

A New Method for Modeling and Solving the Protein Fold Recognition Problem

(Extended Abstract)

Ying Xu, Dong Xu, and Edward C. Uberbacher

Computational Biosciences Section, Life Sciences Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6480, USA
Email: {xyn, xud, ube}@ornl.gov

Abstract

Computational recognition of native-like folds from a protein fold database is considered to be a promising alternative approach to the ab initio fold prediction. We present a new and effective method for protein fold recognition through optimally aligning (threading) an amino acid sequence and a protein fold (template). A protein fold, in our database, is represented as a series of core secondary structures, and the alignment quality is determined by three factors. They are (1) the fitness between each amino acid and the environment of its assigned (aligned) template position; (2) pairwise interaction preferences between amino acids that are spatially close; and (3) alignment gap penalties. Our threading algorithm constructs an optimum alignment between an amino acid sequence of size n and a protein fold template of size m in $O((m+n)^{1+0.5C} M \log(n) n^{C+1})$ time and $O(nm+n^{C+2})$ space, where M is the number of core secondary structures in the fold, and C is a (small) non-negative integer, determined by a mathematical property of the pairwise interactions in the fold. C is less than or equal to 4 for about 75% of the 296 unique folds in our database, when pairwise interactions are restricted to amino acids $\leq 7\text{\AA}$ apart (measured between their beta carbon atoms). An approximation scheme is developed for fold templates with $C > 4$, when threading requires too much memory and time to be practical on a typical workstation.

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.

RECOMB 98 New York NY USA
Copyright 1998 0-89791-976-9/98/3...\$5.00

1 Introduction

Protein fold recognition concerns recognizing the native-like folds (3D structures) of an amino acid sequence, from a protein fold database. It is considered to be a promising approach to predicting the folded structure of a protein from its sequence, based on the general belief that there is only a small number of "domain core" folds in nature, and some averaged preference to particular structural/solvent environments over the entire sequence is sufficient for recognition of its native-like folds [1].

Typically, fold recognition is achieved through optimally aligning an amino acid sequence with each of the protein folds (*protein threading*) in the database. Folds with "good" alignment scores are recognized as native-like folds of the sequence. The quality of such a sequence-structure alignment is determined by (1) the fitness of each individual assignment of an amino acid to a template position, (2) gap penalties for the unaligned amino acids and template positions, and (3) interaction preferences among aligned amino acids that are spatially close.

In formulating the protein threading problem, we follow a few basic assumptions widely used by the protein threading community. We assume that (i) each protein fold is represented as a series of core secondary structures (α -helices and β -sheets) with the connecting loops being removed; (ii) when aligning an amino acid sequence with a fold template, successive positions of each core can only be occupied by adjacent amino acids in the sequence, and alignment gaps are confined to the connecting loop regions or the ends of a core; (iii) only pairwise interactions are considered; and (iv) all terms measuring the fitness of individual assignments, interaction preferences, and gap penalties are additive.

The protein threading problem, under these basic assumptions, is generally considered to be computationally intractable. While this belief is strongly sup-

ported by the proof on the NP-hardness of the threading problem under a particular mathematical formulation [2], it also makes people shy away from seeking a more direct solution to the threading problem. Currently the most commonly used strategies for solving the threading problem include (1) nondeterministic approaches such as Monte Carlo method [3] and Gibbs Sampling [4]; (2) implicitly exhaustive search like branch-and-bound method [5]; (3) local search methods using the so-called “frozen approximation” [6]. While the strength and weakness of these strategies may vary, they do not generally guarantee finding an optimum threading, for practical purposes (it may require too much time, or the result is statistical in nature).

