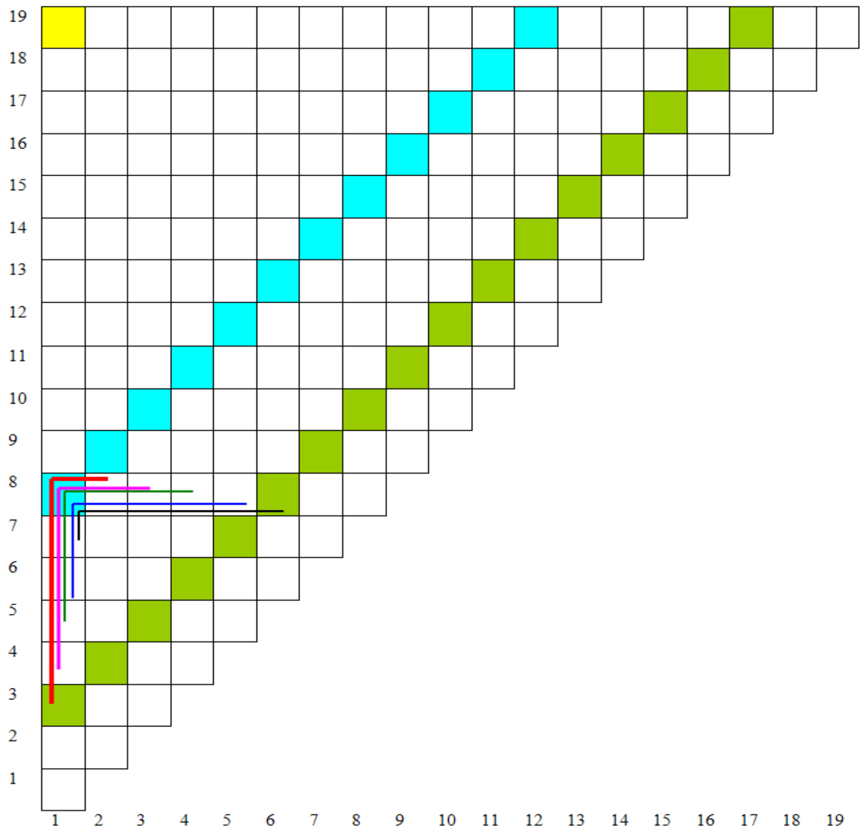


Figure 4.11 The first cell of the 8th diagonal will contain fragments built from two other fragments taken from other cells. The different colored lines show the possible places from which a pair of fragments to be joined can be picked from. For example, the red line shows a fragment from a cell of the same column but from the 3rd diagonal, and another from the same row but the 7th diagonal can be joined to make a fragment of the cell in the 1st column of the 8th row.

Determine which cells, and which slots in the cells to access to fetch the fragments

When filling out a cell with the information of a chain, the program first determines fragments for which lower cell in the triangular matrix can be used to make the chain of this particular cells. An array called fromWhere is used to maintain such information. A random number is generated to pick out which of the entries in this array can be used to determine where to take the fragments from. fromWhere has the i and j indices of both the candidate fragments' position in the triangular matrix. Another random number is used to determine which of the five chains in the first cell should be chosen to be the fragment one, and another random number is used to determine which of the five chains in the second cell should be chosen to be the fragment two. These two fragments are then joined together to form a longer fragment. Information of this longer fragment is then stored in one of the five available slots of the particular cell if none of the residues of this chain has steric clash. If there is a steric clash, that is any of the residue is closer than 3Å to any other residues in the chain, then another set of random numbers is generated to determine the combination of the fragments to be used to build the chain.

Fitting the fragments together by a series of transformations

It is finally necessary to superimpose the last two residues of the first fragment onto the first two residues of the second fragment as shown in figure 4.12, where B and C of Fragment 1 fit onto B' and C' of the second fragment.

