Rapid Protein Side-Chain Packing via Tree Decomposition

Jinbo Xu[1,2]

[1] School of Computer Science, University of Waterloo,
Waterloo, Ontario N2L 3G1, Canada
[2] Department of Mathematics, MIT, Cambridge, MA 02139.
j3xu@theory.csail.mit.edu

Abstract. This paper proposes a novel tree decomposition based side-chain assignment algorithm, which can obtain the globally optimal solution of the side-chain packing problem very efficiently. Theoretically, the computational complexity of this algorithm is $O((N+M)n_{rot}^{tw+1})$ where N is the number of residues in the protein, M the number of interacting residue pairs, n_{rot} the average number of rotamers for each residue and $tw(= O(N^{\frac{2}{3}} \log N))$ the tree width of the residue interaction graph. Based on this algorithm, we have developed a side-chain prediction program SCATD (Side Chain Assignment via Tree Decomposition). Experimental results show that after the Goldstein DEE is conducted, n_{rot} is around 3.5, tw is only 3 or 4 for most of the test proteins in the SCWRL benchmark and less than 10 for all the test proteins. SCATD runs up to 90 times faster than SCWRL 3.0 on some large proteins in the SCWRL benchmark and achieves an average of five times faster speed on all the test proteins. If only the post-DEE stage is taken into consideration, then our tree-decomposition based energy minimization algorithm is more than 200 times faster than that in SCWRL 3.0 on some large proteins. SCATD is freely available for academic research upon request.

1 Introduction

The structure of a protein plays an instrumental role in determining its functions. Experimental methods such as X-ray crystallography and NMR techniques cannot generate protein structures in a high throughput way. Protein structure prediction tools have been frequently used by structural biologists and pharmaceutical companies to analyze the structure features and function characteristics of a protein. Typically, in order to overcome the computational challenge, protein structure prediction is decomposed into two major steps. One is the prediction of the backbone atom coordinates and the other is the prediction of side-chain atom coordinates. The former step is usually done using protein threading programs [1–10] or sequence-based homology search tools such as PDB-BLAST [11]. The task of side-chain prediction is to determine the position of all the side-chain atoms given that the backbone coordinates of a protein are already known. Along with the advancement of protein backbone prediction techniques, side-chain prediction is becoming more important since the backbone coordinates of a protein can be predicted accurately.

Many side-chain prediction methods have been proposed and implemented in the past two decades [12–27]. Almost all of them use a rotamer library, which is a set of side-chain conformation candidates. In order to overcome computational difficulty, the side-chain conformation of a residue is discretized into a finite number of states (rotamers). Each rotamer is a representative of a set of similar side-chain conformations. A backbone-independent rotamer only depends on the type of the residue, while a backbone-dependent rotamer also depends on two dihedral angles (ϕ and ψ) associated with the residue. For example, Dunbrack *et al.* first developed a backbone-independent rotamer library and then a backbone-dependent rotamer library [28].

Given a rotamer library, the side-chain prediction problem can be formulated as a combinatorial search problem. The quality of a side-chain packing can be measured by an energy function, which usually consists of singleton scores and pairwise scores. Singleton score describes the preference of one rotamer for a particular residue and its environment. Singleton score can also describe the interaction between the rotamer atoms and the backbone atoms. Since the position of the backbone atoms are already fixed, this kind of interaction only depends on one movable side-chain atom. Pairwise score measures the interaction between two side-chain atoms. Pairwise score usually is used to avoid inter-atom clashes. A good side-chain packing should avoid as many clashes as possible. In addition, other atom-atom interactions can also be incorporated into the energy function. It is the side-chain atom interaction that makes the side-chain packing problem computationally challenging. The underlying reason is that in order to minimize the conflict, we have to fix the positions of all the interacting side-chain atoms simultaneously.

The side-chain prediction problem has been proved to be *NP*-hard [29, 30], which justifies the development of many heuristic algorithms such as SCAP [19] and MODELLER [31], and some approximate algorithms [23, 24]. These algorithms are usually computationally efficient, but cannot guarantee to find the side-chain assignment with the lowest system energy. The exact algorithms such as DEE/A^* [14, 32] and integer programming approach [25, 27] can find a globally minimum energy for a given rotamer library and an energy function. However, not many algorithms belong to this category due to the expensive computational time. SCWRL 3.0 [17] guarantees to find a globally optimal solution and also runs very fast for many proteins. SCWRL 3.0 first decomposes the residue interaction graph into some small biconnected components. Any two biconnected components share at most one articulation residue. If the side-chain assignment to the articulation residue is fixed, then the side-chain positioning in one component is independent of the other. As such, SCWRL 3.0 can optimize the side-chain assignment of these components one by one. A big optimization problem is decomposed into some small subproblems. According to their report, most of the biconnected components are quite small, containing no more than 21 residues. However, it still takes a long time to optimize the side-chain assignment to a component of 21 residues even if each residue has only 3.5 possible rotamers, which is the average number in our experiments.

