Fast and Accurate Algorithms for Protein Side-Chain Packing

JINBO XU

Toyota Technological Institute at Chicago and Massachusetts Institute of Technology

AND

BONNIE BERGER

Massachusetts Institute of Technology

Abstract. This article studies the protein side-chain packing problem using the tree-decomposition of a protein structure. To obtain fast and accurate protein side-chain packing, protein structures are modeled using a geometric neighborhood graph, which can be easily decomposed into smaller blocks. Therefore, the side-chain assignment of the whole protein can be assembled from the assignment of the small blocks. Although we will show that the side-chain packing problem is still *NP*-hard, we can achieve a tree-decomposition-based globally optimal algorithm with time complexity of $O(Nn_{rot}^{tw+1})$ and several polynomial-time approximation schemes (PTAS), where N is the number of residues contained in the protein, n_{rot} the average number of rotamers for each residue, and $tw = O(N^{2/3} \log N)$ the treewidth of the protein structure graph. Experimental results indicate that after Goldstein dead-end elimination is conducted, n_{rot} is very small and tw is equal to 3 or 4 most of the time. Based on the globally optimal algorithm, we developed a protein side-chain assignment program TreePack, which runs up to 90 times faster than SCWRL 3.0, a widely-used side-chain packing program, on some large test proteins in the SCWRL benchmark database and an average of five times faster on all the test proteins in this database. There are also some real-world instances that TreePack can solve but that SCWRL 3.0 cannot. The TreePack program is available at http://ttic.uchicago.edu/~jinbo/TreePack.htm.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*; G.2.3 [**Discrete Mathematics**]: Applications; J.3 [**Life and Medical Sciences**]: Biology and Genetics

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Protein side-chain packing, tree decomposition, approximate algorithm, polynomial-time approximation scheme

---

The work on this article was partially conducted when J. Xu was with Ming Li's group at the School of Computer Science, University of Waterloo.

This work was also supported by PMMB fellowship when J. Xu was with Bonnie Berger's group.

Authors' addresses: J. Xu, Toyota Technological Institute at Chicago, 1427 East 60th Street, Chicago, IL, 60637, e-mail: j3xu@tti-c.org; B. Berger, Department of Mathematics 2-373, Computer Science and Artificial Intelligence Laboratory 32-G594, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, e-mail: bab@mit.edu.

1. *Introduction*

The three-dimensional structure of a protein plays an instrumental role in determining its function. However, existing experimental methods such as X-ray crystallography and NMR techniques cannot generate protein structures in a high throughput way. Along with more available experimental protein structures and the advancement of protein structure prediction tools, computational approaches to protein structure prediction have become useful and successful, as demonstrated in recent CASP competitions [Moult et al. 1999, 2001, 2003]. 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. 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 [Y. Xu et al. 1998; J. Xu et al. 2003a, 2003b; Jones 1999; Kelley et al. 2000; Alexandrov et al. 1996; von Ohsen et al. 2004; Shi et al. 2001] or homology modeling tools such as PDB-BLAST [Li et al. 2000].

In this article, we focus on developing a fast and accurate algorithm for the second task, side-chain prediction. Here, the goal 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 backbone prediction techniques and thus the accurate prediction of backbone coordinates [Moult et al. 1999, 2001, 2003], side-chain packing is becoming more important. Many side-chain prediction methods have been proposed and implemented in the past two decades [Summers and Karplus 1989; Lee and Subbiah 1991; Holm and Sander 1991; Desmet et al. 1992, 2002; Dunbrack 1999; Canutescu et al. 2003; Samudrala and Moult 1998; Xiang and Honig 2001; Bower et al. 1997; Liang and Grishin 2002; Chazelle et al. 2004; Kingsford et al. 2005; Eriksson et al. 2001; Dukka et al. 2004]. Almost all of them use a rotamer library, which is a set of side-chain conformation candidates. In order to overcome computational complexity, the side-chain conformation space 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. Side-chain packing is to choose one and only one rotamer for each residue according to some criteria. Rotamers can be backbone-independent or backbone-dependent. A backbone-independent rotamer only depends on the type of the residue, while a backbone-dependent rotamer also depends on the local structure features (e.g., two dihedral angles ϕ and ψ) of the residue.

Given a rotamer library, the side-chain prediction problem can be formulated as a combinatorial search problem. The quality of side-chain packing can be measured by an energy function, which usually consists of two types of score items. One type of score item, called a *singleton score*, describes the preference of one rotamer for a particular residue and its environment. The 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 depends only on one movable side-chain atom. The other type of score item measures the interaction between two side-chain atoms. This type of score is called a pairwise score since it depends on two movable side-chain atoms. The pairwise score usually is used to avoid clashes between two side-chain atoms. A good side-chain packing should

avoid as many clashes as possible. Of course, we can also incorporate other atom-atom interactions 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 [Pierce and Winfree 2002; Akutsu 1997] and also hard to approximate [Chazelle et al. 2004], which justifies the development of many heuristic-based algorithms and approximation algorithms [Chazelle et al. 2004]. These heuristic-based algorithms are usually computationally efficient but have no guarantee of performance. There are also some globally exact algorithms such as DEE/A^* [Desmet et al. 1992; Leach and Lemon 1998] and integer programming approach [Eriksson et al. 2001] for this problem. These algorithms find a globally minimum energy but usually cannot be applied to large-scale instances. Theoretically, these globally exact algorithms have a time complexity of $O(Nn_{rot}^N)$ where N is the length of the protein sequence and n_{rot} is the average number of candidate rotamers for each residue. The difficulty of the side-chain packing problem comes from the fact that a protein structure is usually modeled by a very general graph, which does not capture all the geometric characteristics of a protein.

In this article, we present a significantly fast and accurate solution to the protein side-chain packing problem by representing the rotamer conflict relationship using a residue interaction graph, which turns out to be a geometric neighborhood graph. A geometric neighborhood graph describes the relationship among a set of geometric objects. Each geometric object is treated as a vertex and there is one edge between two geometric objects if and only if they are very close. Based on the geometric neighborhood graph, we propose a tree-decomposition based globally optimal algorithm and three polynomial-time approximation schemes for the side-chain packing problem. The tree-decomposition-based globally exact algorithm has a theoretical time complexity of $O(Nn_{rot}^{O(N^{2/3}\log N)})$, which is much better than $O(Nn_{rot}^N)$. Moreover, we demonstrate that in practice the tree decomposition based exact algorithm can achieve a time complexity of $O(Nn_{rot}^4)$ most of the time (see Section 5).

The TreePack (Tree-Decomposition Based Side-Chain Packing) program, which implements the tree-decomposition based algorithm presented here, is much more efficient than SCWRL 3.0 [Canutescu et al. 2003], a widely used side-chain packing program, while maintaining a similar accuracy level. Tested on the 180 proteins in the SCWRL benchmark, TreePack achieves an average runtime of five times faster than SCWRL 3.0 on all the proteins and up to 90 times faster than SCWRL 3.0 on the large proteins. Furthermore, we demonstrate that TreePack can solve some even more difficult instances that SCWRL 3.0 cannot work out at all. These instances are derived from a real-world protein modeling endeavor (Julio Kovacs, personal communication).

2. *Preliminaries*

2.1. SIDE-CHAIN PACKING PROBLEM. 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 the conflict relationship of their rotamers.

Each residue is represented by a vertex in the vertex set V. Each vertex is also associated with the 3D coordinates of its corresponding residue center. Let $D[i]$ denote the set of possible rotamers for residue i. 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. Since the distance between any rotamer atom to its associated residue center is bounded above by a constant, there is a constant D_u such that if the distance between two residues is bigger than D_u, then there is no interaction edge between these two residues. In a normal protein, any two residues cannot be arbitrarily close, which is one of the underlying reasons why lattice models can be used to approximate protein folding. We let a constant D_l ($D_l > 0$) denote the minimum distance between any two residues in a protein. According to simple statistics on the PDB database, 99% of inter-residue distances are beyond $3.5\mathring{A}$. These two observations indicate that the residue interaction graph is sparse. In particular, it can be verified that each residue can only interact with at most $\Delta \leq (1 + \frac{D_u}{D_l})^3$ residues. This fact enables us to use tree-decomposition technique to cut the residue interaction graph into some very small components so that we can do side-chain assignment to each small component almost independently and quickly.

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. $P_{i,j}(l, k)$ is the interaction score between residues i and j when their side-chain conformations are rotamers l and k, respectively.

Let $E(a, b)$ denote the interaction score between two atoms a and b. We use the method in the SCWRL 3.0 article [Canutescu et al. 2003] to calculate $E(a, b)$ as follows.

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

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 [Canutescu et al. 2003]:

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

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

In Eq. (1), K is optimized to 8 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 this side-chain packing is measured by the following energy function.

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

The smaller the system energy $E(G)$ is, the better the side-chain assignment. To solve the side-chain packing problem, we need to develop an efficient algorithm to search for the best rotamer assignment such that the energy $E(G)$ in Eq. (3) is minimized.