Functions from the transformation library are extensively used to fit one fragment on the other. Our aim was to let the fitting fragments have geometries, i.e. angles that are similar to the ones found in the nature. So first it is necessary to calculate what dihedral angle would be made between the plane that contains the last three residues of the first fragment and the first three residues of the second fragment, so that they can be altered if needed to match a more natural geometry. Using some of the transformation functions, the last residue C and the second last residue B of the first fragment third last residue of the first fragment A are brought to the origin. Similarly, the first three residues of the second fragments, B', C' and D' are also moved to the dihedral angle between these four residues, lying at the origin A, B/B', C/C', D' are then calculated. As shown in Figure 4.12, it is seen that the residue B overlaps with residue B', and C overlaps with residue C'.

The best possible dihedral angle that can be formed between the two theta angles in the nature is obtained using the function **findBestAlpha**. findBestAlpha is a function whose aim is to suggest the best dihedral angle that can be contained within two given theta angles. This function gives the suggestion by using the data collected (in the angles array) which contains a list of groups of theta1, theta2 and alpha angles. In other words, the dihedral angle is looked up in a look-up table maintained in an array that contains sets of angles found in the nature. These sets of angles are those found in a real protein. This function takes as parameters the angle formed by the last three residues of the first fragment, and the first three residues of the second fragment as theta1 and theta2. When a dihedral angle is looked for, there might not be an exact match of the pair of angles input, and the pairs of angles in the array that is maintained in the angles array. So it finds the nearest or closest theta angle pairs from the angles array, i.e. the ones that have the shortest distances from the pair of input theta angles using simple sorting. The sorting is done based on the Manhattan distances each of the theta angles pairs have from the input theta angle pair. Once the nearest theta angle pair is found, the corresponding alpha or dihedral angle to that theta pair is selected and returned. This alpha angle has to be assigned in the generated decoy structure in between the planes of the combining residues, in order to give it a more natural structure.

The assignment of the new dihedral angle is done by adjusting the existing dihedral angle in between the two fragments, the value of which is already calculated. To make this adjustment, the difference (deltaAlpha) in the torsion angles is calculated, and using a transformation corresponding to rotation around the z axis, further rotation is done through deltaAlpha in the place where the fragments are overlapped.

Now a series of more transformations are done to do the fitting of the two fragments, with the new dihedral angles. The program calculates the transformation matrix that fits the second fragment to the first fragment with the desired alpha angle. It then copies all the residues of the first fragment, and applies the obtained transformation matrix to the residues of the second fragment starting from the third residue onwards. Finally the program does an inverse of all these transformation to get the required single fragment. Figure 4.12 shows how two chains are combined with overlapping and Figure 4.13 shows how the difference in dihedral angle is determined.