In this paper, we present a novel tree decomposition based approach to the globally optimal side-chain packing problem. The biconnected decomposition of a graph used in SCWRL 3.0 can be considered as only a special case of the tree decomposition. The key point is that we optimize the tree decomposition of a graph such that the resultant components are as small as possible. Theoretically, we can have a polynomial-time algorithm to decompose the residue interaction graph into some components with size $O(N^{\frac{2}{3}} \log N)$ where N is the number of residues, which leads to a globally exact side-chain assignment algorithm with a computational complexity $O(N n_{rot}^{cN^{\frac{2}{3}} \log N})$ where c is a constant. As far as we know, this is the first globally optimal side-chain packing algorithm with a non-trivial time complexity. In contrast, the biconnected decomposition of a graph can easily result in a large component with size $O(N)$ even for some sparse graphs such as a cycle. Experimental results show that a typical residue interaction graph can be decomposed into components of 4 or 5 residues using our decomposition method. Since each residue has no more than 4 candidate rotamers after the Goldstein criterion DEE [33] is conducted, the optimal side-chain assignment of each component can be done very quickly.

The remainder of this paper is organized as follows. In Section 2, we describe the side-chain assignment problem further and formulate it as a combinatorial optimization problem. Section 3 introduces the concepts of tree decomposition and presents the tree decomposition based side-chain assignment algorithm. In this section, we also describe a low-degree polynomial-time algorithm that can decompose a residue interaction graph into some components of size $O(N^{2/3} \log N)$, and a simple heuristic tree decomposition algorithm, which works very well for our purpose. Section 4 describes a graph reduction technique that can be used to prune those residues interacting with only one or two other residues. This pruning strategy can improve the computational efficiency a little bit. In Section 5, we present the experimental results of our algorithm in detail and compare our side-chain prediction program with SCWRL 3.0 in terms of computational efficiency and accuracy. Finally, Section 6 points out that there are some polynomial-time approximation schemes for the side-chain packing problem, and discusses further development of our side-chain packing system.

2 Problem Description

The side-chain prediction problem can be formulated as follows. We use a *residue interaction graph* $G = (V, E)$ to represent the residues in a protein and their relationship. Each residue is represented by a vertex in the set V. For a residue i, we use $D[i]$ denote the set of possible rotamers for this residue. There is an *interaction edge* $(i, j) \in E$ between two residues i and j if and only if there are two rotamers $l \in D[i]$ and $k \in D[j]$ such that at least one atom in rotamer l conflicts with at least one atom in rotamer k. Two atoms conflict with each other if and only if their distance is less than the sum of their radii. We say that residues i and j interact with each other if there is one edge between i and j in G. For each rotamer $l \in D[i]$, there is an associated singleton score, denoted by $S_i(l)$. In our energy function, $S_i(l)$ is the interaction energy between rotamer l and the backbone of the protein. $S_i(l)$ also includes the preference of assigning one rotamer to a specific residue. For any two rotamers $l \in D[i]$ and $k \in D[j]$ $(i \neq j)$, there is also an associated pairwise score, denoted by $P_{i,j}(l,k)$, if residue i interacts with residue j. Let $E(a,b)$ denote the interaction score between two atoms a and b. We use the method in the SCWRL 3.0 paper [17] to calculate $E(a,b)$ as follows .

$$\begin{aligned} E(a,b) &= 0 && r \geq R_{a,b} \\ &= 10 && r \leq 0.8254 R_{a,b} \\ &= 57.273\left(1 - \tfrac{r}{R_{a,b}}\right) && \text{otherwise} \end{aligned}$$

where r is the distance between atoms a and b and $R_{a,b}$ is the sum of their radii. Let $SC(i)$ and $BB(i)$ denote the set of side-chain atoms and the set of backbone atoms of one residue i, respectively, and $Pr_i(l|\phi,\psi)$ denote the probability of rotamer l given the residue i and two angles ϕ and ψ. Then we calculate $S_i(l)$ and $P_{i,j}(l,k)$ as follows [17].

$$S_i(l) = -K \log\left(\frac{Pr_i(l|\phi,\psi)}{\max_{l \in D[i]} Pr_i(l|\phi,\psi)}\right) + \sum_{|i-j|>1} \sum_{s \in SC(i)} \sum_{b \in BB(j)} E(s,b) \qquad (1)$$

$$P_{i,j}(l,k) = \sum_{a \in SC(i)} \sum_{b \in SC(j)} E(a,b) \qquad (2)$$

In Eq. 1, K is optimized to 6 to yield the best prediction accuracy. Please notice that in the above two equations, the position of one side-chain atom depends on its associated rotamer.

Given a side-chain assignment $A(i) \in D[i]$ to residue i $(i \in V)$, the quality of one side-chain packing is measured by the following energy function.

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

The smaller the system energy $E(G)$ is, the better the side-chain assignment. So our goal is to develop an efficient algorithm to search for the side-chain assignment such that the energy $E(G)$ in Eq. 3 is minimized.

3 Side-Chain Prediction Algorithm

In this section, we will first introduce the concept of tree decomposition of a graph, and then describe how to search for the optimal side-chain assignment based on the decomposition. We will also show that there is a low-degree polynomial-time algorithm to decompose the residue interaction graph into some components of size $O(|V|^{\frac{2}{3}} \log |V|)$. Finally, we describe an efficient heuristic algorithm to find a good tree decomposition of a graph.

3.1 Tree Decomposition Concepts

The notions of tree width and tree decomposition are introduced by Robertson and Seymour [34] in their work on graph minors. The tree decomposition of a sparse graph has been applied to many NP-hard problems such as frequency assignment problem [35] and Bayesian inference [36].

Definition 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.