Notice that we assume that there is an interaction score between two rotamers only if they conflict with each other. In fact, we can relax this "conflict" condition to various kinds of practical energy functions. In practice, since the atomic force decreases dramatically with respect to distance, no matter what kind of energy function is used, it is reasonable to assume that there is a distance cutoff such that if the distance between two atoms is beyond this cutoff, then there is no interaction force between them. Therefore, no matter what energy function is used, the geometric characteristics of the residue interaction graph will not change. That is, any residue can only interact with all the residues which are at most distance D_u away and the distance between any two residues is at least D_l. However, the value of D_u will vary with respect to the energy function to be used.

2.2. Tree Decomposition of a Graph. The notions of tree width and tree decomposition were introduced by Robertson and Seymour [1986] in their work on graph minors. The tree decomposition of a sparse graph has been applied to many NP-hard problems such as the frequency assignment problem and Bayesian inference [Bach and Jordan 2002].

Definition 2.1. 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.

Figures 1 and 2 show an example of an interaction graph and a tree decomposition with width 3. The width of a tree decomposition is a key factor in determining the computational complexity of the tree decomposition based side-chain assignment algorithm. The smaller the width of a tree decomposition, the more efficient the tree decomposition based side-chain assignment algorithm. Therefore, we need to optimize the tree decomposition of the residue interaction graph so that we can have a very small tree width. In the next section, we will describe our tree decomposition based side-chain assignment algorithm and analyze its computational complexity.

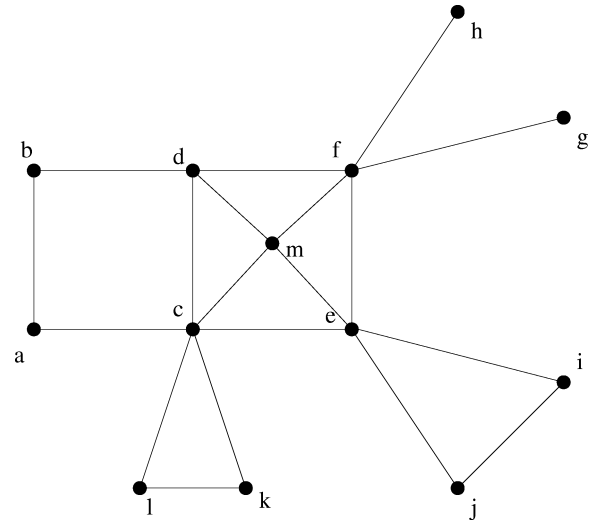

FIG. 1. Example of a residue interaction graph.

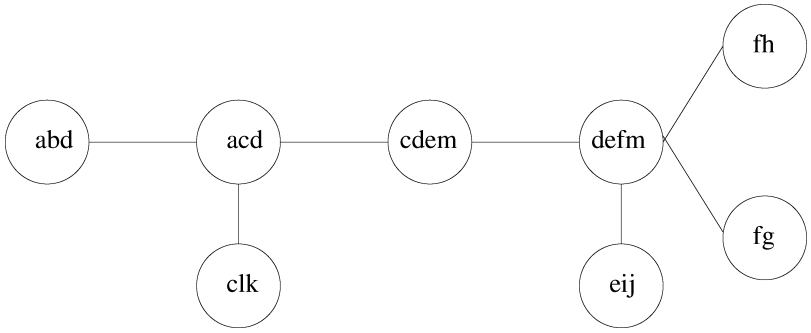

FIG. 2. Example of a tree decomposition of a graph with width 3.

3. *Treepack: Exact Side-Chain Prediction Algorithm*

3.1. FAST TREE DECOMPOSITION BASED SIDE-CHAIN ASSIGNMENT ALGORITHM. Here we assume that we have a tree decomposition (T, X) of a residue interaction graph G. For simplicity, we assume that tree 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 all the vertices in $X_{r,j}$ are removed, then it can be easily verified that this tree decomposition becomes two disconnected subtrees. Let $F(X_j, A(X_{r,j}))$ denote the optimal side-chain assignment score of the subtree rooted at X_j given that the side-chain assignment for $X_{r,j}$ 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 side-chain assignment being $A(X_j)$. Let $D[X]$ denote all the possible side-chain assignments to the vertices in X. Therefore, we have the

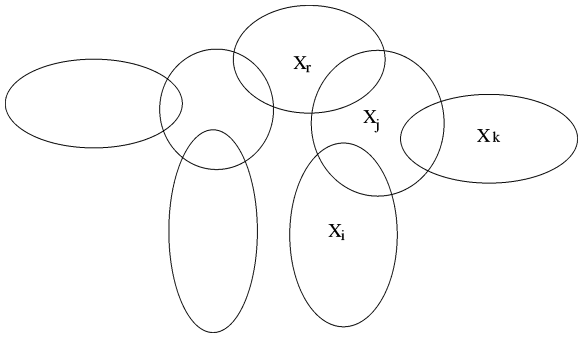

FIG. 3. A tree decomposition (T, X) of G. X_r is the root component. X_j has two child components X_i and X_k.

following recursive equation.

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

Based on the above equation, we can calculate the optimal side-chain packing in two steps. First, we calculate the optimal energy function (that is, the optimal side-chain assignment score) from bottom to top and then we extract the optimal side-chain assignment from top to bottom.

Bottom-to-Top Side Chain Assignment. 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 side-chain assignments to the residues in $X_{j,i}$. Given a side-chain assignment $A(X_{j,i}) \in D[X_{j,i}]$ to the residues in $X_{j,i}$, we enumerate all the possible side-chain assignments to the residues in $X_i - X_{j,i}$ 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 $X_{j,i}$, which depends on $A(X_{j,i})$. All the residues in $X_{j,i}$ form a hyper edge, which is added into the subgraph induced by X_j. When we calculate the energy of the subgraph induced by X_j, we need to incorporate the multi-body score corresponding to this hyper edge. In addition, we also save the optimal assignment for all the residues in $X_i - X_{j,i}$ for each side-chain assignment to $X_{j,i}$ 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 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 we calculate the root node of T, we enumerate the side-chain assignments of all the residues in this node and find the optimal assignment such that the energy is minimized. This minimized energy is also the minimum energy of the whole system.

Top-to-Bottom Side Chain Assignment. After finishing calculating the root node of tree T, we have the optimal assignment to all the residues in this root node. 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 we have the optimal

assignment for all the residues in node j and node i is a child node of j. We can easily extract out the optimal assignment for all the residues in $X_i - X_{j,i}$ based on the assignment to the residues in $X_{j,i}$ 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 $X_i - X_{j,i}$ to obtain the optimal assignment to all the residues in $X_i - X_{j,i}$. 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 $X_{j,i}$.

In addition, based on the definition of tree decomposition, one residue may occur in several tree nodes of the tree decomposition of the residue interaction graph. To avoid incorporating the singleton score of this residue into the overall system energy more than once, we incorporate 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 the residue interaction graph may also occur in several tree nodes. We can use the same method to avoid redundant addition of its pairwise score. For example, in Figure 2, residue d occurs in four tree nodes *abd*, *acd*, *cdem* and *defm*. We can only incorporate the singleton score of residue d once in the whole system. We choose to add the singleton score of residue d when we are calculating the optimal score of *acd* since *acd* is the root node with the maximum height. Similarly, edge *de* occurs in both nodes *cdem* and *defm*. Therefore, the interaction score between residues d and e is added when we are optimizing node *cdem*.

The following two lemmas give the running time of the above algorithm.

LEMMA 3.1. *The tree-decomposition-based side-chain assignment algorithm has time complexity* $O(Nn_{rot}^{tw+1})$ *where* N *is the length of the protein,* n_{rot} *the average number of rotamers for each residue, and* tw *the width of the tree decomposition of the interaction graph. The space complexity of this algorithm is* $O(Nn_{rot}^{tw})$.

In Lemma 3.1, $tw + 1$ is the maximum component size in the tree decomposition of the residue interaction graph and $O(n_{rot}^{1+tw})$ is the time complexity of enumerating all the possible side-chain assignments to the residues in one component. According to this lemma, the computational complexity of tree-decomposition-based side-chain assignment algorithm is exponential with respect to the width of the tree decomposition. Therefore, we need an algorithm to decompose the residue interaction graph into very small components.

Assume that the whole protein is inscribed in a minimal axis-parallel 3D rectangle and the width along each dimension is W_x, W_y, and W_z respectively. Without loss of generality, we can also assume that the left-bottom corner of this rectangle is located at the origin and any point in this rectangle has nonnegative coordinates. Then, we have the following lemma.

LEMMA 3.2. *There is an* $O(Nn_{rot}^{O(\min\{W_xW_y, W_xW_z, W_yW_z\})})$ *time complexity algorithm for the globally optimal solution of the side-chain packing problem.*

PROOF. Assume that $W_xW_y = \min\{W_xW_y, W_xW_z, W_yW_z\}$. Using a set of hyperplanes $z = jD_u$ $(j = 1, 2, \ldots, \frac{W_z}{D_u})$, we can partition the protein into $\frac{W_z}{D_u}$ blocks.