We have developed an algorithm which guarantees finding an optimum alignment between an amino acid sequence and a fold template. As pointed out by Lathrop [2], allowing variable-length gaps and pairwise interactions simultaneously is the reason for the computational difficulty of the threading problem. We have structured our threading algorithm in such a way that its computational complexity can be explicitly represented as a function of a parameter C , which characterizes the overall “structure” of the pairwise interactions and gaps. More specifically, the algorithm runs in $O((m + n^{1+0.5C} M \log(n))n^{C+1})$ time on a sequence of size n and a fold template of size m , consisting of M core secondary structures. This provides an effective framework for an in-depth study on how the fold recognition accuracy relates to the computational complexity of the threading problem. For a fixed fold template, the value of C changes as the allowed maximum distance, D , between two interacting amino acids (measured between their *beta* carbon atoms, or a virtual *beta* carbon position calculated based on the backbone structure for glycine) varies. In a case study, we have demonstrated that C is less than or equal to 4 for about 75% of the 296 unique folds in our database when we restrict $D \leq 7\text{\AA}$, implying the practical solvability of the threading problem on those folds. Our preliminary results suggest that for most of the protein folds, C stays as a small integer (3 or 4) even when D is increased to $9 \sim 10\text{\AA}$. For protein folds with $C > 4$, we have developed an approximation scheme, which removes the minimum number of pairwise interactions to make the C value small enough to be practically solvable.

2 Fold Recognition Through Threading

We have used the 3D protein structure database derived by Fischer *et al.* [7] as our protein database. This database consists of 301 distinct, representative, and accurate structures. We removed 5 uncompact structures (PDB codes 1mec4, 1mypa, 1scma, 2plv4, 2spca), and have also replaced 24 entries which have been up-

dated in Brookhaven Protein Data Bank (PDB) [8]. This database of 296 protein folds serves both as the source of deriving the knowledge-based energy functions and as the templates to align the sequences of unknown structures.

2.1 Energy calculation

A protein structure is governed by physical energies such as bond interaction, van der Waals, and electrostatics. It is well-known that these energy terms are both difficult to calculate (very time-consuming) and too sensitive to small displacements in atomic coordinates. As a result, most threading programs employ knowledge-based potential functions. The total potential energy E_{total} can be decomposed to the following two terms:

$$E_{total} = E_{local} + E_{pair}. \quad (1)$$

E_{local} represents a residue’s local preference of the secondary structures and its preference in certain solvent environment (either exposed to solvent or in the interior of the protein). E_{pair} is the pairwise interaction potential between amino acids.

To calculate E_{local} , most researchers treat the secondary structures preference and the preference of solvent accessibility separately by two decoupled energy functions. We have, instead, used a single energy function for different combinations of secondary structures and solvent accessibilities [9],

$$E_{local}(i, ss, sol) = -\log \frac{N(i, ss, sol)}{N_E(i, ss, sol)}. \quad (2)$$

$N(i, ss, sol)$ is the number of occurrences of amino acid i in the secondary structure conformation ss and in the solvent accessibility type sol , and $N_E(i, ss, sol)$ is the expected number of occurrences of (i, ss, sol) . It can be calculated by $\frac{N_i N_{ss} N_{sol}}{N_r^2}$, where N_i is the number of amino acid i , N_{ss} is the number of residues in the secondary structure ss , N_{sol} is the number of residues in the solvent accessibility type sol , and N_r is the total number of residues (all of which are calculated based on our protein database). The secondary structures of each protein are assigned by the DSSP package [10]. The solvent accessibility, which is defined as the percentage of exposed solvent accessible surface area of the residue, is calculated by the program ACCESS [11]. Three types of solvent accessibility sol are considered: *buried* ($sol \leq 9.7\%$), *exposed* ($sol > 49.0\%$), and the *intermediate* ($9.7\% < sol \leq 49.0\%$). The cutoffs are determined such that in each type of the solvent accessibility there are equal number of residues in the database. Combined with the 3 types of the secondary structures, there are 9 combinations of ss and sol . Our results

show that a secondary structure typically has significantly different preferences with different solvent accessibilities, and *vice versa*, indicating that our function is more sensitive than those treating the two preferences separately by two independent energy functions.

The pairwise potential, $E_{pair}(i, j)$, between residue i and residue j is derived from the frequency of the inter-residue interactions, i.e.,

$$E_{pair}(i, j) = -\log \frac{K(i, j)}{K_E(i, j)}, \quad (3)$$

$K(i, j)$ is the number of contacts between i and j in the database (for a specified cutoff distance), and $K_E(i, j)$ is the expected number of occurrences of (i, j) , which can be calculated by $\frac{\sum_i K(i, j) \sum_i K(i, j)}{\sum_{i, j} K(i, j)}$.