According to the above definition, the decomposition of a graph into biconnected components is also a tree decomposition. Each biconnected component corresponds to a node in T and any two biconnected components share at most one articulation vertex in G. However, the width of a biconnected-component decomposition could be $O(|V|)$, which is much bigger than the tree width of a graph G if G is sparse. For example, when a graph is a cycle, this graph has only one biconnected component—itself. In contrast, the tree width of a cycle is only 2. Figure 1, 2 and 3 show an example of an interaction graph, its biconnected component decomposition with width 6 and a tree decomposition with width 3. The width of a tree decomposition is a key factor determining the computational complexity of all the tree decomposition based algorithms. The smaller the width of a tree decomposition is, the more efficient the algorithm. Therefore, we need to optimize the tree decomposition of the residue interaction graph such that we can have a very small tree width. In the next subsection, we will describe a tree decomposition based side-chain assignment algorithm and analyze its computational complexity.

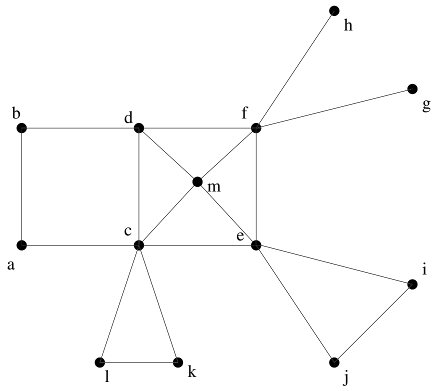

Fig. 1. Example of a residue interaction graph.

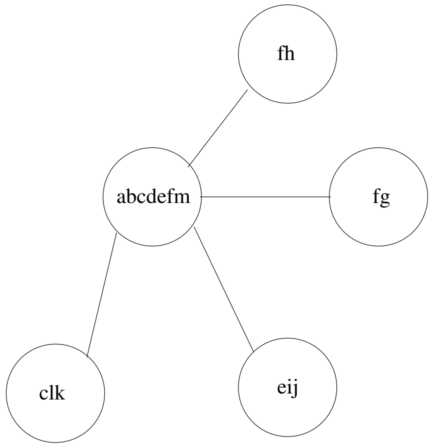

Fig. 2. Example of the biconnected-component decomposition of a graph. The width of this decomposition is 6.

3.2 Tree Decomposition Based Side Chain Assignment Algorithm

In this subsection, we describe an algorithm to search for the optimal side-chain assignment based on a tree decomposition (T, X) of a residue interaction graph G. For simplicity purpose, we assume

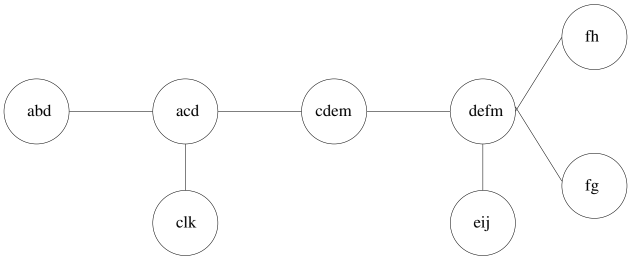

Fig. 3. Example of a tree decomposition of a graph with width 3.

that tree T has a root 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. Our tree decomposition based side-chain assignment algorithm consists of two steps. One is the calculation of the optimal energy in a bottom-to-top way and the other is the extraction of the optimal assignment in a top-to-bottom way.

Bottom-to-Top Suppose we start from a leaf node i in the tree T and node j is the parent of i. Let $X_{i,j}$ denote the intersection between X_i and X_j. Given a side-chain assignment $A(X_{i,j})$ to the residues in $X_{i,j}$, we enumerate all the possible side-chain assignments to the residues in $X_i - X_{i,j}$ and then find the best side-chain assignment such that the energy of the subgraph induced by X_i is minimized. We record this optimal energy as the multi-body score of $X_{i,j}$, which only depends on $A(X_{i,j})$. All the residues in $X_{i,j}$ form a hyper edge, which is added into the subgraph induced by X_j. When the energy of the subgraph induced by X_j is calculated, the multi-body score corresponding to this hyper edge should be included. In addition, we also save the optimal assignment to all the residues in $X_i - X_{i,j}$ for each $A(X_{i,j})$ since in the top-to-bottom step we need it for traceback. For example, in Figure 3, if we assume the node acd is the root, then node $defm$ is an internal node with parent $cdem$. For each side-chain assignment to residues d, e and m, we can find the best side-chain assignment to residue f such that the energy of the subgraph induced by d, e, f and m is minimized. Then we add one hyper edge (d, e, m) to node $cdem$. In this bottom-to-top process, a tree node can be calculated only after all of its child nodes are calculated. When calculating the root node of T, we enumerate the side-chain assignments to all the residues in it and obtain the optimal assignment such that the energy is minimized. This minimized energy is also the optimal menergy of the whole system.

Top-to-Bottom After calculating the root node of tree T, we have the optimal assignment to all the residues in the root. Now we trace back from the parent node to its child nodes to extract out the optimal assignment to all the residues in a child node. Assume that the optimal assignment to all the residues in node j are already known and node i is a child node of j. We can easily extract out the optimal assignment to all the residues in $X_i - X_{i,j}$ based on the assignment to the residues in $X_{i,j}$ since we have already saved this assignment in the bottom-to-top step. Recursively, we can track down to the leaf nodes of T to extract out the optimal assignment to all the residues in G. If we want to save the memory consumption, then we do not need to save the optimal assignments in the bottom-to-top step. Instead, in this step we can enumerate all the assignments to $X_i - X_{i,j}$ to obtain the optimal assignment to all the residues in $X_i - X_{i,j}$. The computational effort for this enumeration is much cheaper than that in the bottom-to-top step since there is only one side-chain assignment to $X_{i,j}$.