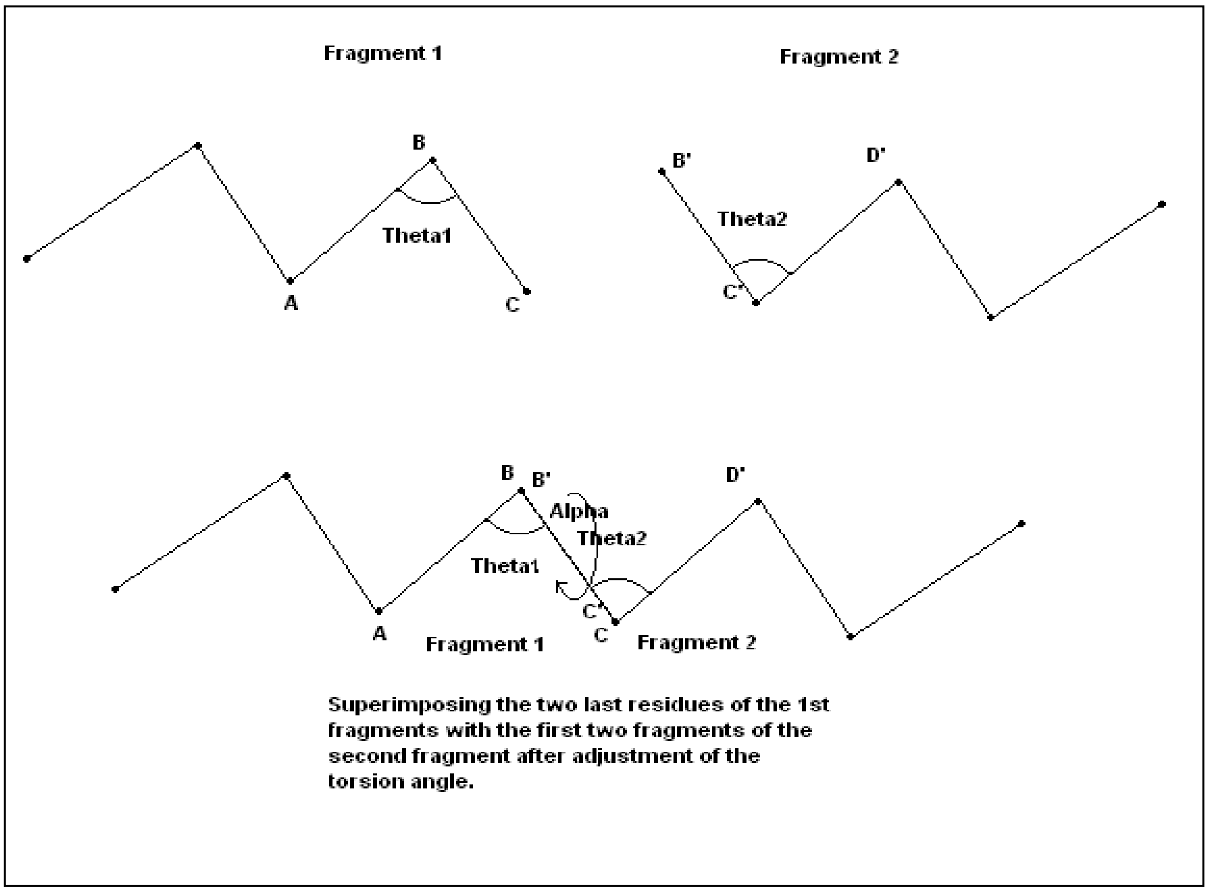

Fragment 1
Fragment 2
B
Theta1
A
C
B'
D'
Theta2
C'
B
B'
D'
Alpha
Theta2
Theta1
C'
A
Fragment 1
C
Fragment 2
Superimposing the two last residues of the 1st fragments with the first two fragments of the second fragment after adjustment of the torsion angle.

Figure 4.12 Two fragments, each of length five are brought together and the last two residues of the first fragment is overlapped by the first two residues of the second fragment to form a fragment of 8 residues.

