A Tree-Decomposition Approach to Protein Structure Prediction

Jinbo Xu
j3xu@theory.csail.mit.edu
Department of Mathematics and CSAIL, MIT
Cambridge, MA 02139

Feng Jiao
fjiao@cs.uwaterloo.ca
School of Computer Science
University of Waterloo
Waterloo, Canada N2L 3G1

Bonnie Berger
bab@csail.mit.edu
Department of Mathematics and CSAIL, MIT
Cambridge, MA 02139

Abstract

This paper proposes a tree decomposition of protein structures, which can be used to efficiently solve two key subproblems of protein structure prediction: protein threading for backbone prediction and protein side-chain prediction. To develop a unified tree-decomposition based approach to these two subproblems, we model them as a geometric neighborhood graph labeling problem. Theoretically, we can have a low-degree polynomial time algorithm to decompose a geometric neighborhood graph $G = (V, E)$ *into components with size* $O(|V|^{\frac{2}{3}} \log |V|)$. *The computational complexity of the tree-decomposition based graph labeling algorithms is* $O(|V|\Delta^{tw+1})$ *where* Δ *is the average number of possible labels for each vertex and* $tw(= O(|V|^{\frac{2}{3}} \log |V|))$ *the tree width of* G. *Empirically,* tw *is very small and the tree-decomposition method can solve these two problems very efficiently. This paper also compares the computational efficiency of the tree-decomposition approach with the linear programming approach to these two problems and identifies the condition under which the tree-decomposition approach is more efficient than the linear programming approach. Experimental result indicates that the tree-decomposition approach is more efficient most of the time.*

1 Introduction

The structure of a protein plays an instrumental role in determining its functions. Protein structures are important for the understanding of life process and drug discovery. However, existing experimental methods such as X-ray crystallography and NMR techniques cannot generate protein structures in a high throughput way. In order to produce protein structures in a large scale, NIH has launched a protein structure initiative. This initiative aims to experimentally determine a few thousands of unique protein structures within 10 years so that most of new proteins can have a similar structure in the Protein Data Bank (PDB). Therefore, the structures of these new proteins can be predicted using template-based methods such as homology modeling and protein threading. Computational approaches to protein structure prediction are becoming useful and successful, as demonstrated in recent CASP competitions [1, 2, 3]. Indeed, protein structure prediction tools have been routinely used by structural biologists and pharmaceutical companies to analyze the structural features and functional characteristics of a protein.

Generally speaking, the 3D structure of a new protein is predicted in the following four steps. First, the backbone skeleton of a protein is predicted using a backbone prediction technique such as protein threading and homology modeling. Secondly, loops are added to connect the backbone segments together to form a complete backbone conformation. The first two steps are usually conducted in a single step if the *ab inito* folding method is used to predict the structure of a protein. Then side-chain orientation is assigned so that we have a full-atom model for the new protein. Finally, some molecular dynamic simulation techniques can be used to further refine the predicted structure.

Unfortunately, both the protein threading problem and the protein side-chain prediction problem are NP-hard [4, 5, 6, 7]. Many heuristic algorithms and programs have been developed to solve these two problems [8, 9, 10, 11, 12, 13, 14, 15]. In this paper, we present a tree-decomposition based approach to decompose a protein structure into some

Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference (CSB'05)
0-7695-2344-7/05 $20.00 © 2005 IEEE

small pieces. This tree-decomposition method enables us to solve both the protein threading problem and the protein side-chain prediction problem. In order to solve these two problems using the same method, we formulate them to a geometric neighborhood graph labeling problem. Experimental results show that using the tree-decomposition method, we can solve both problems very efficiently. The tree-decomposition based approach to the threading problem runs faster than the linear programming approach [16, 17] most of the time. We also identify the condition under which the tree-decomposition approach is more efficient than the linear programming approach. If we combine both approaches, then we can achieve a better efficiency than any single method.

The major contributions of this paper are: (i) formulating both the protein threading problem and the side-chain prediction problem into a sparse graph labeling problem; (ii) proposing a tree-decomposition based approach to the protein threading problem; and (iii) identifying the rule by which we can easily tell for a given threading instance, which method is more efficient, the tree-decomposition approach or the linear programming approach.

The remainder of this paper is organized as follows. Section 2 formulates the threading problem and the side-chain packing problem into a geometric graph labeling problem. In Section 3, we introduce the concept of tree decomposition, several different tree-decomposition methods, and tree-decomposition based label assignment algorithm. Theoretically, a low-degree polynomial-time algorithm exists to decompose a sparse geometric neighborhood graph into some components of size $O(|V|^{2/3}\log|V|)$. In Section 4, we present the experimental results of our algorithm in detail and compare the tree-decomposition based approach with the linear programming approach in terms of computational efficiency and prediction accuracy. Finally, Section 5 draws some conclusions.

2 Problem Formulation

This section briefly introduces the protein threading problem and the side-chain prediction problem and then formulate them to a geometric neighborhood graph labeling problem.