In addition, based on the definition of tree decomposition, one residue might occur in several tree nodes. To avoid incorporating the singleton score of this residue into the overall system energy more than once, we include the singleton score of this residue into the system only when we are calculating the tree node with the maximal height among all the nodes containing this residue. 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 1. *The tree decomposition based side-chain assignment algorithm has a computational complexity of $O((|V| + |E|)n_{rot}^{1+tw})$ where V is the set of residues in the system, E the set of interaction edges, n_{rot} the average number of rotamers for each residue, and tw the width of the tree decomposition. The space complexity of this algorithm is $O(|V|n_{rot}^{tw})$.*

According to the above lemma, the computational complexity of tree decomposition based side-chain assignment algorithm is exponential to the width of the tree decomposition. Therefore, we need an algorithm to decompose the interaction graph into very small components.

3.3 Construction of Tree Decomposition

The optimal tree decomposition of a general graph has been proved to be *NP*-hard [37], which means it is unlikely to find the optimal decomposition of a graph within polynomial time. However, since the residue interaction graph is a geometric graph, we can very quickly obtain a tree decomposition with width $O(|V|^{\frac{2}{3}} \log |V|)$, based on the following sphere separator theorem [38].

Theorem 1. *Given a residue interaction graph $G = (V, E)$, there is a separator subset U with size $O(|V|^{\frac{2}{3}})$ of V such that removal of U from the graph can partition V into two subsets V_1 and V_2, and the following conditions are satisfied: (1) there is no interaction edge between V_1 and V_2; (2) $|V_i| \leq \frac{4}{5}|V|$ for $i = 1, 2$. In addition, such a subset U can be computed by a deterministic algorithm in random linear time.*

Before presenting the proof of the above theorem, we first introduce the definition of *k-ply neighborhood system* and a *sphere separator* theorem [38].

Definition 2 (k-ply neighborhood system). *A k-ply neighborhood system in $\Re^3$ is a set $\{B_1, B_2, ..., B_n\}$ of closed balls in $\Re^3$ such that no point in $\Re^3$ is strictly interior to more than k of the balls.*

Theorem 2 (Sphere Separator Theorem). *For every k-ply neighborhood system $\{B_1, B_2, ...B_n\}$ in $\Re^3$, there is a sphere separator S such that:(1)$|N_E| \leq \frac{4}{5}n$ where N_E contains all the balls in the exterior of S; (2) $|N_I| \leq \frac{4}{5}n$ where N_I contains all the balls in the interior of S; (3) $|N_o| = O(k^{\frac{1}{3}}n^{\frac{2}{3}})$ where N_o contains all the balls that intersect S. In addition, such an S can be computed by a deterministic algorithm in random linear time.*

Based on the above theorem, we can prove Theorem 1 as follows.

Proof of Theorem 1 Since the distance between any side-chain atom and its associated C_α atom is bounded above by a constant, according to the definition of interaction edge in Section 2, there is a constant $d_u > 0$ such that if the distance between two residues is more than d_u, then there will be no interaction edge between these two residues no matter which rotamer is assigned to them. In this paper, we use the position of C_α atoms to calculate the distance between two residues. For each residue, we construct a ball with radius $d_u/2$ centered at its C_α atom. In a normal protein, the distance between any two residues should be no less than a constant d_l. Therefore, there is a constant k $(\leq (1+\frac{d_u}{d_l})^3)$ such that no point in $\Re^3$ is strictly interior to more than k balls. Based on Theorem 2, we can have a sphere S such that S intersects with only $O(|V|^{2/3})$ balls and all the other balls are located inside or outside S in a balanced way. Let U denote the set of all the residues with its ball intersecting with S. Then we have $|U| = O(|V|^{2/3})$ and U will partition graph G into two subgraphs with balanced size.

Based on Theorem 1, we can prove the following theorem.

Theorem 3. *There is a low-degree polynomial-time algorithm that can find a tree decomposition of the residue interaction graph with a tree width of* $O(|V|^{\frac{2}{3}} \log |V|)$

Proof. Based on Theorem 1, we can partition G into two subgraphs G_1 and G_2 by removing a separator subset of $O(|V|^{\frac{2}{3}})$ residues such that there is no interaction edge between G_1 and G_2 and $1/4 \leq |V(G_1)|/|V(G_2)| \leq 4$. Recursively, we can also partition G_1 and G_2 into smaller subgraphs until the size of the subgraph is $O(|V|^{2/3})$. Finally, we can have a binary partition tree in which each subtree corresponds to a subgraph, and the root of the subtree is the separator subset of the subgraph. Based on this binary partition tree, we can construct a tree decomposition of G as follows. For each partition tree node, we construct a decomposition component by assembling together all the residues along the path from the partition tree root to this node. We can easily verify that all the components form a tree decomposition of the graph. Since the height of the binary partition tree is $O(\log |V|)$, the tree width of this tree decomposition is $O(|V|^{\frac{2}{3}} \log |V|)$. Each partition step can be finished within linear time, so we can construct such a tree decomposition within low-degree polynomial-time.

Combining Theorem 3 and Lemma 1, we can calculate the computational complexity of the tree-decomposition based algorithm.