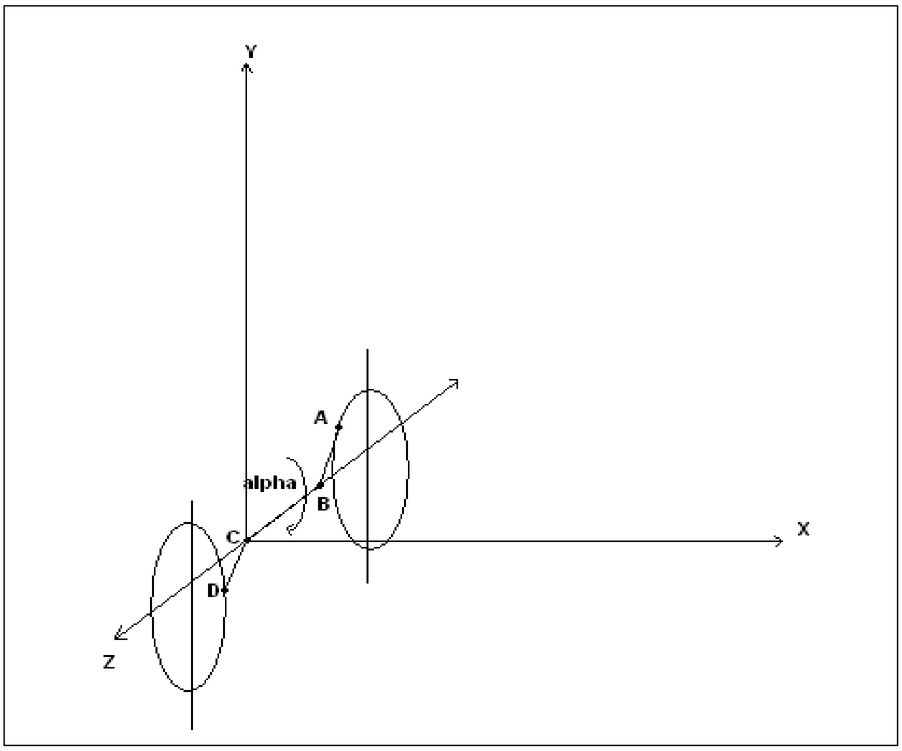


Figure 4.13 The four residues at a junction are brought to lie near the negative Z axis, with B C lying on the axis, so that the difference between the value of the dihedral angle at the formed junction and that of the suggested one can be calculated so that the angle can be adjusted to match the suggested one.

5 Similarities and differences with other works

The system that was built has similarities with some of the systems which were studied to obtain some ideas about how to approach this project. Still it had some novelties and differences to all of the previously built systems. The similarities and differences with the different algorithm and different approaches are described below in detail.

5.1 CKY and ZAMDP algorithms

The system that has been built for the project is implemented on an algorithm very similar to the CKY or more closely to the ZAMDP algorithm described by Dill, et al. (2007). My system also breaks a whole protein sequence in very small fragments first. By organizing the information about these small fragments in cells of a triangular matrix, it gradually combines the information of small fragments, and continues to build longer structure or decoys, and eventually forms structures of the same length as the original structure. A triangular array or matrix, is used to store the information of the fragments. Each cell in the array can hold information of five individual fragments. CKY fills out a triangular parse chart in the bottom-up form by first filling out the cells in the main diagonal with individual words, and then moves to the next diagonal level, and fills out the cells chart[i][i+1], and then the next level, i.e. cells chart[i][i+2], so on and so forth with information of parsing trees until it reaches the corner most cell, or the topmost cell, chart[1][n]. Just like in The CKY algorithm, this system also fills up the diagonals layer by layer with information about the smaller decoy fragments, starting from the third diagonal layer from the bottom till it reaches the cell in the top-most corner.

Instead of keeping information of parsing trees, in each of the cells my system stores information about the smaller decoy fragments. Instead of starting to fill out the chart from the first diagonal, my system starts filling out the cells in the chart starting from the third diagonal. For a protein sequence of length, N, the third bottom diagonal, that is cells with indexes CELL[i][i+2], till CELL[N-2][N], is filled first. Each of these cells has the information of five pseudo-protein chain of length three. CELL[1][3] has info about representation of chain starting from the residue number 1 to 3. Similarly, CELL [5][7] has information about chain starting from residue number 5 to 7. That is [i][i+2] has information about chain from residue number i to i+2. Then the next diagonal layer, that is layer uses information from this bottom most layer to generate information for chains of length four. Thus CELL [1][4] has information required to represent chain from residue number 1 to 4. That is CELL [i][i+2+1] till CELL[N-2-1][N], are next filled. Then similarly the next layer of diagonal is filled, and so on and so forth.