2.1 Protein Threading

Here we briefly introduce the protein threading problem. For its detailed description, please refer to Xu et al.'s paper [16, 17]. Protein threading is one of the most successful methods for protein structure prediction. It is a pattern-matching or template-based structure prediction method. Protein threading predicts the structure of a new protein by first aligning its sequence to all the existing structures and then finding the structure with the best alignment to the sequence. An existing structure is also called a structural template. The basic premise of protein threading is that in nature there are only a few thousands of unique protein structures. Therefore, most new proteins can find a similar structure in PDB.

A structural template can be modeled using a template contact graph as follows. The primary structure of a template is parsed as a linear series of cores with a connecting loop between two adjacent cores. Cores are the most conserved segments in a protein structure. When aligning a protein sequence with structure to be predicted to a template, alignment gaps are confined to loops. The biological justification is that cores are so conserved that the chance of insertions or deletions within them is very slim. We consider only interactions between residues in the cores. It is generally believed that interactions involving loop residues can be ignored as their contribution to fold recognition is relatively insignificant. We say that an interaction exists between two residues if the spatial distance between their C_β atoms is within $7\dot{A}$ and they are at least 4 residues apart along the template sequence. We say that an interaction exists between two cores if there exists at least one inter-residue interaction between the two cores. We can model a protein structural template using a template contact graph $G = (V, E)$. Each template core is represented by a vertex in V. There is one edge between two cores if and only if an interaction exists between them.

Therefore, the protein threading problem can be formulated as follows [16, 17]. Let $D[i]$ denote the set of possible alignment positions for core i. For each possible alignment position $l \in D[i]$, there is an associated singleton score, denoted by $S_i(l)$. This singleton score measures how well to align core i to sequence position l. In our energy function, $S_i(l)$ includes mutation score, environmental fitness score and secondary structure score. For any two alignment positions $l \in D[i]$ and $k \in D[j]$ $(i \neq j)$, there is also an associated pairwise score, denoted by $P_{i,j}(l,k)$, if there is an edge between core i and core j. $P_{i,j}(l,k)$ is the interaction score between cores i and j when their alignment positions are l and k, respectively. In the sequence-template alignment, there is no crossover allowed. That is, if $i < j$, then k must be larger than l plus the length of core i. In order to guarantee a valid alignment, we can set $P_{i,j}(l,k)$ to be $+\infty$ when crossover occurs. For the alignment involved with the loop regions of a template, some gaps may exist. In order to penalize gaps, the scoring function also contains some gap penalty. Assume that core i is aligned to sequence position $A(i)$. The quality of this sequence-template alignment is measured by the following energy function.

$$E(G) = \sum_{i \in V} S_i(A(i)) + \sum_{i \neq j, (i,j) \in E} P_{i,j}(A(i), A(j)) \quad (1)$$

The smaller the system energy $E(G)$ is, the better the sequence-template alignment.

2.2 Protein Side-Chain Prediction

Assume that we have a protein backbone structure, the task of side-chain prediction is to assign a side-chain conformation to each backbone position. Usually, for each backbone position, there are many possible side-chain conformations (also called rotamers), each with an occurring probability. When we assign side chains to the backbone, we need not only to pick up those side-chain conformations with high occurring probability, but also to avoid as many atomic clashes as possible. Two atoms clash if and only if their distance is less than the sum of their radii. For a detailed description of the side-chain prediction problem, please refer to Xu's paper [18]. The side-chain prediction problem can be formulated as follows. We use a *residue interaction graph* $G = (V, E)$ to model the residues in a protein and their potential clash relationship. Each vertex in V denotes the center of one residue. Let $D[i]$ denote the set of possible rotamers for residue i. Two residues have potential atomic clashes if some of their roatmers have atomic clashes. We add one interaction edge between two residues if they have potential clahses. For each rotamer $l \in D[i]$, we use $S_i(l)$ to denote its associated singleton score, which is the interaction energy between rotamer l and the backbone of the protein. The singleton score $S_i(l)$ also includes an item reflecting the occurring probability of this rotamer. If some atoms in two rotamers $l \in D[i]$ and $k \in D[j]$ $(i \neq j)$ clash, then we use $P_{i,j}(l, k)$ to denote an interaction score between these two roatmers, which is the clash penalty between them. Given a side-chain assignment $A(i) \in D[i]$ to residue i $(i \in V)$, the quality of this side-chain packing is measured by the following energy function.

$$E(G) = \sum_{i \in V} S_i(A(i)) + \sum_{i \neq j, (i,j) \in E} P_{i,j}(A(i), A(j)) \quad (2)$$

The smaller the system energy $E(G)$ is, the better the side-chain assignment.