Theorem 4. *The tree decomposition based side-chain assignment algorithm has a computational complexity of* $O\left((|V|+|E|)\, n_{rot}^{O\left(|V|^{2/3}\log |V|\right)}\right)$ *where* V *is the set of residues in the system,* E *the set of interaction edges,* n_{rot} *the average number of rotamers for each residue.*

Although we give a low-degree polynomial-time algorithm to find a tree decomposition of G with a theoretically sound tree width, in practice we can use a simple heuristic algorithm to decompose the graph. Later in this paper we will show that the tree decompositions found by the heuristic algorithm are good enough. Many of them have a tree width of only 3 or 4, which leads to a very efficient algorithm for side-chain prediction. In our program, we use "minimum degree" heuristic [39] to recursively partition the graph. Specifically, we choose the vertex with the smallest number of neighbors. Then we add edges to the graph such that any two neighbors of the selected vertex is connected by an edge. Finally, we remove the selected vertex and its adjacent edges from the graph and recursively choose the next vertex. The selected vertex with its neighbors form a partition component of the graph. This algorithm runs very efficiently and also effectively for our purpose.

4 Graph Reduction Technique

After DEE is conducted, many residues interact with only one or two other residues since many rotamers are removed from the candidate lists. That is, the interaction graph contains many vertices with degree one or two, which results in many small components in the tree decomposition of this graph. Although it is extremely fast to compute the optimal energy of these small components, it will incur certain amount of overheads since we need to do many bookkeepings for the calculation of one component. We can use a graph reduction technique to remove these low-degree vertices before applying the tree decomposition to the interaction graph. The reduced system will have the same energy as the original system and can be easily recovered to the original system. For a residue i with degree one, assume i is only adjacent to residue j. Then, we can remove i by modifying the singleton score of j as follows.

$$S_j(k) \leftarrow S_j(k) + \min_{l \in D[i]} \{S_i(l) + P_{i,j}(l,k)\} \tag{4}$$

Since residue i only interacts with residue j, for each rotamer k of residue j, we can find the best rotamer for residue i and then remove residue i from the system. The optimal system energy will not change, and once the rotamer assignment to residue j is fixed, we can also easily obtain the optimal rotamer assignment to residue i.

For a residue with degree two, we can prune it by modifying the pairwise interaction scores of its two adjacent residues j_1, j_2 as follows.

$$P_{j_1,j_2}(k_1,k_2) \leftarrow P_{j_1,j_2}(k_1,k_2) + \min_{l \in D[i]} \{S_i(l) + P_{i,j_1}(l,k_1) + P_{i,j_2}(l,k_2)\} \tag{5}$$

Since residue i only interacts with two residues j_1 and j_2, for each combination of rotamer assignment to j_1 and j_2, we can find the best rotamer for i and then remove i from the system. The optimal system energy will not change, and once the rotamer assignment to residue j_1 and j_2 is fixed, we can also easily obtain the optimal rotamer assignment to residue i.

After applying the above-mentioned graph reduction technique to the interaction graph, the resultant new interaction graph looks more neat. The tree decomposition of new interaction graph will not generate any component of size one and few components of size two. Please notice that the graph reduction technique will not change the tree width of a graph. Therefore, this technique will not improve the theoretical computational complexity of the problem, but will improve the practical computational efficiency a little bit.

5 Experimental Results

We have implemented the idea presented in this paper as a program SCATD. In order to compare SCATD with SCWRL 3.0 [17], we test SCATD on the set of 180 proteins listed in the SCWRL 3.0 paper. The reason that we compare SCATD with SCWRL 3.0 is that both programs use the same rotamer library, similar energy function, same dead-end elimination method and, furthermore, solve the problem to its globally optimal solution.

Just like what is done in SCWRL 3.0 [17], for a particular residue and its associated two angles ψ and ϕ, we rank all the candidate rotamers from the highest probability to the lowest and remove the tail candidate if the probabilities of all the rotamers before it add up to 0.90 or above. Then we apply the Goldstein criterion dead-end elimination technique [33] to remove those rotamers that cannot be a part of the optimal side-chain assignment. Finally, we construct the residue interaction graph according to its definition described in Section 2. We build SCATD using BALL library [40] for some basic objects such as proteins, residues, and atoms, ANN library [41, 42] to determine if two atoms conflict or not, and split package [43] for tree decomposition of a graph.

5.1 Computational Efficiency

The residue interaction graphs are decomposed into small components with size no more than 10, much smaller than that reported in SCWRL 3.0 [17]. Most components have size only 4 or 5. Figure 4 shows the distribution of component sizes after the "minimum-degree" tree decomposition algorithm described in Section 3 is applied to the residue interaction graphs of the 180 proteins. As reported in the SCWRL3.0 paper, the maximum biconnected component size is 21 and there are quite a few of components with size larger than 10. As discussed in Section 3, the computing time of both SCATD and SCWRL 3.0 is exponential to the component size. Therefore, we can expect that our algorithm will be much more efficient than SCWRL 3.0. We also calculated the biconnected decomposition of all the interaction graphs generated by SCATD. Figure 5 illustrates the distribution of biconnected component sizes. As shown in Figure 5, the biconnected components of our interaction graphs have a bigger size than those in SCWRL 3.0 [17]. This indicates that we did not make the problem easier by using a slightly different energy function. Since the average number of rotamers for an active residue is 3.5, the algorithm used in SCWRL 3.0 cannot work very well on our interaction graphs if the biconnected component has size greater than 20.