All the interaction edges only exist within a block or between two adjacent blocks. Therefore, we can have a tree-decomposition of the protein structure, where each component is formed by two adjacent blocks. Obviously, the tree width of the decomposition is no more than $O(\frac{W_x W_y D_u}{D_l})$. Therefore, the side-chain assignment can be calculated within time complexity $O(Nn_{rot}^{O(\min\{W_x W_y, W_x W_z, W_y W_z\})})$. □

The above lemma gives a very simple tree-decomposition-based algorithm for the side-chain packing problem. In practice, we should use a more complicated decomposition method (e.g., minimum-degree heuristic) to cut the protein into small blocks. Consequently, this lemma gives the upper bound of the time complexity of the tree-decomposition-based algorithm.

3.2. FAST CONSTRUCTION OF A GOOD TREE DECOMPOSITION. Although the optimal tree decomposition of a general graph has been proved to be *NP*-hard [Arnborg et al. 1987], we obtain a polynomial-time algorithm for the residue interaction graph. In particular, since the residue interaction graph is a geometric graph, we can give a polynomial-time algorithm for finding a tree decomposition with width $O(|V|^{\frac{2}{3}} \log |V|)$, based on the sphere separator theorem [Miller et al. 1997]. A critical observation is that in a residue interaction graph each residue can be adjacent to at most $\Delta \leq (1 + \frac{D_u}{D_l})^3$ residues (see Section 2 for the definitions of D_u and D_l).

THEOREM 3.3. *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 linear-time randomized algorithm.*

Before presenting the proof of this theorem, we first introduce the definition of a *k-ply neighborhood system* and the *sphere separator* theorem [Miller et al. 1997].

Definition 3.4. (*k-ply neighborhood system*). A k-ply neighborhood system in $\Re^3$ is a set $\{B_1, B_2, \ldots, B_n\}$ of closed balls in $\Re^3$ such that no point in $\Re^3$ is strictly interior to more than k of the balls.

THEOREM 3.5 (SPHERE SEPARATOR THEOREM). *For every k-ply neighborhood system $\{B_1, B_2, \ldots B_n\}$ in $\Re^3$, there is a sphere separator S such that:* (1) $|N_E| \leq \frac{4}{5}n$ *where N_E contains the set of all balls in the exterior of S;* (2) $|N_I| \leq \frac{4}{5}n$ *where N_I contains the set of all balls in the interior of S; and* (3) $|N_o| = O(k^{\frac{1}{3}} n^{\frac{2}{3}})$ *where N_o contains the set of all balls that intersect S. In addition, such an S can be computed by a linear-time randomized algorithm.*

Using the above theorem, we can prove Theorem 3.3 as follows.

PROOF. In this article, we use the position of C_α atoms to calculate the distance between two residues. For each residue, we construct a ball with radius $\frac{1}{2} D_u$ centered at its C_α atom. In accordance with the definitions of D_u and D_l in Section 2, no point in the three dimensional space is strictly interior to more than $k = \Delta \leq (1 + \frac{D_u}{D_l})^3$ balls. Based on Theorem 3.5, 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 their balls intersecting

with S. Then, we have $|U| = O(|V|^{2/3})$ and U will partition V into two subsets V_1 and V_2 such that $|V_i| \leq \frac{4}{5}|V|$ $(i = 1, 2)$ and there is no interaction edge between V_1 and V_2. □

Using Theorem 3.3, we can prove the following theorem:

THEOREM 3.6. *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. Using Theorem 3.3, we can partition G into two subgraphs G_1 and G_2 by removing a subset of $O(|V|^{\frac{2}{3}})$ residues such that there is no interaction edge between G_1 and G_2 and $\frac{1}{4}(1 - \epsilon) \leq \frac{|V(G_1)|}{|V(G_2)|} \leq 4(1 + \epsilon)$ where $\epsilon = O(|V|^{-\frac{1}{3}})$. 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. Using 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 partition tree 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 completed within linear time, so we can construct such a tree decomposition within low-degree polynomial time. Figure 8 in Appendix A gives an example of this algorithm. □

Combining Theorem 3.6 and Lemma 3.1, we can have the following theorem regarding the computational complexity of our tree decomposition based algorithm.

THEOREM 3.7. *The time complexity of the tree-decomposition-based side-chain assignment algorithm is* $O(Nn_{rot}^{O(N^{2/3} \log N)})$*, as well as the space complexity.*

To the best of our knowledge, this computational complexity is the first one that is not $O(Nn_{rot}^{N})$.

Although we have a low-degree polynomial-time algorithm for finding a tree decomposition of G with a theoretically sound tree width, in practice, we can apply the "minimum degree" heuristic algorithm [Berry et al. 2003] to decompose the graph. An example of tree decomposition using this heuristic algorithm is presented in the Appendix. Later in this article, we will show that the tree decompositions found by the heuristic algorithm are good enough. Many tree decompositions have a tree width of only 3 or 4, which leads to a very efficient algorithm for side-chain prediction.

4. *Methods: Implementation Details*

TreePack uses the backbone-dependent rotamer library described in Dunbrack and Cohen [1997]. The occurring probability of each rotamer depends on the residue type and two angles ψ and ϕ. If the residue does not have two torsion angles, then we use the backbone-independent rotamer library developed by Dunbrack and Cohen [1997]. First, TreePack prunes those rotamers with very low occurring probability.

Just like what is done in SCWRL 3.0 [Canutescu et al. 2003], for a particular residue and its associated two angles ψ and ϕ, TreePack ranks all the candidate rotamers from the highest probability to the lowest and removes the tail candidate if the probabilities of all the rotamers before it add up to 0.90 or above. Then, TreePack applies the Goldstein criterion dead-end elimination (DEE) technique [Goldstein 1994] to remove those rotamers that cannot be a part of the optimal side-chain assignment. Finally, TreePack constructs the residue interaction graph according to its definition described in Section 2. TreePack uses the BALL library [Kohlbacher and Lenhof 2000] for some basic objects such as proteins, residues, and atoms, the ANN library [Arya et al. 1998; Mount and Arya 1997] to determine if two atoms conflict or not, and the split package [Amir 2001] for tree decomposition of a graph. We tested six different tree-decomposition methods implemented in the split package and found that the "minimum-degree" heuristic method generates the smallest tree width most of the time.

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 a certain amount of overhead since much bookkeeping is required for the calculation of one component. We can use a graph reduction technique to remove these low-degree vertices before applying the tree decomposition algorithm to the interaction graph. The reduced system will have the same energy as the original system and can be easily converted into 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 for 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 resulting new interaction graph looks cleaner. The tree decomposition of new interaction graph will generate no components of size one and few components of size two. Please note 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 slightly improve the practical computational efficiency.

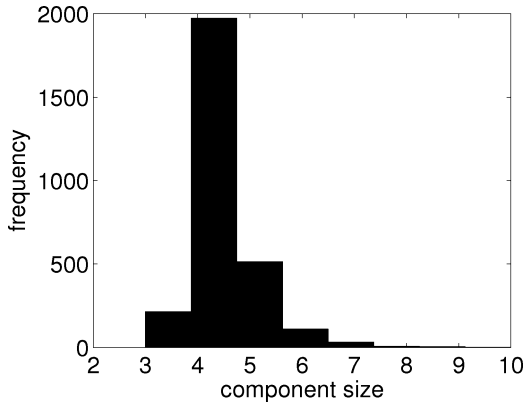

FIG. 4. The component size distribution of "minimum-degree" heuristic decomposition method. The components with size one or two are ignored.

5. *Experimental Results*

In order to compare TreePack with SCWRL 3.0 [Canutescu et al. 2003], we tested TreePack on the same set of 180 proteins as listed in the SCWRL 3.0 paper. The reason that we compared TreePack 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.

5.1. COMPUTATIONAL EFFICIENCY. TreePack runs much more efficiently than SCWRL 3.0. The residue interaction graphs are decomposed into small components containing no more than 10 residues. Most components contain only 4 or 5 residues. Figure 4 shows the distribution of component sizes when the "minimum-degree" tree decomposition algorithm [Berry et al. 2003] is applied to the residue interaction graphs of the 180 proteins. We ran TreePack and SCWRL 3.0 on a Debian Linux box with a 1.7GHz Pentium CPU. TreePack 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, TreePack is more than 5 times faster than SCWRL 3.0. Among these 180 test proteins, the maximum CPU time spent by TreePack on an individual protein is 8.68 seconds. We further compared TreePack with SCWRL 3.0 on some large proteins including 1gai (472 AAs), 1xwl (580 AAs), 1a8i (812 AAs), 1bu7 (910 AAs) and 1b0p (2462 AAs). It takes TreePack 2.98, 4.64, 8.68, 7.62 and 21.03 seconds respectively, while SCWRL takes 266.62, 26.51, 184.36, 56.50, and 300.0 seconds, respectively.

While TreePack and SCWRL 3.0 have the similar overall algorithmic structure, TreePack's speedup is derived from our fast tree-decomposition-based algorithm, as opposed to SCWRL's biconnected-decomposition-based algorithm. In particular, TreePack consists of the following major steps: loading the 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. Using the 180 proteins, the average times are: 0.2822 seconds for library loading, 0.2164 seconds for coordinate calculation, 0.7924 seconds for interaction score calculation, 0.3038 seconds for DEE, and 0.0239 seconds for the tree-decomposition-based algorithm. The average time spent by TreePack on the tree-decomposition-based energy minimization algorithm is only 1.5% of the total

computational time. By observing the running status of SCWRL 3.0 on several large proteins such as 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, TreePack can minimize the energy of 1a8i within 0.4 seconds and 1b0p within 0.7 seconds at the post-DEE stage. Therefore, for large proteins such as 1a8i and 1b0p, if we only consider the the energy minimization algorithm at the post-DEE stage, then TreePack runs more than two hundred times faster than SCWRL 3.0.