2.3 Graph Labeling Problem

Both the protein threading problem and the protein side-chain prediction problem can be formulated as a problem of assigning some labels to a sparse geometric neighborhood graph $G = (V, E)$. In a geometric neighborhood graph, each vertex v in V represents a point in $\Re^3$ and v can be adjacent to only those vertices that are spatially close to it. That is, no edge exists between two vertices if their spatial distance is beyond a cutoff D_u. Both the template contact graph and the residue interaction graph are a sparse geometric neighborhood graph. First, a vertex in these two graphs represent a geometric object and can be treated as a 3D point. Secondly, in the protein template, the core length is much smaller compared to the template length. We can assume that the core length is constant. So there is no interaction edge between two cores if their spatial distance is large. For the side-chain packing problem, each rotamer has a constant diameter and is bounded to its backbone position by a constant distance. So there is no edge between two backbone positions (residues) if their distance is beyond a constant distance. In addition, in a normal protein, the distance between two residues cannot be too close. This indicate that each vertex v can only be adjacent to a limited number of vertices. Therefore, both the template contact graph and the residue interaction graph are a sparse geometric neighborhood graph.

Both the protein threading problem and the side-chain prediction problem can be formulated into a sparse geometric neighborhood graph labeling problem as follows. In the threading problem, each possible sequence alignment position for a single core can be treated as a possible label assignment to a vertex in G. For the side-chain prediction problem, a possible label assignment to a vertex represents a possible side-chain conformation. Let $D[i]$ denote the set of possible label assignments to vertex i. For each possible label assignment $l \in D[i]$, there is an associated singleton score, denoted by $S_i(l)$. This singleton score measures how well to assign label l to vertex i. For any two label assignments $l \in D[i]$ and $k \in D[j]$ $(i \neq j)$, there is also an associated pairwise score, denoted by $P_{i,j}(l, k)$, if there is an edge between vertex i and vertex j. $P_{i,j}(l, k)$ is the interaction score between vertices i and j when their label assignments are l and k respectively. Given a label assignment $A(i) \in D[i]$ to vertex i $(i \in V)$, the quality of this assignment is measured by the following energy function.

$$E(G) = \sum_{i \in V} S_i(A(i)) + \sum_{i \neq j, (i,j) \in E} P_{i,j}(A(i), A(j)) \quad (3)$$

The smaller the system energy $E(G)$ is, the better the label assignment. Our goal is to optimize the above energy function to obtain the best label assignment.

3 A Tree-Decomposition Approach to The Graph Labeling Problem

In this section, we first introduce the concept of tree decomposition, then describe several different methods to decompose a graph into a tree decomposition, and finally describe how to search for the optimal label assignment based on the tree decomposition of a graph.

3.1 Tree Decomposition

Tree decomposition of a graph has been introduced by Robertson and Seymour [19] since a long time ago. The technique of decomposing a sparse graph to its tree-decomposition has been applied to many NP-hard problems such as frequency assignment problem [20] and Bayesian inference [21].

Definition 3.1 *Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair (T, X) satisfying the following conditions:*

1. *$T = (I, F)$ is a tree with a node set I and an edge set F,*
2. *$X = \{X_i | i \in I, X_i \in V\}$ and $\bigcup_{i \in I} X_i = V$. That is, each node in the tree T represents a subset of V and the union of all the subsets is V,*
3. *for every edge $e = \{v, w\} \in E$, there is at least one $i \in I$ such that both v and w are in X_i, and*
4. *for all $i, j, k \in I$, if j is a node on the path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.*

The width of a tree decomposition is $\max_{i \in I}(|X_i| - 1)$. The tree width of a graph G, denoted by $tw(G)$, is the minimum width over all the tree decompositions of G.

Figure 1 and 2 give an example of a graph and one of its tree decompositions, respectively. The width of a tree decomposition is a key factor in determining the computational complexity of all the tree-decomposition based algorithms. The smaller the width of a tree decomposition is, the more efficient the tree decomposition based algorithms. Therefore, we need to optimize the tree decomposition of a graph such that we can have a very small tree width.

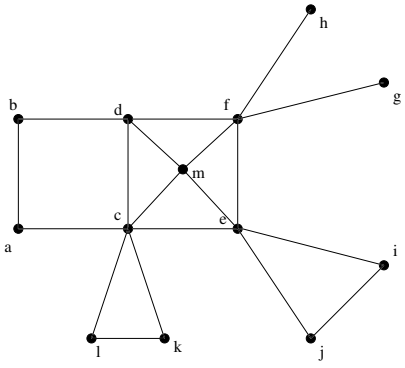

Figure 1. Example of a graph.

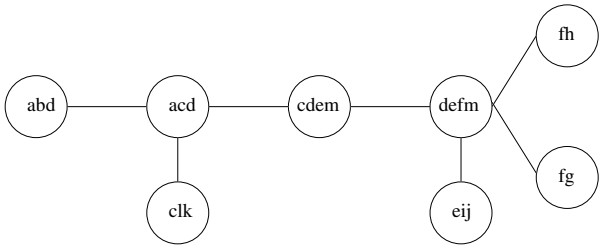

Figure 2. Example of a tree decomposition with width 3.