In our case study, we have used 7Å as the cutoff distance, which accounts for most of the important inter-residue interactions [12].

2.2 Problem formulation

Now we give a formal definition of the protein threading problem. Let $s = s_1 s_2 \dots s_n$ be an amino acid sequence, and (t, T) be a protein fold template, with $t = t_1 t_2 \dots t_m$ being a sequence of template positions (with an array of physical properties attached to each of them) and $T = T_1, \dots, T_M$ being the sequence of core secondary structures t is partitioned into. Let $(\bar{s}, \bar{t}) = ((\bar{s}_1, \bar{t}_1), (\bar{s}_2, \bar{t}_2), \dots, (\bar{s}_k, \bar{t}_k))$ be an alignment [13] of s and t , where \bar{s}_i (similarly \bar{t}_i) is either an element of s (or t) or ϕ (representing a gap), and $\max\{n, m\} \leq k \leq m + n$. Let $\mathcal{A}(s, t, T)$ denote the set of all possible alignments between s and t , which satisfy the following constraint^a: if s_i is aligned with t_p and s_j aligned with t_q , where t_p and t_q are from the same core of T , then s_{i+x} is aligned with t_{p+x} for all $x \in [1, j-i]$, and for any i and j . Let $\mathcal{I}(t, T)$ be the set of all pairs of positions from t 's different cores, with their distance \leq a specified cutoff value. For each pair (\bar{s}_i, \bar{t}_i) , $F(\bar{s}_i, \bar{t}_i)$ denotes the alignment score of the two, which could be either a fitness score or a gap penalty depending on what \bar{s}_i and \bar{t}_i are. For each pair of positions $(\bar{t}_i, \bar{t}_j) \in \mathcal{I}(t, T)$, and the two amino acids \bar{s}_i and \bar{s}_j assigned to them, the interaction preference is given by $P(\bar{s}_i, \bar{s}_j, \bar{t}_i, \bar{t}_j)$. A *protein threading problem* is defined as to find an alignment between the amino acid sequence s and the fold template (t, T) that minimizes the following function:

$$\min_{(\bar{s}, \bar{t}) \in \mathcal{A}(s, t, T)} \left\{ \sum_{1 \leq i \leq |\bar{s}|} F(\bar{s}_i, \bar{t}_i) + \sum_{(\bar{t}_i, \bar{t}_j) \in \mathcal{I}(t, T)} P(\bar{s}_i, \bar{s}_j, \bar{t}_i, \bar{t}_j) \right\}. \quad (4)$$

^aInformally, gaps are confined to regions between two cores or at the ends of a core.

3 An Algorithm for Optimum Threading

This section presents an algorithm for solving the minimization problem (4), and analyzes its computational complexity. This algorithm significantly improves the computational complexity of our previous work [14], and applies to a much wider range of threading problems.

3.1 The threading algorithm

An *interaction graph* of a fold template (t, T) is defined to have a vertex set $\{t_i | 1 \leq i \leq m\}$ and an edge set $\{(t_i, t_j) | (t_i, t_j) \in \mathcal{I}(t, T)\}$. Cutting^b the graph between vertices t_i and t_{i+1} creates a set of *open links* (see Figure 1), each of which has one end in the graph and the other end open. If we assign an orientation^c to a cut edge, one of its two open links is called an *in-link* and the other one an *out-link*. An *assignment* (of an amino acid) to an in-link means an (assumed) assignment of the same amino acid to the end vertex of its corresponding out-link.

We now outline a simple strategy for solving the minimization problem (4), which slightly generalizes^d Smith-Waterman sequence alignment algorithm and uses a *threading matrix* (storing combined alignment and interaction scores) like an alignment matrix in [13]. The basic idea is to calculate the threading matrix between the amino acid sequence s and the partial template $t[1, i]$ (note $t = t[1, m]$) for each combination of all possible assignments of the amino acids to the open links of $t[1, i]$ (treated as in-links), and resolve any inconsistency between an assumed assignment and the actual assignment to t_i by assigning $+\infty$ to the corresponding matrix cell, if t_i is a right-end vertex of a link. It repeatedly expands the threading matrix to include the next template position just as in Smith-Waterman algorithm, until $i = m$.