Recently, a user (Julio Kovacs) tested TreePack on three real-world instances[1] that SCWRL 3.0 failed to solve within 12 hours. Instead, it takes TreePack no more than 20 minutes on a single 1.7-GHz CPU to solve any of the three instances. These three instances are derived from rotating the helices in the KCSA protein (pdb id 1r3j) by a certain degree. Some rotations lead to heavy atomic clashes so that SCWRL 3.0 failed to terminate within a reasonable amount of time.

5.2. PREDICTION ACCURACY. We measure the consistency between the predicted torsion angles and the real torsion angles as the prediction accuracy of TreePack. While efficiency has been greatly increased, accuracy has not been lost. A prediction of a torsion angle is judged as correct if its deviation from the experimental value is no more than 40 degrees. The prediction accuracy of one amino acid is the ratio of the number of correctly predicted residues to the number of all the residues with this amino acid type. Since TreePack uses the same rotamer library as SCWRL 3.0 and a similar energy function, we should expect that there will not be a big difference in terms of prediction accuracy between TreePack and SCWRL 3.0. Table I lists the prediction accuracy of 18 types of amino acids. In this table, we compare TreePack and SCWRL 3.0 in terms of the prediction accuracy of the torsion angles. Each amino acid has one to four torsion angles depending on its type. Given a residue with known backbone coordinates, if its torsion angles are known, then the coordinates of its side-chain atom can be uniquely determined. The minor difference between TreePack and SCWRL 3.0 comes from the fact that the atom radii in the BALL library differ slightly from those used by SCWRL 3.0. An interesting result is that TreePack does better in predicting CYS and worse in ASN and ASP than SCWRL 3.0.[2] We are going to investigate this interesting phenomena in further projects.

5.3. RELATIONSHIP BETWEEN PERFORMANCE AND ATOM RADII. If the distance between two rotamer atoms is no more than the sum of their radii, then there is an interaction between the two residues to which these two rotamers belong. To tell if there is an interaction between two residues or not, we need to consider the following three factors: the spatial positions of the backbones of the two residues, the size of their rotamers and the orientation of their rotamers. In fact, it is possible that two residues (e.g., TYR and ARG) interact with each other while their spatial distance (i.e., the distance between their C-alpha atoms) is more than $15.4\dot{A}$.

[1]They are available at http://ttic.uchicago.edu/~jinbo/TreePack.htm/.

[2]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 our program. The overall accuracy of SCWRL does not improve if "-u" option is enabled.

TABLE I. PREDICTION ACCURACY OF TREEPACK AND SCWRL 3.0 IN THE 180-PROTEIN TEST SET

	TreePack		SCWRL 3.0	
amino acid	χ_1 accuracy	χ_1 and χ_2 accuracy	χ_1 accuracy	χ_1 and χ_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	0.8241	0.7322	0.8262	0.7374

A prediction of a torsion angle is judged as correct if its deviation from the experimental value is no more than 40 degrees. The prediction accuracy of one amino acid is the ratio of the number of correctly predicted residues to the number of all the residues with this amino acid type. We use χ_1 to denote the first torsion angle and χ_2 the second. Some amino acids have only one torsion angle. "χ_1 and χ_2 accuracy" means that both χ_1 and χ_2 must be correct.

The running time of TreePack depends on the atom radii used in TreePack. To test the dependency of the running time on the atom radii, we tried out three different sets of atom radii: VDW radii, Amber94 radii, and PARSE radii. When tested on the 180 proteins, there is no obvious difference in terms of the running time since TreePack runs really fast on these test proteins. When tested on the three hard examples provided by Julio Kovacs, TreePack runs fast with Amber94 radii and slowly with PARSE radii. Please note that SCWRL failed to run on these three difficult examples. The prediction accuracy also depends on the atom radii. When tested on the 180 proteins, TreePack generates a higher prediction accuracy with PARSE radii and VDW radii and a lower accuracy with Amber94 radii. TreePack uses VDW radii as the default to achieve a good prediction accuracy without losing too much computational efficiency. Tables II and III summarize the performance of TreePack with different atom radii.

5.4. PERFORMANCE ON NONNATIVE BACKBONES. We tested our program TreePack on nonnative backbones generated by a protein threading program. We arbitrarily chose 24 CASP6 test proteins[3] and then used a protein threading server RAPTOR to find their best templates and to generate the alignments between the test proteins and their templates. The test proteins, their PDB IDs and their best

[3]http://predictioncenter.org/casp6/Casp6.html.

TABLE II. PREDICTION ACCURACY OF TREEPACK WITH DIFFERENT ATOM RADII WHEN TESTED ON THE 180 PROTEINS

	χ_1	χ_{1+2}
PARSE	0.826	0.733
VDW	0.824	0.732
Amber94	0.784	0.683

TABLE III. RUNNING TIME (SECONDS) OF TREEPACK ON THREE HARD REAL-WORLD INSTANCES. TREEPACK WAS TESTED ON A LINUX BOX WITH 1.4-GHZ CPU

	PARSE	VDW	Amber94
hard1	3175	1264	542
hard2	5.03	3.70	3.56
hard3	3179	1193	472

TABLE IV. THE CASP6 TEST PROTEINS, THEIR PDB IDS AND THE BEST TEMPLATES GENERATED BY RAPTOR

CASP6 ID	t0199	t0200	t0201	t0202	t0203	t0205	t0208	t0217
PDB ID	1stz	1t70	1s12	1suw	1vkp	1vm0	1tz9	1vpq
template	1mkmA	1hplA	1ml8A	1nnwA	1rxxA	1h0xA	1k77A	1i60A
CASP6 ID	t0219	t0222	t0223	t0224	t0225	t0226	t0228	t0229
PDB ID	1vrb	1vli	1vkw	1rhx	1vl0	1wiw	1vlp	1vla
template	1h2kA	1o60A	1bkjA	1jx7A	1n7hA	1c7qA	1qapA	1n2fA
CASP6 ID	t0231	t0232	t0233	t0234	t0235	t0246	t0247	t0252
PDB ID	1vkk	1zgm	1vqu	1vl7	1vjv	1vlc	1vlo	1u60
template	1cfyA	1f3aA	1ol7A	1dnlA	1nbfA	1hqsA	1pj5A	1mkiA

templates are listed in Table IV. Most of the test proteins have a good alignment to their best templates. The backbone coordinates of one test protein are generated using MODELLER [Sali and Blundell 1993] based on the alignment between the test protein and its template.[4] We predicted the side-chain coordinates of the test proteins using the following four side-chain prediction programs: TreePack, SCWRL 3.0, MODELLER and SCAP [Xiang and Honig 2001]. SCAP was tested using the CHARMM force field with the heavy atom model and the largest rotamer library available to SCAP. SCAP runs very slowly compared to the other three programs. Table V gives the overall prediction accuracy of the four programs. As shown in this table, the prediction accuracy of TreePack is comparable to SCWRL 3.0 and much better than those of SCAP and MODELLER. This result indicates that TreePack also works well for the nonnative backbones.

6. *Theoretical Results*

6.1. HARDNESS RESULTS. The side-chain packing problem is still *NP*-hard even if a protein structure is modeled by a geometric neighborhood graph. In previous

[4]All the backbone data of the test proteins is available at http://ttic.uchicago.edu/~jinbo/TreePack.htm.

TABLE V. PREDICTION ACCURACY OF TREEPACK, SCWRL 3.0, MODELLER AND SCAP ON 24 NONNATIVE BACKBONES

	χ_1	χ_{1+2}
TreePack	0.520	0.314
SCWRL3.0	0.530	0.334
SCAP	0.488	0.259
MODELLER	0.428	0.220

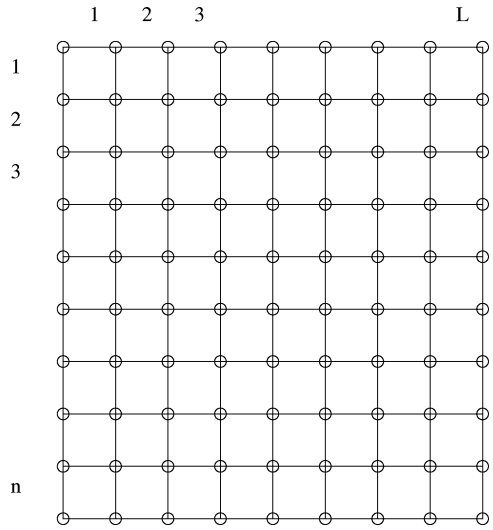

FIG. 5. A protein with nL residues. Each grid point corresponds to a residue.

papers [Pierce and Winfree 2002; Chazelle et al. 2004], the hardness proof of this problem was based on a very general graph representation of protein structures. Therefore, it is necessary to provide a new hardness proof for our model since a residue interaction graph provides more information than a general graph. Here, we prove the following two hardness results, assuming that there is no interaction between two residues if their distance is beyond a constant D_u. However, we do not specify the singleton and pairwise interaction scores.