3.2 Algorithms for Tree Decomposition of A Graph

The optimal tree decomposition of a general graph is *NP*-hard [22], which means it is unlikely to find the optimal tree-decomposition of a graph within polynomial time. The graph discussed in this paper has two special features: (i) the graph is a geometric neighborhood graph and (ii) the graph is also sparse. Therefore, using the sphere separator theorem [23], we can have a low-degree polynomial-time algorithm to decompose the graph into some components with size $O(\frac{D_u}{D_l}|V|^{\frac{2}{3}} \log |V|)$ [18]. Strictly speaking, we have the following theorem.

Theorem 3.2 *Let $G = (V, E)$ denote a graph describing the relationship among a set of 3D points. The distance between any two vertices in G is at least a constant D_l and the distance between any two adjacent vertices is no more than D_u. There is a low-degree polynomial-time algorithm to decompose G into some components with size $O(\frac{D_u}{D_l}|V|^{\frac{2}{3}} \log |V|)$.*

The sphere separator based tree decomposition algorithm is not easy to implement. Besides this theoretically-sound tree decomposition method, many heuristic-based tree decomposition methods exist in the literature. Later in this paper we will compare the performance of six heuristic-based tree decomposition algorithms and show that some work very well in practice. The graph under consideration can be decomposed into some very small components, which leads to a very efficient tree-decomposition based graph labeling algorithm. Here we describe six different tree decomposition algorithms.

Method 1: minimum vertex separator This method recursively partitions a graph into two disconnected subgraphs using a minimum vertex separator [24]. Finally, we get a tree with separators being the internal nodes and the final subgraphs being the leaf nodes. All the vertices along the path from the tree root node to any tree node form a decomposition component. Given an undirected

graph $G = (V, E)$, a vertex set S is called an (a, b)-*vertex-separator* if it satisfied (i) $\{a, b\} \subset V \setminus S$ and (ii) every path connecting a and b in G passes through at least one vertex contained in S. Among all the separators, the one with the minimum cardinality is called the minimum (a,b)-*vertex-separator*.

Method 2: two-way-1/2-triangle This method is similar to the minimum vertex separator method. The only difference is that this method uses a minimum 1/2-balanced vertex separator rather than the minimum vertex separator. The "1/2-balanced" vertex separator always partitions a graph into two subgraphs, each containing no more than one half of all the vertices.

Method 3: two-way-2/3-triangle This method is similar to the two-way-1/2-triangle method. The only difference is that this method uses a minimum 2/3-balanced vertex separator rather than a minimum 1/2-balanced vertex separator. The "2/3-balanced" vertex separator always partitions a graph into two subgraphs, each containing no more than two thirds of all the vertices.

Method 4: minimum-degree It is a heuristic algorithm that iteratively chooses one vertex and form a decomposition component based on this vertex [25]. At each iteration, this method chooses a vertex with the smallest number of neighbors and add edges to the graph such that any two neighbors of the selected vertex is connected by an edge. The selected vertex with its neighbors form a partition of the graph. Then, this method removes the selected vertex from the graph and recursively choose the next vertex until the graph is empty.

Method 5: minimum-width It is a heuristic method similar to the "minimum-degree" method. It differs from the "minimum-degree" method in that this method does not add edges to the neighbors of the chosen vertex.

Method 6: minimum-discrepancy It is a heuristic method similar to the "minimum-degree" method. It differs from the "minimum-degree" method in that at each iteration, this method selects the vertex with the minimum number of edges missing between its neighbors.

3.3 Tree Decomposition-Based Graph Labeling Algorithms

Assume that we have a tree decomposition (T, X) of a geometric neighborhood graph G. We describe an algorithm to search for the optimal label assignment based on the tree decomposition. For simplicity, we assume that tree

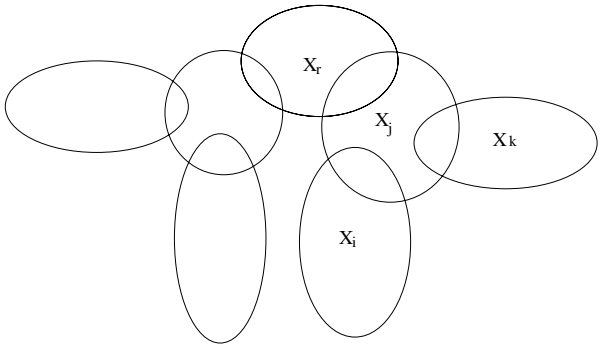

Figure 3. A tree decomposition (T, X) of G.