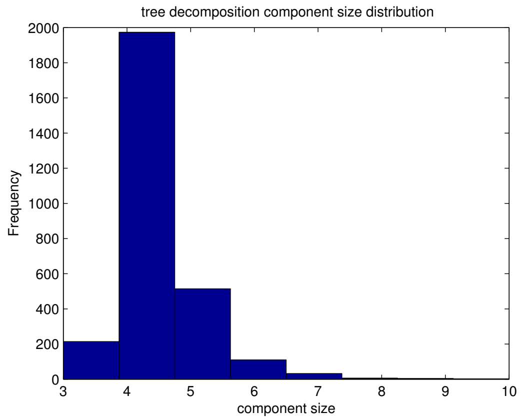

Fig. 4. The component size distribution of "minimum-degree" heuristic decomposition method. The components with size one or two are ignored.

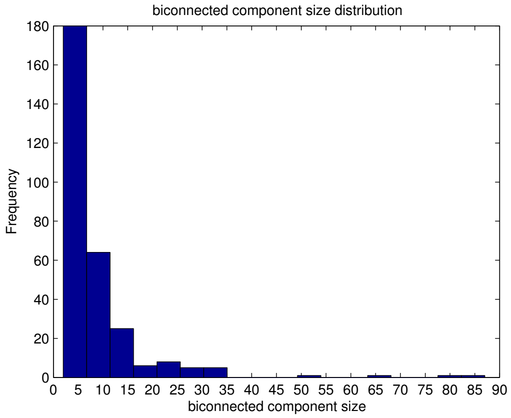

Fig. 5. The biconnected component size distribution of residue interaction graph. The components with size one or two are ignored.

We also ran SCATD and SCWRL 3.0 on a Debian Linux box with a 1.7GHz Pentium CPU. SCATD can do the side-chain prediction for all the 180 proteins within no more than 5 minutes while SCWRL 3.0 takes approximately 28 minutes. On average, SCATD is more than 5 times faster than SCWRL 3.0. Among these 180 test proteins, the maximum CPU time spent by SCATD on an individual protein is 8.68 seconds. We further compare SCATD with SCWRL 3.0 on some large proteins including 1gai, 1xwl, 1a8i, 1bu7 and 1b0p. The results are shown in Table 1. As shown in this table, SCATD is up to 90 times faster than SCWRL 3.0 for large proteins.

Table 1. Computational time spent by SCWRL 3.0 and SCATD on the side-chain prediction of several large proteins.

protein	# residues	SCWRL 3.0 (s)	SCATD (s)
1gai	472	266.62	2.98
1a8i	812	184.36	8.68
1b0p	2462	300.00	21.03
1xwl	580	26.51	4.64
1bu7	910	56.50	7.62

SCATD consists of the following major steps: loading rotamer library, converting rotamer angles to atom coordinates, calculating interaction scores between two atoms, dead-end elimination (Goldstein criterion), and energy minimization via tree decomposition. We test the CPU time spent on each step to examine which steps are the CPU bottleneck in SCATD. Table 2 lists the detailed CPU times spent on each step by SCATD. As shown in this table, the tree decomposition based side-chain assignment algorithm is not the bottleneck at all. The average CPU time spent on this step is only 1.5% of the total computational time. Since we do not have the source code of SCWRL 3.0, we have no way to exactly measure the time spent by SCWRL 3.0 on the post-DEE stage. By observing the running status of SCWRL 3.0 on 1a8i and 1b0p, we find that the CPU time spent by SCWRL 3.0 on the post-DEE stage is approximately 70% of the total computational time. Nevertheless, our tree decomposition based algorithm can minimize the energy of 1a8i within 0.4 seconds and 1b0p within 0.7 seconds after DEE is conducted. Therefore, for large proteins such as 1a8i and 1b0p, our tree decomposition based algorithm runs more than two hundred times faster than the biconnected decomposition based algorithm in SCWRL 3.0.

Table 2. The average CPU time for each step.

	library loading	coordinate calculation	interaction score calculation	DEE	tree decomposition based algorithm
time (s)	0.2822	0.2164	0.7924	0.3038	0.0239

5.2 Prediction Accuracy

While SCATD runs much faster than SCWRL 3.0, SCATD does not lose any accuracy. SCATD uses the same rotamer library as SCWRL 3.0 and a slightly different energy function. Table 3 lists the prediction accuracy of 18 types of amino acids. The prediction accuracy of both programs are very close. The minor difference comes from the fact that the atomic radii in the BALL library is slightly different from those in SCWRL 3.0. An interesting result is that SCATD does better in predicting CYS and worse in ASN and ASP than SCWRL 3.0.[3]

[3] We disable the “-u” option of SCWRL 3.0 in order to compare both programs fairly since we have not implemented disulfide bond detection in SCATD. The overall accuracy of SCWRL 3.0 does not improve if “-u” option is enabled.

Table 3. Prediction accuracy of SCATD and SCWRL 3.0 in the 180-protein test set. A prediction is judged as correct if its deviation from the experimental value is no more than 40 degree. For χ_{1+2} to be correct, both χ_1 and χ_2 must be correct.