THEOREM 6.1. *Protein side-chain packing is NP-hard even if all the energy items are negative or zero.*

PROOF. We reduce the longest common subsequence problem to this problem. The longest common subsequence problem is defined as follows. Given any n strings $s_1, s_2, \ldots, s_n$ of length m, and an integer L, is there a common subsequence of $s_1, s_2, \ldots, s_n$ of length more than L? For a detailed description and hardness proof of the longest common subsequence problem, please refer to Jiang and Li [1995]. Here we use $s_i(l)$ to denote the lth character of string s_i.

As shown in Figure 5, we construct a protein structure in a 2D grid of length n and width L. Each grid point (i, j) $(i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, L)$ represents one of the $N = nL$ residues. Assume there is an interaction edge between two residues if and only if their distance in the grid is 1. Assume that each residue at

grid point (i, j) has m rotamers $R_{i,l}(l = 1, 2, \ldots, m)$, each of which is uniquely identified by a character $s_i(l)$ and its position l in the string s_i. Let $c(R)$ denote the character type of rotamer R. We define the side chain assignment scoring function as follows:

—For a pair of residues $r_1 = (i, j)$ and $r_2 = (i + 1, j)$, if their rotamers are R_{i,k_1} and R_{i+1,k_2} respectively and $c(R_{i,k_1}) = c(R_{i+1,k_2})$, then the pairwise score $P_{r_1,r_2}(R_{i,k_1}, R_{i+1,k_2}) = -1$. Otherwise, $P_{r_1,r_2}(R_{i,k_1}, R_{i+1,k_2}) = 0$.

—For a pair of residues $r_1 = (i, j)$ and $r_2 = (i, j+1)$, if their rotamers are R_{i,k_1} and R_{i,k_2} and $k_1 < k_2$, then the pairwise score $P_{r_1,r_2}(R_{i,k_1}, R_{i,k_2}) = -nL$. Otherwise, $P_{r_1,r_2}(R_{i,k_1}, R_{i,k_2}) = 0$. This condition maintains the order of the characters in a string.

—All other scores are 0.

There is a common subsequence of length L among $s_1, s_2, \ldots, s_n$ if and only if there is a side-chain assignment with a score no more than $-n^2L(L-1)-(n-1)L$. On one hand, if there is a common subsequence of length L, then for the residue at grid point (i, j), we choose the rotamer R_{i,l_j} identified by $s_i(l_j)$ and l_j, where l_j is the position of the jth common subsequence character in string s_i. The resultant system energy is $-n^2L(L-1)-(n-1)L$. On the other hand, if there is a side-chain assignment such that the system energy is no more than $-n^2L(L-1)-(n-1)L$, notice that the system energy is no less than $-n^2L(L-1)-(n-1)L$. In order to obtain a system energy of $-n^2L(L-1)-(n-1)L$, the pairwise interaction score between any two horizontally adjacent residues should be $-nL$ and the pairwise score between any two vertically adjacent residues should be -1. So the side-chain assignment implies a common subsequence of at least L. Since the longest common subsequence problem is NP-hard, this argument completes the proof. □

THEOREM 6.2. *Protein side-chain packing is NP-hard even if all energy items are nonnegative.*

PROOF. This theorem can be proved using a technique similar to the proof of Theorem 6.1. However, the following scoring function is used.

—For a pair of residues (i, j) and $(i + 1, j)$, their pairwise score is 0 if and only if their rotamers are R_{i,k_1} and R_{i+1,k_2} and $c(R_{i,k_1}) = c(R_{i+1,k_2})$. Otherwise, their pairwise score is 1.

—For a pair of residues (i, j) and $(i, j + 1)$, their pairwise score is 0 if and only if their rotamers are R_{i,k_1} and R_{i,k_2} and $k_1 < k_2$. Otherwise, their pairwise score is 1.

—All other scores are 0.

Using a similar analysis in the proof of Theorem 6.1, It is easy to show that a common subsequence of length L exists among $s_1, s_2, \ldots, s_n$ if and only if there is a side-chain assignment with score 0. □

6.2. POLYNOMIAL-TIME APPROXIMATION SCHEMES. We introduce several polynomial-time approximation schemes (PTAS) for the side-chain packing problem, each scheme suitable for a different type of energy function, some of which make sense in practice. These results indicate that given any arbitrary error ϵ $(0 < \epsilon < 1)$, there is a polynomial-time algorithm (with respect to N and n_{rot})

generating a side-chain assignment such that the system energy is within a factor of $(1 \pm \epsilon)$ times the lowest. In contrast, using a general graph model of protein structures, one group [Chazelle et al. 2004] proved that it is *NP*-complete to approximate this problem within a factor of cN where c is a positive constant.

The basic idea underlying these algorithms is to partition the whole protein structure into small blocks, assign side chains to each block separately, and then combine the assignments of all the blocks. We assume that the protein under consideration is inscribed in a minimal 3D rectangle with dimensions W_x, W_y, and W_z and the left bottom corner of the rectangle is the origin point.

LEMMA 6.3. *If every energy item in Eq.* (3) *is negative, then there is an algorithm with time complexity* $O(kNn_{rot}^{O(W_x k)})$ *for side-chain assignment such that the system energy is no more than* $(1 - \frac{2}{k})$ *times the lowest.*

PROOF. For simplicity, in this proof, we assume that we want to maximize Eq. (3) and all the energy items in this equation are positive. Therefore, we need to prove that there is an algorithm to assign side chains such that the system energy is at least $(1 - \frac{2}{k})$ times the best possible. The intuition is to cut the protein structure into some nonoverlapping blocks using k different partitioning schemes. Each block can be tree-decomposed into some components containing no more than $O(kW_x)$ residues. Therefore, the side-chain packing of each block can be done within time proportional to $O(n_{rot}^{O(kW_x)})$, using the globally optimal algorithm on each block. We then prove that among k different partitioning schemes, at least one can give us a good side-chain packing. Please see Figure 6 for an example of $k(= 3)$ different partition schemes. Using a group of hyperplanes $y = y_j = jD_u$ $(j = 0, 1, \ldots, \frac{W_y}{D_u})$, we can partition the protein into $\frac{W_y}{D_u}$ basic blocks along the y-axis, each of which has dimension $W_x \times D_u \times W_z$. Let B_j $(j = 1, 2 \ldots, \frac{W_y}{D_u})$ denote the set of residues contained in the basic block $\{(x, y, z) | 0 \le x \le W_x, y_{j-1} \le y < y_j, 0 \le z \le W_z\}$. Let R_j denote the union of $B_{j+1}, B_{j+2}, \ldots, B_{j+k-1}$.[5] Let $G(R_j)$ denote the subgraph induced by R_j plus the interaction edges connecting R_j and B_j. Similarly, let $G(B_j)$ denote the subgraph induced by B_j plus the interaction edges connecting B_j and B_{j-1}.

We optimize the side chain assignment of the protein structure using k different partition schemes and prove that at least one of them will give a good side-chain packing. For a given partition scheme s $(0 \le s < k)$, let $RS_s = \bigcup_{j:j\%k=s} G(R_j)$ and $BS_s = \bigcup_{j:j\%k=s} G(B_j)$.[6] As shown in Figure 6, RS_s refers to the shadowed areas and BS_s refers to the nonshadowed areas. We optimize the side-chain assignment for the residues in RS_s first using our tree-decomposition-based algorithm and then for the remaining residues (that is, the residues in the white area) while fixing the side-chain assignment to the residues in RS_s. Let $E(RS_s)$ and $E(BS_s)$ denote the optimal energy of RS_s and BS_s, respectively, and $E_s = E(RS_s) + E(BS_s)$. The union of RS_s and BS_s contains all the residues and inter-residue interaction edges in the protein. Since all the energy items are non-negative, the total energy E_s is greater

[5]If the subscript of B is greater than $\frac{W_y}{D_u}$, then we replace the subscript with its module over $\frac{W_y}{D_u}$.
[6]In this proof, $j\%k$ represents the module of j over k.