It can be shown that the minimum threading score between a sequence s and a template t among all assumed assignments is an optimum solution to problem (4), using a simple inductive argument on i , $i \in [1, m]$. A careful implementation of this strategy solves problem (4) in $O(mn20^{\max_{1 \leq i < m} \|C_i\|})$ time, where C_i denotes the open links at cut point i , and $\|\cdot\|$ represents the cardinality of a set. For an average fold template, the maximum cut size $\max_{1 \leq i < m} \|C_i\|$ could be as large as a few dozen or even larger, making this strategy impractical. In the rest of this section, we improve on this basic strategy by exploiting the constraint that gaps are confined to regions between cores or at the ends

^bCutting the interaction graph always means to cut between two adjacent vertices in this paper.

^cThe orientation of a cut edge is determined dynamically by our algorithm, and it affects its computational complexity.

^dSome care needs to be taken so that gaps are all confined to regions between cores or at the ends of a core.

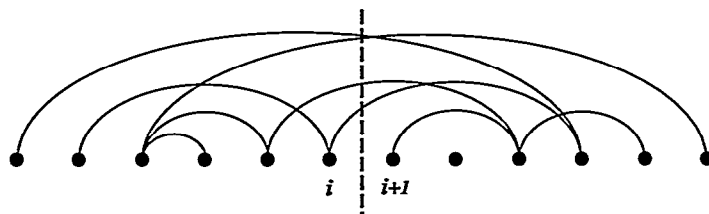


Figure 1: An interaction graph and a cut between vertices i and $i + 1$.

of a core. For the simplicity of discussion, we further assume that gaps are all confined to regions between cores. Extension to situations with gaps being at the ends of a core is straightforward.

Note that the essence of this strategy is to (repeatedly) solve the threading problem on a partial template with open links by examining all combinations of amino acid assignments to them, and the computational complexity depends on the maximum size of open links. While preserving the basic idea, we generalize this strategy by using a more general divide-and-conquer approach to greatly reduce the maximum open link size, and hence improve the computational complexity. It uses two key ideas: (1) group non-independent open links together, and (2) hierarchically partition the fold template into sub-templates in such a way that the maximum open link size, throughout the partition, is minimized.

Let $C_{i,j}$ be the set of edges connecting vertices of cores T_i and T_j , and a, b be two open links of $C_{i,j}$ (assuming $\|C_{i,j}\| \geq 2$). Since no gaps are allowed within a core, the choices of assignments to a and b are not independent. Obviously the total of number of all possible assignments to $C_{i,j}$'s open links is $O(n)$ (the sequence size). Hence we can redefine the interaction graph by contracting vertices of each core into one vertex and contracting the edges between two consecutive cores into one edge (see Figure 2(a)). Now each open link has $O(n)$ possible assignments, i.e., each position on the sequence gives a valid assignment to an open link.

The following observation helps to further reduce the open link size. Consider a series of cores a, b, c and d , and links (a, b) , (a, c) and (a, d) between them (see Figure 2(b)). A cut between a and b creates a partition a and (b, c, d) , and three open links. Note that these links are not independent due to the restriction of no gaps within a core. If we let them be all out-links of a (and hence in-links of b, c, d), these links effectively reduce to one link for this particular partition and this orientation assignment. Generally, for each core a , the number of open links can be counted as follows: all out-links of a are counted as one link; And for a series of cores in a partition, all its in-links directed from the same core (outside of this series of cores) are counted as one link. We call the number of open links counted

this way the *effective number of open links*.

The basic idea of our divide-and-conquer approach is to repeatedly bi-partition the (contracted) interaction graph (see Figure 2(c)), and solve the threading problem on a partial template with open links by optimally merging solutions for its sub-templates. We use the following example to explain the basic idea.