T has a root X_r and that each node is associated with a height. The height of a node is equal to the maximum height of its child nodes plus one. Figure 3 shows an example of a tree decomposition in which component X_r is the root. Let $X_{r,j}$ denote the intersection between X_r and X_j. If we remove all the vertices in $X_{r,j}$, then this tree decomposition becomes two disconnected subtrees. Let $F(X_j, A(X_{r,j}))$ denote the optimal label assignment of the subtree rooted at X_j given that the label assignment to $X_{r,j}$ is fixed to $A(X_{r,j})$. Then $F(X_j, A(X_{r,j}))$ is independent of the rest of the whole tree decomposition. Let $C(j)$ denote the set of child components of X_j and $Score(X_j, A(X_j))$ denote the assignment score of component X_j with the label assignment being $A(X_j)$. Let $D[X]$ denote all the possible label assignments to the vertices in X. Therefore, we have the following recursive equation.

$$F(X_j, A(X_{r,j})) = \min_{A \in D[X_j - X_{r,j}]} \{ \sum_{i \in C(j)} F(X_i, A(X_{j,i})) + Score(X_j, A(X_j)) \}$$

Based on the above equation, we can calculate the optimal label assignment in two steps. First, we calculate the optimal energy function from bottom to top and then we extract the optimal label assignment from top to bottom.

Bottom-to-Top Starting from a leaf node i in the tree T, we assume that node j is the parent of i in T. Let $X_{j,i}$ denote the intersection between X_i and X_j and $D[X_{j,i}]$ the set of all the possible label assignments to the vertices in $X_{j,i}$. Given a label assignment $A(X_{j,i}) \in D[X_{j,i}]$ to the vertices in $X_{j,i}$, we enumerate all the possible label assignments to $X_i - X_{j,i}$ and then find the best label assignment such that the energy of the subtree rooted at X_i is minimized. We use $F(X_i, A(X_{j,i}))$ to denote this minimized energy. At the same time, we also save the optimal assignment to $X_i - X_{j,i}$ for a given $A(X_{j,i})$ since in the top-to-bottom step we need it for traceback. For example, in Figure 2, if we assume the node acd is the root, then node $defm$ is an internal node with parent $cdem$. For each label assignment to vertices d,

e and m, we can find the best label assignment to vertex f such that the energy of the subtree rooted at $defm$ is minimized. In this bottom-to-top process, a tree node can be calculated only after all of its child nodes are calculated. When we calculate the root node of T, we enumerate all the possible label assignments to this node and find the optimal label assignment such that the energy is minimized. This minimized energy is also the minimum energy of the whole system.

Top-to-Bottom After finishing calculating the root node of tree T, we obtain the optimal label assignment to this root node. Now we trace back from the parent node to its child nodes to extract out the optimal label assignment to all the child nodes. Assume that we have the optimal label assignment to node j and node i is a child of j. We can easily extract out the optimal label assignment to $X_i - X_{j,i}$ based on the assignment to $X_{j,i}$ since we have already saved this label assignment in the bottom-to-top step. Recursively, we can track down to the leaf nodes of T to extract out the optimal label assignment to all the vertices in G.

In addition, based on the definition of tree decomposition, one vertex might occur in several tree nodes of the tree decomposition of G. To avoid incorporating the singleton score of this vertex into the overall system energy more than once, we incorporate the singleton score of this vertex into the system only when we are calculating the tree node with the maximal height among all the nodes containing this vertex. We can prove that there is one and only one such a tree node. Similarly, an edge in graph G might also occur in several tree nodes. We can use the same method to avoid redundant addition of its pairwise score.

Based upon the above description, we have the following lemma.

Lemma 3.3 *The tree-decomposition based label assignment algorithm for a graph $G = (V, E)$ has a computational complexity of $O((|V| + |E|)\Delta^{1+tw})$ where Δ is the average number of possible labels for each vertex, and tw is the width of the tree decomposition of G. The space complexity of this algorithm is $O(|V|\Delta^{tw})$.*

4 Experimental Results

This section compares the computational efficiency of the tree-decomposition approach and the linear programming approach to protein structure prediction and identifies the condition under which the tree-decomposition based approach is more efficient than the linear programming approach. Both approaches can solve the problems to their optimal solutions. Therefore, both approaches have the same prediction accuracy.

4.1 Protein Threading

Table 4.1 lists the performance of six heuristic-based tree-decomposition methods for the decomposition of 5280 template contact graphs. Any two templates share no more than 40% sequence identity. As shown in this table, the minimum-degree and the minimum-discrepancy methods are the best for the decomposition of the template contact graphs. This table also indicates that more than 98% template contact graphs can be decomposed into components containing no more than 6 vertices if a good decomposition method is employed.

Using the minimum-degree method to decompose a template contact graph, the tree-decomposition based protein threading method runs more efficiently than the linear programming approach when the tree width of the decomposition is no more than 5. To compare the computational efficiency of the tree-decomposition based approach and the linear programming approach, we randomly chose 1000 structural templates from RAPTOR's template database and 100 sequences from the Lindahl's benchmark [26], respectively. Any two structural templates share no more than 40% sequence identity, so do any two sequences. We threaded each sequence to each template using both approaches. Tested on a PC Linux box with a 1.7GHz CPU, it takes the linear programming approach approximately 100 hours to finish all the 100,000 threading pairs and the tree-decomposition approach approximately 58 hours. In our experiment, we used the linear program solver CLP in the COIN package to solve all the linear programs.