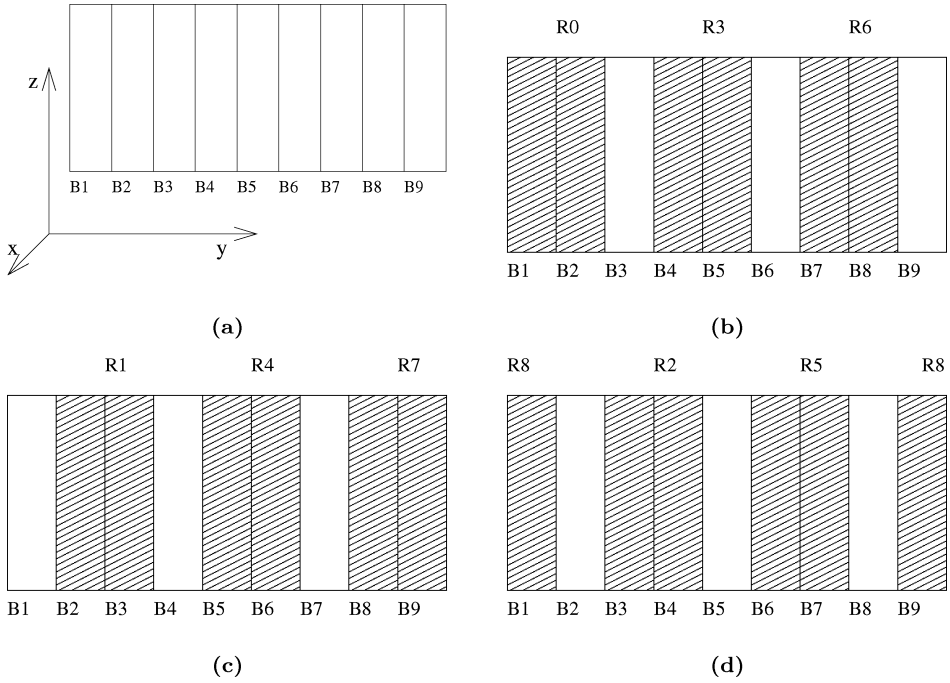

FIG. 6. Example of protein structure partition schemes for a given integer $k = 3$. (a) A protein structure is cut into some basic blocks $B_1, B_2, \ldots B_9$ along the y-axis. Each B_i has dimension $W_x \times D_u \times W_z$. (b) Partition scheme 0: $R_0 = B_1 \bigcup B_2$, $R_3 = B_4 \bigcup B_5$, and $R_6 = B_7 \bigcup B_8$. (c) Partition scheme 1: $R_1 = B_2 \bigcup B_3$, $R_4 = B_5 \bigcup B_6$, and $R_7 = B_8 \bigcup B_9$. (d) Partition scheme 2: $R_2 = B_3 \bigcup B_4$, $R_5 = B_6 \bigcup B_7$, and $R_8 = B_9 \bigcup B_1$. For each partition scheme, we do side-chain assignment for all the residues in the shadowed areas first and then for the remaining residues. In general, each shadowed area contains $k - 1$ basic blocks and its tree width is $O(k \min\{W_x, W_z\})$.

than or equal to the globally optimized energy E_{opt}.

$$E_s = E(RS_s) + E(BS_s) \geq E_{opt}. \tag{6}$$

Summing over all values of s in Eq. (6), we have the following:

$$\sum_{0 \leq s < k} E_s \geq kE_{opt}. \tag{7}$$

Now we will prove that $\sum_{s=0}^{k-1} E(BS_s)$ is no more than $2E_{opt}$. Then, the sum of all the $E(RS_s)$ is at least $(k-2)E_{opt}$ and there is at least one s^* such that $E(RS_{s^*}) \geq (1 - \frac{2}{k})E_{opt}$. Therefore, there is a side-chain assignment with energy at least $(1 - \frac{2}{k})E_{opt}$.

The union of all the BS_s is equal to $\bigcup G(B_j)$, which can be divided into two disjoint subsets $\bigcup_j G(B_{l+2j})$ $(0 \leq l < 2)$. For a given l, $G(B_{l+2j_1})$ and $G(B_{l+2j_2})$ are disjoint if $j_1 \neq j_2$. Notice that we assume that all the energy items are nonnegative, so the whole energy of $\bigcup_j G(B_{l+2j})$ is no more than E_{opt} no matter how we do the side-chain assignment. Therefore, $\sum_{s=0}^{k-1} E(BS_s)$ is no more than $2E_{opt}$.

For each partition scheme s, the algorithm needs to optimize the side-chain assignment for the residues in RS_s. Based on Lemma 3.2, the side-chain assignment for the residues in R_j can be optimized by an algorithm with time complexity $O(|R_j| n_{rot}^{O(kW_x)})$. Once the side-chain assignments for the residues in RS_s are fixed,

the algorithm assigns side chains to the remaining residues in BS_s. So the time complexity of doing side-chain packing for each partition scheme is $O(Nn_{rot}^{O(kW_x)})$ and the time complexity of this algorithm is $O(kNn_{rot}^{O(kW_x)})$. □

Lemma 6.3 is proved by cutting the whole protein structure along the y-axis. We can further cut the protein along the x-axis to obtain the following desired theorem.

THEOREM 6.4. *If every energy item in Eq.* (3) *is negative and the system energy should be minimized, then there is an algorithm with time complexity* $(k^2Nn_{rot}^{O(k^2)})$ *to do side-chain assignment such that the system energy is no more than* $(1 - \frac{4}{k})$ *times the best possible.*

The detailed proof of the above theorem is shown in the Appendix. In the energy function used by SCWRL 3.0, the pairwise energy items are positive and bounded from above by 10 units, so Theorem 6.4 cannot be applied. Instead, we have the following theorem.

THEOREM 6.5. *Assume that all the pairwise energy items in Eq.* (3) *are positive and the system energy should be minimized. There is an algorithm with time complexity* $O(l^2N^{O(l^2)+1}\log N/\log n_{rot})$ *to do side-chain assignment such that the system energy is no more than* $(1 + \frac{1}{l})$ *times the best possible if the lowest system energy is* $\Omega(NP_{\max}\sqrt{\log n_{rot}/\log N})$, *where* $P_{\max}$ *is the maximum among all* $P_{i,j}(A(i), A(j))$.

The detailed proof of Theorem 6.5 is in the Appendix. If all the pairwise scores are negative, then using a technique similar to the proof of Theorem 6.5, we have the following theorem.

THEOREM 6.6. *Assume that all the pairwise scores in Eq.* (3) *are negative and the system energy should be minimized. There is an algorithm with time complexity* $O(l^2N^{O(l^2)+1}\log N/\log n_{rot})$ *to do side-chain assignment such that the system energy is no more than* $(1 - \frac{1}{l})$ *times the best possible if the lowest system energy is no more than* $cNP_{\min}\sqrt{\log n_{rot}/\log N}$, *where* c *is a positive constant and* $P_{\min}$ *is the minimum among all* $P_{i,j}(A(i), A(j))$.

Theorems 6.5 and 6.6 indicate that we can approximate the system energy within a factor of $(1 \pm \epsilon)$ if the lowest system energy is bounded far away from 0. That is, we can have a good approximation algorithm if there is no way to avoid all the atomic clashes in the whole system. This condition usually holds if the protein backbone is predicted by a protein threading technique or if there are heavy potential atomic clashes in the system.

7. *Conclusions*

We presented a tree-decomposition-based algorithm, TreePack, for protein side-chain prediction. This algorithm runs much faster than SCWRL 3.0, a widely used side-chain prediction program, while maintaining a similar accuracy level. Using the tree decomposition of protein structures, we not only give a fast, rigorous and accurate protein side-chain assignment method, but also develop several polynomial-time approximation schemes (PTAS) for this problem. These theoretical results indicate that the protein side-chain packing problem is not as computationally hard

as previously believed. These theoretical results also open up a new avenue of study for the side-chain packing problem. On one hand, they stimulate us to develop a physically meaningful energy function so that the PTAS algorithms can be used to approximate the system energy to an arbitrary precision. On the other hand, we could further enlarge the rotamer library to take advantage of the computational efficiency of our algorithms, which should in turn give better prediction accuracy.

We are currently pursuing combining the tree decomposition technique and the protein structure partition technique used in the PTAS algorithm to develop a heuristic algorithm for the side-chain packing problem with any kind of energy function. That is, if the tree width of the whole residue interaction graph is big, then we first cut the protein structure into several small blocks such that each small block has a reasonable tree width. Then we use the tree-decomposition-based algorithm to do side-chain assignment for each small block separately. Using various partition schemes, this heuristic algorithm should be able to generate a reasonable side-chain packing for the whole protein structure.

Appendix A.

A.1 AN EXAMPLE OF TREE DECOMPOSITION PROCEDURE. Figure 7 gives an example of decomposing a graph into a tree decomposition using a "minimum-degree" based heurisitics algorithm.

A.2. AN EXAMPLE FOR THE PROOF OF THEOREM 3.6. Figure 8 gives an example of the algorithm described in the proof of Theorem 3.6. Please note that the tree decomposition generated by this algorithm might not have the optimal treewidth. However, we can guarantee that its treewidth is $O(|V|^{2/3} \log |V|)$.

A.3. PROOF OF THEOREM 6.4. For simplicity, we assume that we want to maximize Eq. (3) and all the energy items are positive. Therefore, we need to prove that there is an algorithm to generate a side-chain assignment with energy at least $(1 - \frac{4}{k})$ times the optimal. The proof of this theorem is very similar to that of Lemma 6.3. We partition the protein structure along the x-axis into $\frac{W_x}{D_u}$ blocks, using hyperplanes $x = x_i = i D_u$ $(i = 0, 1, \ldots, \frac{W_x}{D_u})$. Let R_i denote the partial protein structure contained in the rectangle $\{(x, y, z) | x_{i+1} \leq x < x_{i+k}\}$ and let T_i denote the partial protein structure contained in $\{(x, y, z) | x_i \leq x < x_{i+1}\}$. In accordance with Lemma 6.3, we have an approximation algorithm to optimize the energy of the subgraph induced by R_i and T_i, respectively. Similarly, We can also prove that the sum of the energies of all the subgraphs T_i is no more than $2E_{opt}$. Therefore, the best energy obtained by this algorithm is no less than $(1 - \frac{4}{k})E_{opt}$. The time complexity is $(k^2 N n_{rot}^{O(k^2)})$.