Dill et al. used HP model in two dimensional square lattice to test their hierarcical test principal. The amino-acids were categorized as hydrophobic (H) and polar (P) monomers. They used short chains of not more than 20 monomers and used exhaustive search methods to find all the possible conformations of the chains formed in the lattice using an energy function based on number of HH contacts present. With help of this model they tested if their ZAMDP algorithm can pick the optimal conformation. However, HP model was not incorporated in the work that was done in this thesis and an off-lattice model was used.

5.2 Fragment picking approach

When one cell is filled, it requires fetching of information of two smaller fragments of two sizes from two different cells from lower diagonals. In my system the choice is made by a series of random numbers. One of these random numbers is used to pick out the combination the two lower cells that can provide the information of the smaller fragments. Each of the other two random numbers help to choose which of the five fragments should be picked from each of the two chosen cells. Unlike, Hamelryck et al. (2006), complicated *HMM* is not used, rather simple random number generators provided by C library functions are used to pick out the new fragments. Hamelryck et al. (2006) have used secondary structure fragments, which is also not used in my system.

5.3 Determining the stabilities of the structure generated

Dill et. al (2007) use the HP-Model to determine energy of each of the chains formed, but in my system, that is not done. Instead of calculating an energy value for each of the fragments formed, my system looks if there is any steric clash in a fragment formed. This procedure resembles that described in Hamelryck et al. (2006). Similar to their approach, any fragments having a steric clash, that is having alpha carbons in close proximity, are discarded, and their information are not stored any more. Random numbers are generated to suggest one more set of fragments to build the another fragment.

Kolodny and Levitt (2002) ensures self-avoidance of the residues placing any two alpha carbons are separated by at least 2.5 Å. Hamelryck, et al. (2006) used 4Å to be critical distance between any two residues to avoid steric clashes. My system checks for steric clashes and discards a structure if any two alpha carbon atoms in it are placed at a distance less than 3 Å. Kolodny and Levitt (2002) enforce compactness by requiring the decoys not to exceed a range of sizes.

5.4 Representation of the backbone structure

Hamelryck et al. (2006) uses only an alpha carbon backbone to represent the protein chains, and thus call these chains decoys as they lack many information present about the real protein chain. Decoys are so called also because they are not the real native structures, but are close representation of them, or their fragments. I have adopted this term decoy, from their paper, as I am also using only alpha-carbon backbones, and trying to build fragments of protein-like-structures.

Hamelryck et al. (2006) also describe how, angles subtended by three adjacent alpha carbons (denoted by ‘Theta’), and the torsion or dihedral angles between the two planes (denoted in their paper by ‘Tao’) can describe the conformation of a structure. My system also uses this information about the theta angles, and the dihedral angles of the different fragments, in order to combine or join two individual fragments together. The theta angles and the dihedral angles are maintained as bigger fragments are built. When the smallest fragments are built, theta angles are chosen randomly from a list of sample theta angles typically to be found in a similar protein structure. Also when two fragments are combined together, the theta angle between the last three residues of the first fragments, and the theta angle between the first three residues of the second fragment are noted to suggest a dihedral angle that should be between the planes of the two combining fragment. This suggestion is also made using a look up table of combination of two adjacent theta angles and a typical dihedral angle in between them. Adjustments are made while combining the two fragments, so that the dihedral angle matches the one suggested from the lookup table.

5.5 Combining the fragments

Kolodny and Levitt (2002) uses assembling of small fragment taken from a fragment library to build larger decoys. These fragments from the library form the building blocks of the decoys, as they are picked and added to each other to lengthen the decoy chain. Each added fragment is placed on another by overlapping the first three residues of the fragment on the last three residues of the growing chain. The system that I built joins the two fragments by overlapping the last two residues of the first fragment on the first two residues of the second fragment. In my system, only the shortest fragments are taken from a library of real conformations from known structures and larger fragments are assembled from these.