We further examine the condition under which the tree-decomposition approach is better than the linear programming approach. The running time of the tree-decomposition based approach is related to both the tree width of the tree decomposition, the template length and the sequence length. We calculate the average running time of threading a given sequence to all the templates with the same tree width. In Figure 4, we use a '*' to indicate that the linear programming method is more efficient than the tree-decomposition based method and a '.' to indicate the reverse situation. As shown in Figure 4, the tree-decomposition based method runs faster than the linear programming approach when the tree width is smaller than 5. From this figure we can see that when the tree width is smaller than 5, the tree-decomposition method is always more efficient than the linear programming method. If the tree width is equal to or large than 5, the linear programming method is better if the sequence length is large. When the tree width is equal to 5, the tree-decomposition based method is better if the sequence has no more than 350 residues. When the tree width is equal to 6 or 7, the tree-decomposition based method is better if the sequence length is less than 170. When the tree width is equal to 8, the tree-decomposition based method

Table 1. Performance of six different tree decomposition methods. The numbers in this table are the percentage of template contact graphs with a given treewidth.

tree decomposition method	tree width					average
	≤ 2	3	4	5	≥ 6	tree width
minimum vertex separator	18.13	10.53	11.86	12.33	47.16	5.676
two-way-1/2-triangle	18.86	20.25	16.52	22.92	21.46	4.121
two-way-2/3-triangle	21.19	14.53	29.72	19.05	15.51	3.874
minimum-degree	22.97	34.41	31.12	9.53	1.97	3.198
minimum-width	22.92	27.35	25.72	13.84	10.34	3.565
minimum-discrepancy	22.95	34.39	31.93	9.09	1.63	3.185

is better if the sequence length is less than 150. In a summary, the trend is that when the tree width is big and the sequence is long, then the linear programming method is better than the tree-decomposition method, otherwise the tree-decomposition method is better.

We can further improve the computational efficiency by combining these two methods. That is, we use the tree-decomposition based approach to the protein threading problem when one of the following conditions is satisfied: (i) the template tree width is smaller than 5; or (ii) the template tree width is equal to 5 and the sequence has no more than 350 residues; or (iii) the template tree width is less than 9 and the sequence has no more than 150 residues. Otherwise, we use the linear programming approach. Then the total running time of threading all the 100,000 pairs can be improved to 52 hours, which is approximately half of the running time of the linear programming approach.

Both the tree-decomposition based approach and the linear programming approach have the same prediction accuracy. We compare the prediction accuracy of these two approaches using thirty CASP6 test proteins, which were released from June 2004 to August 2004. These test proteins are available at the CASP6 website. The template database was generated from the PDB database in April 2004. In total there are about 5000 templates and any two templates share no more than 40% sequence identity. Table 2 lists the best template predicted for each test protein using two different threading approaches (i.e., linear programming and tree-decomposition). Both approaches generate the same top template for each test protein.

4.2 Protein Side-Chain Prediction

Table 3 lists the performance of six heuristic-based tree-decomposition methods on the 178 residue interaction graphs generated from 178 test proteins used by SCWRL [27]. This table indicates that both the minimum-degree and the minimum-discrepancy methods are the best. Using these two methods, all the residue interaction graphs can be decomposed into components containing no more than 7 residues.

The linear program formulation used in protein structure prediction package RAPTOR [16, 17] can also be used to solve the side-chain prediction problem. We implemented the linear programming approach to the side-chain prediction problem in order to compare the computational efficiency of the tree-decomposition based approach and the linear programming approach to the side-chain prediction problem. The experimental result indicates that the tree-decomposition based side-chain prediction algorithm runs slightly faster than the linear programming method. For the tree-decomposition based approach, we use the minimum-degree heuristic method to decompose all the residue interaction graphs. For the linear programming approach, we use the linear programming solver CLP in the COIN package to solve all the linear programs. Table 4 lists the computational time of both methods. The tree-decomposition based approach is slightly more efficient than the linear programming approach to the side-chain prediction problem.

We also compared the side-chain prediction accuracy of the linear programming approach and the tree-decomposition approach. For the tree-decomposition approach, we tested the above six different tree-decomposition generation methods. All the methods give the same prediction accuracy as the linear programming approach.