A.4. PROOF OF THEOREM 6.5. Since we are minimizing a positive objective function, we need to show that there is a polynomial-time algorithm to assign side chains so that the system energy is no more than $(1 + \epsilon)$ times the optimal. We use a slightly different partition method to cut the protein structure into small blocks. We cut the protein structure along both the x-axis and y-axis into $\frac{W_x}{D_u} \times \frac{W_y}{D_u}$ basic blocks $B_{i,j}$, using $\frac{W_x}{D_u} + \frac{W_y}{D_u} + 2$ hyperplanes $x = x_i = i \frac{W_x}{D_u}$ $(i = 0, 1, \ldots, \frac{W_x}{D_u})$, $y = y_j = j \frac{W_y}{D_u}$ $(j = 0, 1, \ldots, \frac{W_y}{D_u})$. Each basic block $B_{i,j}$ has dimension $D_u \times D_u \times W_z$ and is defined by $\{(x, y, z) | x_i \leq x < x_{i+1}, y_j \leq y < y_{j+1}, 0 \leq z < W_z\}$. For a

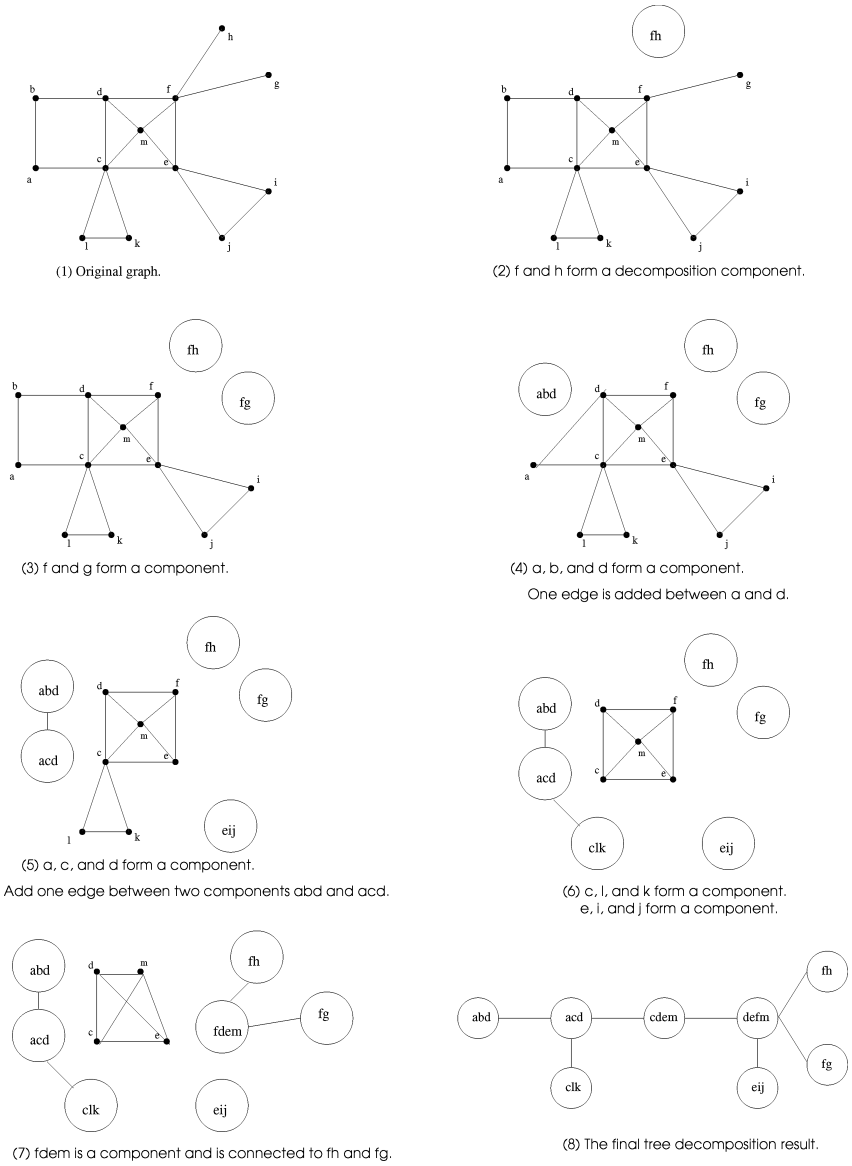

FIG. 7. An illustration of tree decomposition process using the minimum-degree heuristic technique. This technique employs the following procedures: (i) choose the vertex with the minimal degree. The chosen vertex and its neighbors form a component; (ii) make the neighbors of the chosen vertex form a clique; (iii) remove the chosen vertex and repeat the above two steps on the remaining subgraph until the subgraph is empty.

given integer k, let $R_{i,j}$ denote the union of $B_{p,q}$ ($i \le p < i + k$, $j \le q < j + k$). We optimize the side-chain assignment for the protein structure using k^2 different partition schemes and prove that at least one of them will give a good side-chain packing. Given a partition scheme (r, c) $(0 \le r < k, 0 \le c < k)$, let $RS_{r,c}$ denote the set of all $R_{r+ik,c+jk}$ ($i = 0, 1, \ldots, j = 0, 1, \ldots,$). Obviously, any two R_{i_1,j_1}, R_{i_2,j_2} in $RS_{r,c}$ are disjoint. Let $C_{r,c}$ denote all the interaction edges between two

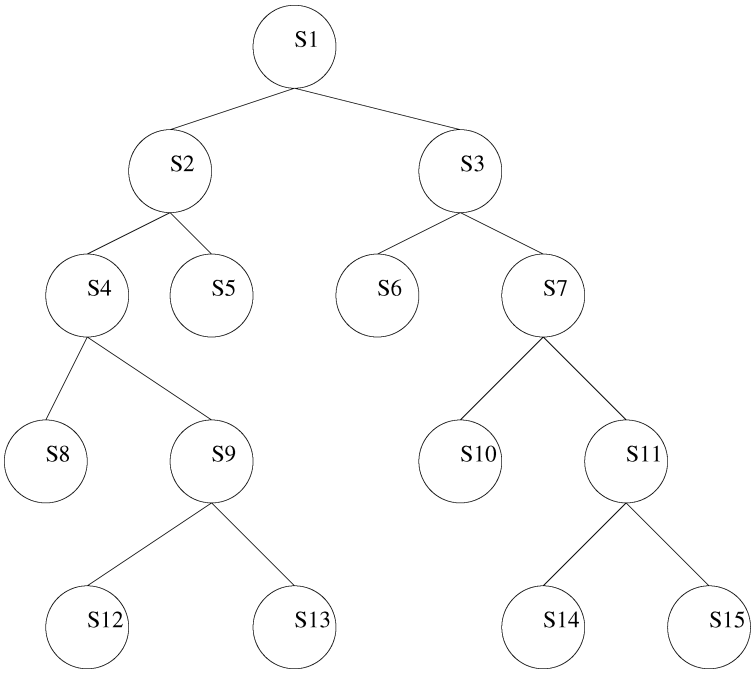

FIG. 8. An illustration of tree decomposition process using the algorithm presented in the proof of Theorem 3.6. Each balanced separator S_i contains $O(|V|^{2/3})$ vertices and corresponds to a tree decomposition component. All the separators on the path from S_i to S_1 form a decomposition component. The height of the tree is $O(\log |V|)$.

different R in $RS_{r,c}$. We assign side chains to all the residues in the protein structure by optimizing the energy of each subsystem induced by an individual R in $RS_{r,c}$ independently. Let $E(R_{r,c})$ denote the optimal energy of the subsystem $R_{r,c}$ and let $E(C_{r,c})$ denote the energy of $C_{r,c}$ given that the side-chain assignment to all the residues in $RS_{r,c}$ are already fixed. We have the following equation.

$$E_{opt} \leq E(R_{r,c}) + E(C_{r,c}) \leq E_{opt} + E(C_{r,c}). \tag{8}$$

Summing over all values of r, c in the above equation, we have

$$k^2 E_{opt} \leq \sum_{0 \leq r < k, 0 \leq c < k} (E(R_{r,c}) + E(C_{r,c})) \leq k^2 E_{opt} + \sum_{0 \leq r < k, 0 \leq c < k} E(C_{r,c}). \tag{9}$$

Since in the residue interaction graph each residue can only be adjacent to $O((\frac{D_u}{D_l})^3)$ residues, $|\cup_{0 \leq r < k, 0 \leq c < k} C_{r,c}| \leq O(kN)$ holds. Therefore, we have

$$\sum_{0 \leq r < k, 0 \leq c < k} E(C_{r,c}) \leq O(kNP_{\max}). \tag{10}$$

There is at least a pair of (r_0, c_0) such that

$$E_{r_0,c_0} = E(R_{r_0,c_0}) + E(C_{r_0,c_0}) \leq E_{opt} + \frac{NP_{\max}}{k} \tag{11}$$

$$E_{r_0,c_0} \leq E_{opt}\left(1 + \frac{NP_{\max}}{E_{opt}k}\right). \tag{12}$$

The running time of this algorithm is $k^2 N n_{rot}^{O(k^2)}$. If k^2 is equal to $l^2 \log N / \log n_{rot}$, then $\frac{NP_{\max}}{Ek}$ is $O(\frac{1}{l})$ since E_{opt} is $\Omega(NP_{\max}\sqrt{\log n_{rot}/\log N})$. Therefore, we have a PTAS algorithm with time complexity $O(l^2 N^{l^2+1} \log N / \log n_{rot})$ for the side-chain packing problem.