6 Results

The focus of my project was to build a system that can take a known protein sequence and the information about naturally occurring geometries of that protein, to assemble a number of protein-like structures or decoys, which do not have any steric clashes. The system has generated decoy structures using the geometries of Z(Taq), an affibody complex with PDB identification number 2B88. Affibody proteins form a class of engineered binding protein. The native structures of affibody complexes are already determined and are available in the protein database. These structures are characterized with 58 residues arranged in three-helix bundle (Lendel, et al., 2006). The geometries of the Z(Taq) complex were obtained from its PDB file. The angles formed by the consecutive residues and the dihedral angles formed by the planes in which these residues lie were used to build the decoys in the system.

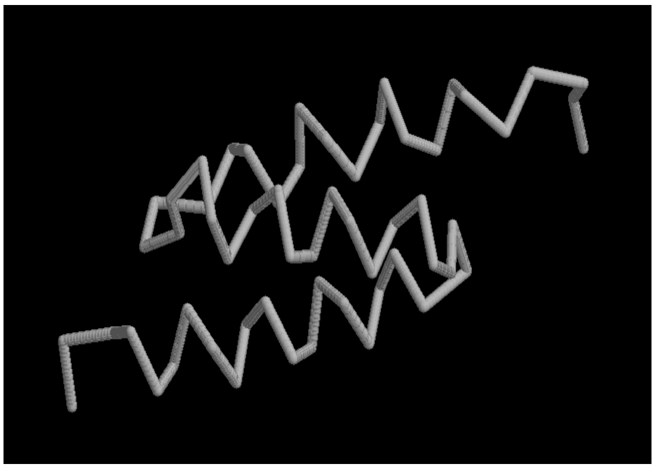


Figure 6.1 An image showing Z(Taq) Affibody structure drawn using RASMOL

Figure 6.1 shows an image of Z(Taq) using RASMOL, a software used for viewing the structures of proteins, DNA's and other macromolecules. My system tries to build decoy structures using natural geometries found in Z(TAQ). It generates decoys composed of helical structures. The helical structures are present as the geometries used for building the decoys are taken from a protein complex which three helices in it.

The five decoy structures generated in my system where drawn using RASMOL. The images are shown in Figure 6.2 in the next page. Decoy building of this level can be used for prediction of structures of unknown or newly found proteins.