	SCATD		SCWRL 3.0	
amino acid	χ_1 accuracy	χ_{1+2} accuracy	χ_1 accuracy	χ_{1+2} accuracy
ARG	0.7576	0.6135	0.7673	0.6381
ASN	0.7666	0.6479	0.7898	0.6749
ASP	0.7727	0.6668	0.8147	0.7129
CYS	0.7746	–	0.7052	–
GLN	0.7466	0.5086	0.7464	0.5290
GLU	0.7057	0.4922	0.7177	0.5223
HIS	0.8363	0.7711	0.8523	0.7877
ILE	0.9352	0.8376	0.9195	0.8095
LEU	0.9070	0.8279	0.9007	0.8203
LYS	0.7371	0.5607	0.7421	0.5773
MET	0.8183	0.6702	0.8016	0.6657
PHE	0.9306	0.8625	0.9354	0.8728
PRO	0.8511	0.7949	0.8449	0.7875
SER	0.6957	–	0.6730	–
THR	0.8871	–	0.8846	–
TRP	0.8786	0.6482	0.8828	0.6468
TYR	0.9085	0.8460	0.9212	0.8627
VAL	0.9212	–	0.9081	–
overall	0.8256	0.7329	0.8262	0.7374

6 Discussions

Based on the tree-decomposition of protein structures, we not only can give a fast, rigorous and accurate protein side-chain assignment method, but also can develop several polynomial-time approximation schemes (PTAS) to this problem. When an optimization problem admits a PTAS, it means that given an arbitrary error ϵ ($1 > \epsilon > 0$), there is a polynomial-time algorithm to approximate its objective function value within a factor of $(1 \pm \epsilon)$. In contrast, based on a general graph model, Chazelle *et al.* [23] proved that it is *NP*-complete to approximate this problem within a factor of $\Omega(N)$. Due to space limit, we only introduce the following three theorems without giving any proof, which will be presented in the extended version of this paper.

Theorem 5. *If every energy item in Eq. 3 is negative and the system energy should be minimized, then the side-chain packing problem admits a PTAS.*

Theorem 6. *Assume that all the pairwise energy items in Eq. 3 are positive and the system energy should be minimized. The side-chain packing problem admits a PTAS if the lowest system energy is* $\Omega(NP_{max}\sqrt{\frac{\log n_{rot}}{\log N}})$ *where* P_{max} *is the maximum among all* $P_{i,j}(A(i), A(j))$.

Theorem 7. *Assume that all the pairwise scores in Eq. 3 are negative and the system energy should be minimized. The side-chain packing problem admits a PTAS if the lowest system energy is no more than* $cNP_{min}\sqrt{\log n_{rot}/\log N}$ *where* c *is a positive constant and* P_{min} *the minimum among all* $P_{i,j}(A(i), A(j))$.

These theoretical results, especially Theorem 5, will stimulate us to develop a new energy function satisfying the conditions specified in these theorems and also having a good prediction accuracy so that we can apply these polynomial-time approximation algorithms to the problem. With a polynomial-time algorithm, we can deal with a larger rotamer library, which may result in a better prediction accuracy.

In protein structure prediction server RAPTOR [44, 45], we have developed a linear programming (LP) algorithm to obtain the globally optimal solution of the protein threading problem. The LP formulation used by RAPTOR can also be used to formulate the side-chain prediction problem. Mathematically, threading problem and side-chain prediction problem can be formulated in a very similar way. In fact, several research groups have proposed several more or less similar LP formulations for the side-chain packing problem [24, 25, 27]. We have also tested our LP formulation for the side-chain packing problem. In our setting, the tree decomposition based algorithm runs slightly faster than the LP approach. Interestingly, the tree decomposition algorithm proposed in this paper can also be used to the protein threading problem and contact map-based protein structure comparison.

The energy function used by our program is still very simple. In the future, we plan to add the disulfide bond detection into SCATD. We also plan to investigate more involved energy function to see how effective our algorithm is. For example, we can incorporate hydrogen bonds, electrostatics and solvation terms into our energy function. The major contribution of this paper is a novel and very efficient algorithm to the optimal protein side-chain packing problem but not a new energy function.

In this paper, we only test SCATD on the native backbone of the test proteins. The next step is to test SCATD on those backbones predicted by structure prediction programs. After all, a major usage of SCATD is to build the side-chain coordinates for a protein after its backbone coordinates are predicted.

7 Acknowledgments

The major work presented in this paper was done when the author is affiliated with Ming Li's group at the University of Waterloo. Some experimental analysis and the writing of this paper was done when the author is with Bonnie Berger's group at MIT. The author gratefully acknowledges support from NSERC CRC chair grant and PMMB fellowship.

References

1. B. Rost. TOPITS: Threading one-dimensional predictions into three-dimensional structures. In C. Rawlings, D. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors, *Third International Conference on Intelligent Systems for Molecular Biology*, pages 314–321, Cambridge, England, 1995. AAAI Press.
2. 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.
3. D. Kim, D. Xu, J. Guo, K. Ellrott, and Y. Xu. PROSPECT II: Protein structure prediction method for genome-scale applications. *Protein Engineering*, 16(9):641–650, 2003.
4. D.T. Jones. GenTHREADER: An efficient and reliable protein fold recognition method for genomic sequences. *Journal of Molecular Biology*, 287:797–815, 1999.
5. L.A. Kelley, R.M. MacCallum, and M.J.E. Sternberg. Enhanced genome annotation using structural profiles in the program 3D-PSSM. *Journal of Molecular Biology*, 299(2):499–520, 2000.
6. N.N. Alexandrov, R. Nussinov, and R.M. Zimmer. Fast protein fold recognition via sequence to structure alignment and contact capacity potentials. In *Biocomputing: Proceedings of 1996 Pacific Symposium*, 1996.
7. N. von Ohsen, I. Sommer, R. Zimmer, and T. Lengauer. Arby: automatic protein structure prediction using profile-profile alignment and confidence measures. *Bioinformatics*, 20(14):2228–35, Sep 2004.
8. J. Shi, L. B. Tom, and M. Kenji. FUGUE: Sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*, 310:243–257, 2001.