ACKNOWLEDGMENTS. We are grateful to Dr. Julio Kovacs, who tried out TreePack and gave us helpful comments and test instances. The authors would like to thank all the anonymous reviewers, whose insightful comments are very helpful to the improvement of the article.

REFERENCES

AKUTSU, T. 1997. NP-hardness results for protein side-chain packing. In *Genome Informatics 8*, S. Miyano and T. Takagi, Eds. 180–186.

ALEXANDROV, N., NUSSINOV, R., AND ZIMMER, R. 1996. Fast protein fold recognition via sequence to structure alignment and contact capacity potentials. In *Biocomputing: Proceedings of 1996 Pacific Symposium*.

AMIR, E. 2001. Efficient approximation for triangulation of minimum treewdith. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI '01)*. 7–15.

ARNBORG, S., CORNEIL, D., AND PROSKUROWSKI, A. 1987. Complexity of finding embedding in a k-tree. *SIAM J. Algeb. Disc. Meth. 8*, 277–284.

ARYA, S., MOUNT, D., NETANYAHU, N., SILVERMAN, R., AND WU, A. 1998. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM 45*, 891–923.

BACH, F., AND JORDAN, M. 2002. Thin junction trees. In *Advances in Neural Information Processing Systems (NIPS)*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds. Vol. 14. 569–574.

BERRY, A., HEGGERNES, P., AND SIMONET, G. 2003. The minimum degree heuristic and the minimal triangulation process. In Lecture Notes in Computer Science, vol. 2880. Springer-Verlag, New York, 58–70.

BOWER, M., COHEN, F., AND DUNBRACK, JR., R. L. 1997. Prediction of protein side-chain rotamers from a backbone-dependent rotamer library: A new homology modeling tool. *J. Mol. Biol. 267*, 1268–1282.

CANUTESCU, A., SHELENKOV, A., AND DUNBRACK, JR., R. L. 2003. A graph-theory algorithm for rapid protein side-chain prediction. *Prot. Sci. 12*, 2001–2014.

CHAZELLE, B., KINGSFORD, C., AND SINGH, M. 2004. A semidefinite programming approach to side-chain positioning with new rounding strategies. *INFORMS J. Comput. Special Issue in Computational Molecular Biology/Bioinformatics*, 86–94.

DESMET, J., MAEYER, M. D., HAZES, B., AND LASTER, I. 1992. The dead-end elimination theorem and its use in protein side-chain positioning. *Nature 356*, 539–542.

DESMET, J., SPRIET, J., AND LASTER, I. 2002. Fast and accurate side-chain topology and energy refinement (faster) as a new method for protein structure optimization. *Protein: Struct. Funct. Gen. 48*, 31–43.

DUKKA, K., TOMITA, E., SUZUKI, J., AND AKUTSU, T. 2004. Protein side-chain packing problem: a maximum common edge-weight clique algorithmic approach. In *Proceedings of the 2nd Asia Pacific Bioinformatics Conference*. 191–200.

DUNBRACK JR., R. L. 1999. Comparative modeling of CASP3 targets using PSI-BLAST and SCWRL. *Protein: Struct. Funct. Gen. 3*, 81–87.

DUNBRACK JR., R. L., AND COHEN, F. 1997. Bayesian statistical analysis of protein side-chain rotamer preferences. *Protein Sci. 6*, 1661–1681.

ERIKSSON, O., ZHOU, Y., AND ELOFSSON, A. 2001. Side chain-positioning as an integer programming problem. In *Proceedings of the 1st International Workshop on Algorithms in Bioinformatics*. Springer-Verlag, New York, 128–141.

GOLDSTEIN, R. 1994. Efficient rotamer elimination applied to protein side-chains and related spin glasses. *Biophys. J. 66*, 1335–1340.

HOLM, L., AND SANDER, C. 1991. Database algorithm for generating protein backbone and sidechain coordinates from a ca trace: Application to model building and detection of coordinate errors. *J. Mol. Biol. 218*, 183–194.

JIANG, T., AND LI, M. 1995. On the approximation of shortest common supersequencesand longest common subsequences. *SIAM J. Comput. 24*, 5, 1122–1139.

JONES, D. 1999. GenTHREADER: An efficient and reliable protein fold recognition method for genomic sequences. *J. Mol. Biol. 287*, 797–815.

KELLEY, L., MACCALLUM, R., AND STERNBERG, M. 2000. Enhanced genome annotation using structural profiles in the program 3D-PSSM. *J. Mol. Biol. 299*, 2, 499–520.

KINGSFORD, C. L., CHAZELLE, B., AND SINGH, M. 2005. Solving and analyzing side-chain positioning problems using linear and integer programming. *Bioinformatics 21*, 1028–1036.

KOHLBACHER, O., AND LENHOF, H. 2000. BALL—Rapid software prototyping in computational molecular biology. *Bioinformatics 16*, 9, 815–824.

LEACH, A., AND LEMON, A. 1998. Exploring the conformational space of protein side chains using dead-end elimination and the A* algorithm. *Protein: Struct. Funct. Gen. 33*, 227–239.

LEE, C., AND SUBBIAH, S. 1991. Prediction of protein side-chain conformation by packing optimization. *J. Mol. Biol. 217*, 373–388.

LI, W., PIO, F., PAWLOWSKI, K., AND GODZIK, A. 2000. Saturated blast: Detecting distant homology using automated multiple intermediate sequence blast search. *Bioinformatics 16*, 1105–1110.

LIANG, S., AND GRISHIN, N. 2002. side-chain modelling with an optimized scoring function. *Protein Sci. 11*, 322–331.

MILLER, G. L., TENG, S., THURSTON, W., AND VAVASIS, S. A. 1997. Separators for sphere-packings and nearest neighbor graphs. *J. ACM 44*, 1, 1–29.

MOULT, J., FIDELIS, F., ZEMLA, A., AND HUBBARD, T. 2001. Critical assessment of methods on protein structure prediction (CASP)-round IV. *Proteins: Struct. Funct. Gen. 45*, S5 (Dec.), 2–7.

MOULT, J., FIDELIS, F., ZEMLA, A., AND HUBBARD, T. 2003. Critical assessment of methods on protein structure prediction (CASP)-round V. *Proteins: Struct., Funct. Gen. 53*, S6 (Oct.), 334–339.

MOULT, J., HUBBARD, T., FIDELIS, F., AND PEDERSEN, J. 1999. Critical assessment of methods on protein structure prediction (CASP)-round III. *Proteins: Struct. Funct. Gen. 37*, S3 (Dec.), 2–6.

MOUNT, D., AND ARYA, S. 1997. ANN: A library for approximate nearest neighbor searching. In *Proceedings of the 2nd CGC Workshop on Computational Geometry*.

PIERCE, N., AND WINFREE, E. 2002. Protein design is NP-hard. *Protein Engi. 15*, 10, 779–782.

ROBERTSON, N., AND SEYMOUR, P. 1986. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms 7*, 309–322.

SALI, A., AND BLUNDELL, T. 1993. Comparative protein modelling by satisfaction of spatial restraints. *J. Mol. Biol.*, 779–815.

SAMUDRALA, R., AND MOULT, J. 1998. Determinants of side chain conformational preferences in protein structures. *Protein Engi. 11*, 991–997.

SHI, J., TOM, L. B., AND KENJI, M. 2001. FUGUE: Sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *J. Mol. Biol. 310*, 243–257.

SUMMERS, N., AND KARPLUS, M. 1989. Construction of side-chains in homology modelling: Application to the c-terminal lobe of rhizopuspepsin. *J. Mol. Biol. 210*, 785–811.

VON OHSEN, N., SOMMER, I., ZIMMER, R., AND LENGAUER, T. 2004. Arby: automatic protein structure prediction using profile-profile alignment and confidence measures. *Bioinformatics 20*, 14 (Sept.), 2228–35.

XIANG, Z., AND HONIG, B. 2001. Extending the accuracy limits of prediction for side-chain conformations. *J. Mol. Biol. 311*, 421–430.

XU, J., LI, M., KIM, D., AND XU, Y. 2003a. RAPTOR: optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology 1*, 1, 95–117.

XU, J., LI, M., LIN, G., KIM, D., AND XU, Y. 2003b. Protein threading by linear programming. In *Biocomputing: Proceedings of the 2003 Pacific Symposium*. Hawaii, USA, 264–275.

XU, Y., XU, D., AND UBERBACHER, E. 1998. An efficient computational method for globally optimal threadings. *J. Comput. Biol. 5*, 3, 597–614.

RECEIVED AUGUST 2005; REVISED FEBRUARY 2006 AND MAY 2006; ACCEPTED JUNE 2006