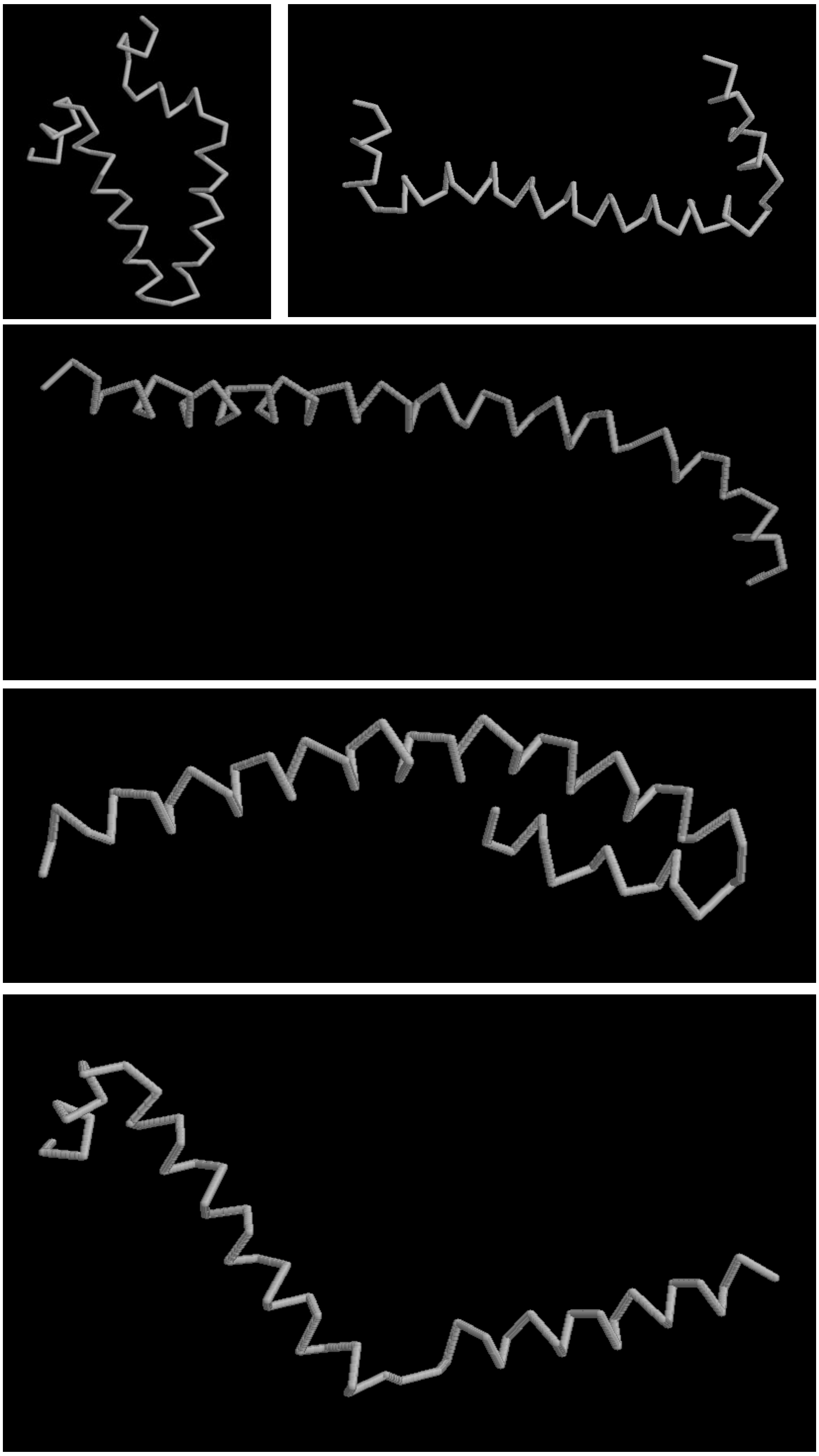


Figure 6.2 Images of the five decoys generated by the system developed.

7 Conclusion

7.1 Achievements

The aim of this project was to build a system that can assemble decoys from small fragments of proteins using a dynamic programming algorithm. It was aimed towards producing decoys which would have geometries similar to the ones of naturally occurring protein and would have no steric overlaps. Thus this project was an *ab initio* approach towards building decoy structures. The system achieves the aimed task very well, as it succeeds in building decoy structure with natural-like geometries and no steric clashes. It generated decoys with expected secondary structures. The system builds five decoy structures of full length. However, a slight modification in the code can allow it to build more of these structures.

The system used a well structured data structure, to maintain the information required and used quite an efficient algorithm which assembled protein like decoys which did not have any steric clashes. When the system was given the task to assemble decoys, it generated them quickly.

7.2 Limitations

The system used real values of angles (subtended by consecutive alpha carbons) found in real protein chains with known structures. The torsion or dihedral angles which were assigned to the structure at their location of junction during joining the two fragments are also picked out from a list of torsion angles found in the same known structure. Still the resulting structures produced by the system had slightly different torsion angles, than what is tried to be assigned to them. This slight difference might have been resulted from rounding of errors adding up at every stage of joining of the small fragments.

Even though assembling the decoys was successful, the ones generated were found to be not very compact as expected in a natural protein structure. Compactness in the resulted structures probably is not achieved because of the check against steric clash, where the system omitted generated structures which had alpha carbons lying closer than a threshold distance. More compactness could have been achieved if a check for radius of gyration was also included in the system, which would have omitted structures with a radius of gyration over a threshold value as done by Hamelryck, et al. (2006).