5 Conclusions

In this paper we presented a unified model to formulate the two key subproblems of protein structure prediction: protein threading and protein side-chain prediction. This unified model enables us to use a tree-decomposition based approach or a linear programming approach to solve both subproblems in a very similar way. Both approaches have their own advantages, but the tree-decomposition based approach is more efficient than the linear programming approach. We also obtained the rule by which for a given threading pair, we can easily choose the approach that is the

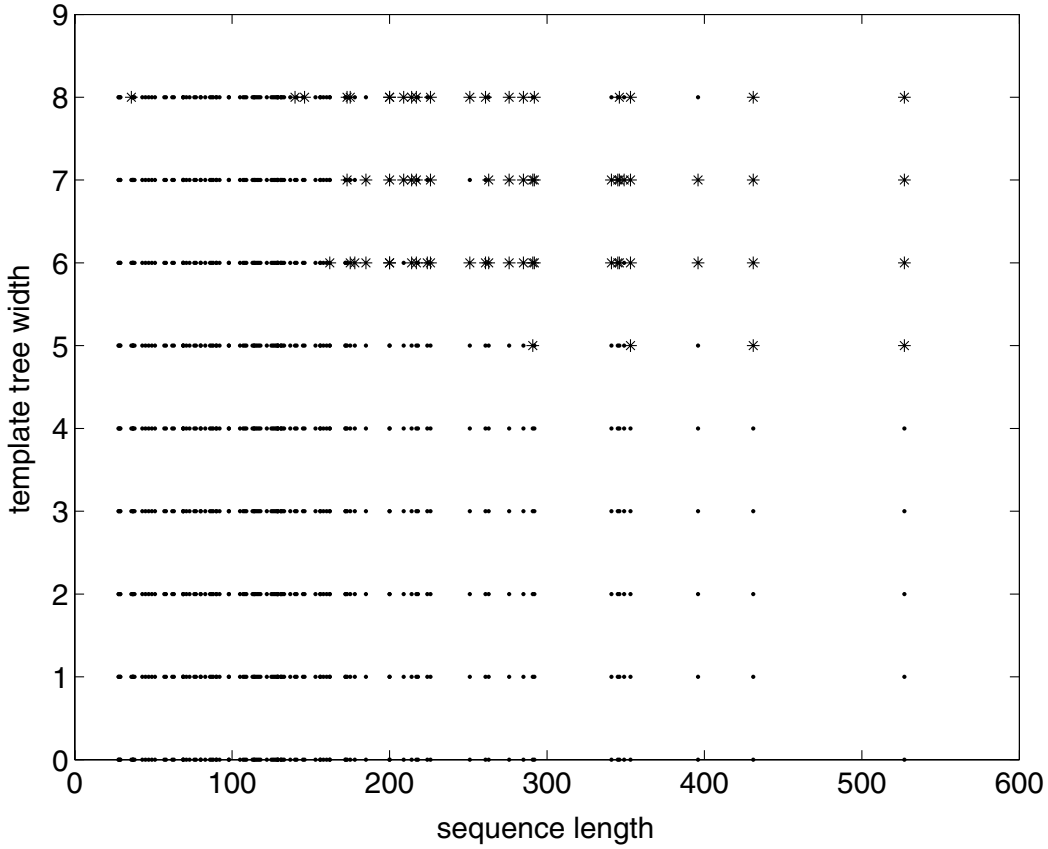

Figure 4. Computational efficiency of tree-decomposition based approach and linear programming approach to the protein threading problem. A '*' indicates that the linear programming approach is better, while a '.' indicates that the tree-decomposition based approach is better.

most efficient for aligning this pair. Combing these two approaches, we can achieve a better computational efficiency than any single method.

6 Acknowledgments

The authors gratefully acknowledge support from PMMB fellowship.

References

[1] J. Moult, T. Hubbard, F. Fidelis, and J. Pedersen. Critical assessment of methods on protein structure prediction (CASP)-round III. *Proteins: Structure, Function and Genetics*, 37(S3):2–6, December 1999.

[2] J. Moult, F. Fidelis, A. Zemla, and T. Hubbard. Critical assessment of methods on protein structure prediction (CASP)-round IV. *Proteins: Structure, Function and Genetics*, 45(S5):2–7, December 2001.

[3] J. Moult, F. Fidelis, A. Zemla, and T. Hubbard. Critical assessment of methods on protein structure prediction (CASP)-round V. *Proteins: Structure, Function and Genetics*, 53(S6):334–339, October 2003.

[4] R.H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering*, 7:1059–1068, 1994.

[5] T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210:261–275, 1999.

[6] N.A. Pierce and E. Winfree. Protein design is NP-hard. *Protein Engineering*, 15(10):779–782, 2002.