9. R.H. Lathrop and T.F. Smith. A branch-and-bound algorithm for optimal protein threading with pairwise (contact potential) amino acid interactions. In *Proceedings of the 27th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, 1994.
10. T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210:261–275, 1999.
11. W. Li, F. Pio, K. Pawlowski, and A. Godzik. Saturated BLAST: detecting distant homology using automated multiple intermediate sequence BLAST search. *Bioinformatics*, 16:1105–1110, 2000.
12. N.L. Summers and M. Karplus. Construction of side-chains in homology modelling: Application to the c-terminal lobe of rhizopuspepsin. *Journal of Molecular Biology*, 210:785–811, 1989.
13. L. Holm and C. Sander. Database algorithm for generating protein backbone and sidechain coordinates from a C_α trace: Application to model building and detection of coordinate errors. *Journal of Molecular Biology*, 218:183–194, 1991.
14. J. Desmet, M. De Maeyer, B. Hazes, and I. Laster. The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, 356:539–542, 1992.
15. J. Desmet, J. Spriet, and I. Laster. Fast and accurate side-chain topology and energy refinement (faster) as a new method for protein structure optimization. *Protein: Structure, Function and Genetics*, 48:31–43, 2002.
16. R.L. Dunbrack Jr. Comparative modeling of CASP3 targets using PSI-BLAST and SCWRL. *Protein: Structure, Function and Genetics*, 3:81–87, 1999.
17. 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.
18. R. Samudrala and J. Moult. Determinants of side chain conformational preferences in protein structures. *Protein Engineering*, 11:991–997, 1998.
19. Z. Xiang and B. Honig. Extending the accuracy limits of prediction for side-chain conformations. *Journal of Molecular Biology*, 311:421–430, 2001.
20. M.J. Bower, F.E. Cohen, and R.L. Dunbrack Jr. Prediction of protein side-chain rotamers from a backbone-dependent rotamer library: A new homology modeling tool. *Journal of Molecular Biology*, 267:1268–1282, 1997.
21. S. Liang and N.V. Grishin. side-chain modelling with an optimized scoring function. *Protein Science*, 11:322–331, 2002.
22. E.J. Hong and T. Lozano-Perez. Protein side-chain placement: probabilistic inference and integer programming methods. Technical report, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Jan 2004.
23. 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.
24. C. L. Kingsford, B. Chazelle, and M. Singh. Solving and analyzing side-chain positioning problems using linear and integer programming. *Bioinformatics*, 2004.
25. 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.
26. K.C. Dukka, E. Tomita, J. Suzuki, and T. Akutsu. Protein side-chain packing problem: a maximum common edge-weight clique algorithmic approach. In *The Second Asia Pacific Bioinformatics Conference*, 2004.
27. E. Althaus, O. Kohlbacher, H.P. Lenhof, and P. M*ü*ller. A branch and cut algorithm for the optimal solution of the side-chain placement problem. Technical Report MPI-I-2000-1-001, Max-Planck-Institute f*ü*r Informatik, 2000.
28. R.L. Dunbrack Jr. and M. Karplus. Backbone-dependent rotamer library for proteins: Application to side-chain prediction. *Journal of Molecular Biology*, 230:543–574, 1993.
29. N.A. Pierce and E. Winfree. Protein design is NP-hard. *Protein Engineering*, 15(10):779–782, 2002.
30. T. Akutsu. NP-hardness results for protein side-chain packing. In S. Miyano and T. Takagi, editors, *Genome Informatics 8*, pages 180–186, 1997.
31. A. Sali and T.L. Blundell. Comparative protein modelling by satisfaction of spatial restraints. *Journal of Molecular Biology*, pages 779–815, 1993.
32. A.R. Leach and A.P. Lemon. Exploring the conformational space of protein side chains using dead-end elimination and the A^* algorithm. *Protein: Structure, Function and Genetics*, 33:227–239, 1998.
33. R.F. Goldstein. Efficient rotamer elimination applied to protein side-chains and related spin glasses. *Biophysical Journal*, 66:1335–1340, 1994.
34. N. Robertson and P.D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.
35. 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.
36. 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.
37. 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.
38. 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.

39. A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. volume 2880, pages 58–70. Proceedings WG 2003 - 29th Workshop on Graph Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer Verlag, June 2003.
40. O. Kohlbacher and H.P. Lenhof. BALL - rapid software prototyping in computational molecular biology. *Bioinformatics*, 16(9):815C824, 2000.
41. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of ACM*, 45:891–923, 1998.
42. D.M. Mount and S. Arya. ANN: a library for approximate nearest neighbor searching. In *2nd CGC Workshop on Computational Geometry*, 1997.
43. E. Amir. Efficient approximation for triangulation of minimum treewdith. In *17th Conference on Uncertainty in Artificial Intelligence (UAI '01)*, 2001.
44. 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.
45. 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.