7.3 Possible improvements and future work

In spite of the limitations in the system, it still carries out its task fairly well. Still, there remains an opportunity to make a great deal of improvement on this system. As mentioned earlier, if a check for compactness is included, by only allowing structure within a certain radius of gyration, more real-looking decoy structures can possibly be generated.

At the moment, the system does not score the fragments generated at each stage of assembling, with any energy potential or scoring function. A scoring can be maintained, and the structures generated at each stage can be sorted depending on their score, and the best ones can be used for the assembling process in the next stage.

For now, the constituent fragments are picked randomly at each stage. At a particular stage, any of these fragments has almost equal probability of being picked for usage in the next stage of assembling. As discussed in the literature, complex stochastic functions can also be used to determine this choice of fragments. HP model can be incorporated also for much better results, i.e. the positions of hydrophobic and polar residues can be used to determine the scoring function of a generated fragment. As mentioned above, sorting of best scored results can also improve with the choosing process.

The current system can serve as a good framework for a decoy predicting system. Its implementation can be easily modified to include the complex scoring functions which can lead to much better results. Thus, any future work on the current system is warmly encouraged.

Reference

Bates, M., 1995 *Models of natural language understanding*, Proc. Natl. Acad. Sci. USA 92, pp. 9977-9982.

Berman, H., Henrick, K., Nakamura, H., 2003. *Announcing the worldwide Protein Data Bank*, Nat. Struct. Biol., 10, pp. 980.

Bernstein, F. C., Koetzle, T. F., Williams, G. J. B, Meyer Jr, E. F., Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T., Tasumi, M., 1977. *The Protein Data Bank a computer-based archival file for macromolecular structures*, Eur. J. Biochem. 80, pp. 319-324.

Bonneau, R., Baker, D., 2001. *Ab initio protein structure prediction: Progress and Prospects*, Biomol. Struct. 2001.30, pp. 173–89.

Dill, K.A., Lucas, A., Hockenmaier, J., Hunag, L., Chinag,D., Joshi, A.K., 2007. *Computational linguistics: A new tool for exploring biopolymer structures and statistical mechanics*, Polymer, 48, pp. 4289-4300.

Eddy, S. R., 2004. *What is dynamic programming?,* Nature Biotechnology 22(7), pp. 909-910.

Hamelryck,T, Kent, J.T., Krogh, A., 2006. *Sampling realistic protein conformation using local structural bias.* PLoS Comput Biol 2(9): e131. DOI: 10.1371/journal.pcbi.0020131.

Hockenmaier, J., Joshi, A.K., Dill, K., 2006. *Routes are trees: The parsing perspective on protein folding*, PROTEINS: Structure, Function, and Bioinformatics 66, pp. 1-15.

Keasar, C., Levitt, M., 2003. *A novel approach to decoy set generation: Designing a physical energy function having local minima with native structure characteristics*, J. Mol. Biol., 329, pp. 159-174.

Kolodny, R., Levitt, M., 2002. *Protein decoy assembly using short fragments under geometric constraints*, Biopolymers, 68, pp. 278–285.

Lendel, C., Dogan, J., Hard, T., 2006. *Structural basis for molecular recognition in an affibody: affibody complex*, J.Mol.Biol. 359, pp. 1293-1304.

Lobanov, M. Yu., Bogatyreva, N. S., Galzitskaya, O. V., 2008. *Radius of Gyration as an indicator of protein structure compactness,* Molecular Biology, 42(4), pp. 623–628

Phillips, R., Kondev, J., Theriot, J., 2009. *Physical biology of the cell,* Garland Science

Pietzsch, J., 2003 *The importance of protein folding.* [Online] Available at: http://www.nature.com/horizon/proteinfolding/background/importance.html [Accessed 30 April 2010].

Plaxco, K. W., Simons, K. T., Baker , D., (1998). *Contact order, transition state placement and the refolding rates of single domain proteins,* J. Mol. Biol., 277, pp. 985-994