Table 2. Top templates chosen by both linear programming approach and tree-decomposition approach for the thirty CASP6 test proteins.

test protein	t0200	t0201	t0202	t0203	t0204	t0205	t0206	t0207	t0208	t0209
top template	1hp1a	1unnc	1ko7a	1rxxa	1guqa	1h0xa	1mv3a	1h75a	1k77a	1i78a
test protein	t0210	t0211	t0212	t0213	t0214	t0215	t0216	t0217	t0218	t0219
top template	1j58a	1gvna	1fw9a	1ukfa	1nwaa	1vjqa	1iruf	1i60a	1hz4a	1h2ka
test protein	t0220	t0221	t0222	t0223	t0224	t0225	t0226	t0227	t0228	t0229
top template	1nvta	1dnya	1o60a	1nox	1jx7a	1n7ha	1c7qa	1mq0a	1qapa	1ml8a

Table 3. Tree width distribution of different tree-decomposition methods on the 178 test proteins used by SCWRL.

tree decomposition method	percentage of residue interaction graphs with a given tree width									average tree width
	≤ 2	3	4	5	6	7	8	9	≥ 10	
minimum vertex separator	34.27	20.22	7.87	11.24	5.056	5.618	3.933	2.809	8.989	5.208
two-way-1/2-triangle	34.27	30.34	8.43	12.92	6.180	4.494	0	1.124	2.247	3.927
two-way-2/3-triangle	34.27	23.60	20.79	10.67	5.056	1.685	2.247	0.5618	1.124	3.848
minimum-degree	34.27	39.89	20.79	2.247	2.247	0.5618	0	0	0	3.315
minimum-width	34.27	32.58	17.42	7.865	3.933	1.685	0	1.124	1.124	3.663
minimum-discrepancy	34.27	39.89	20.22	2.809	2.809	0	0	0	0	3.315

[7] T. Akutsu. NP-hardness results for protein side-chain packing. In S. Miyano and T. Takagi, editors, *Genome Informatics 8*, pages 180–186. 1997.

[8] R.H. Lathrop and T.F. Smith. A branch-and-bound algorithm for optimal protein threading with pairwise(contact potential) amino acid interactions. IEEE Computer Soc. Press, 1994.

[9] D.T. Jones. GenTHREADER: An efficient and reliable protein fold recognition method for genomic sequences. *Journal of Molecular Biology*, 287:797–815, 1999.

[10] Y. Xu, D. Xu, and E.C. Uberbacher. An efficient computational method for globally optimal threadings. *Journal of Computational Biology*, 5(3):597–614, 1998.

[11] C. L. Kingsford, B. Chazelle, and M. Singh. Solving and analyzing side-chain positioning problems using linear and integer programming. *Bioinformatics*, 2004.

[12] O. Eriksson, Y. Zhou, and A. Elofsson. Side chain-positioning as an integer programming problem. In *Proceedings of the First International Workshop on Algorithms in Bioinformatics*, pages 128–141. Springer-Verlag, 2001.

[13] S. Liang and N.V. Grishin. side-chain modelling with an optimized scoring function. *Protein Science*, 11:322–331, 2002.

[14] B. Chazelle, C. Kingsford, and M. Singh. A semidefinite programming approach to side-chain positioning with new rounding strategies. *Informs Journal on Computing, Special Issue in Computational Molecular Biology/Bioinformatics*, pages 86–94, 2004.

[15] K.C. Dukka, E. Tomita, J. Suzuki, and T. Akutsu. Protein side-chain packing problem: a maximum common edge-weight clique algorithmic approach. *Journal of Bioinformatics and Computational Biology*, 3:103–126, 2005.

[16] J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. Protein threading by linear programming. In *Biocomputing: Proceedings of the 2003 Pacific Symposium*, pages 264–275. Hawaii, USA, 2003.

[17] J. Xu, M. Li, D. Kim, and Y. Xu. RAPTOR: optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1):95–117, 2003.

[18] J. Xu. Rapid side-chain packing via tree decomposition. In *Proceedings of the Ninth Annual International Conference on Research in Computational Molecular Biology (RECOMB 2005)*. 2005.

Table 4. Average computational time of the tree-decomposition based approach and the linear programming approach to the side-chain prediction problem. (unit: second)

tree width	linear programming	tree-decomposition
≤ 2	0.941	0.888
3	1.723	1.644
4	2.519	2.423
5	5.118	4.202
6	3.252	3.380
≥ 7	4.540	4.440

[19] N. Robertson and P.D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[20] A.M.C.A. Koster, S.P.M. van Hoesel, and A.W.J. Kolen. Solving frequency assignment problems via tree-decomposition. Research Memoranda 036, Maastricht : METEOR, Maastricht Research School of Economics of Technology and Organization, 1999. available at http://ideas.repec.org/p/dgr/umamet/1999036.html.

[21] F. Bach and M. Jordan. Thin junction trees. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 14. 2002.

[22] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embedding in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[23] G. L. Miller, S. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of ACM*, 44(1):1–29, 1997.

[24] Eyal. Amir and Sheila. McIlraith. Partition-based logical reasoning. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[25] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. In *Lecture Notes in Computer Science, Vol. 2880*, pages 58–70. Springer Verlag, June 2003.

[26] E. Lindahl and A. Elofsson. Identification of related proteins on family, superfamily and fold level. *Journal of Molecular Biology*, 295:613–625, 2000.

[27] A.A. Canutescu, A.A. Shelenkov, and R.L. Dunbrack Jr. A graph-theory algorithm for rapid protein side-chain prediction. *Protein Science*, 12:2001–2014, 2003.