Consider a cut of the interaction graph at any position $i \in [1, M - 1]$. We want to show that a minimum-scoring threading between s and (t, T) can be constructed through merging a minimum-scoring threading between $s[1, j]$ and $T[1, i]$ with open links and a minimum-scoring threading between $s[j + 1, n]$ and $T[i + 1, M]$ with open links, for some $j \in [1, n - 1]$ and some set of "consistent" assignments to both open link sets of the two sub-templates. For each in-link, we examine every possible assignment to the end vertex of its corresponding out-link and calculate the interaction score $P()$ based on each assumed assignment; and for each out-link, we examine every possible assignment to its own end vertex and set the corresponding interaction score $P()$ to be zero.

Let $min_score(x, y, A)$ be the minimum threading score (as defined in problem (4)) between a (sub)sequence x and a (sub)template y under the assumption that y 's open links are assigned with A . The following statement can be shown using an inductive argument, which we omit in this abstract.

$$min_score(s, T, \emptyset) = \min_{1 \leq j < n, A} \{ min_score(s[1, j], T[1, i], A) + min_score(s[j + 1, n], T[i + 1, M], A) \} \quad (5)$$

where A represents a set of assignments to $T[1, i]$ (and also to $T[i + 1, M]$). This equation is the foundation of our divide-and-conquer algorithm. Note that equation (5) is true for any $i \in [1, M - 1]$ and any orientation assignments to the cut edges, which allows us to choose a partition strategy in such a way that its effective open link sizes are as small as possible. Section 3.2 gives a partition algorithm, which repeatedly bi-partitions a (partial) template into two sub-templates until all sub-templates become cores. It partitions the template in such a way that the maximum effective open link size

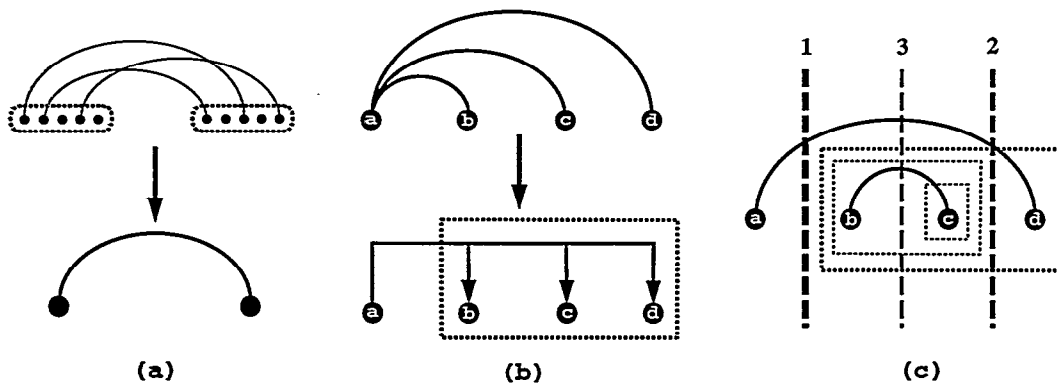


Figure 2: (a) Contraction of interaction graph. Each dotted box represents a core. (b) Merge of in-links and out-links. (c) A hierarchical partition of a template. Each node represents a core, and the numbered vertical lines represent a partition in the specified order.

of the partition is minimized. We call this minimized maximum effective open link size an *optimum open link size* of the template.

We now give the pseudo code of the threading algorithm. Given are a sequence s , a fold template (t, T) , and a partition strategy, calculated by the algorithm of Section 3.2. L represents the open links of T and A denotes a set of assignments to L . $score$ is a one-dimensional array, storing the last column (each template position corresponds to one column) of the threading matrix between T and $s[k, n]$; And $score1$ and $score2$ are defined similarly.

Algorithm *threading* $(T, s, k, L, A, score)$

1. if T is a core then call *align* $(T, s, k, L, A, score)$;
2. else
3. begin
4. divide T to T_1 and T_2 based on the partition strategy, and divide L to L_1 and L_2 accordingly;
let L_0 be the open links (between T_1 and T_2) created;
5. for each combination of assignments A_0 to L_0 do
6. begin
7. call *threading* $(T_1, s, k, L_0 \cup L_1, A_0 \cup A_1, score1)$;
8. for $i = k$ step $+1$ until n do
9. begin
10. call *threading* $(T_2, s, i, L_0 \cup L_2, A_0 \cup A_2, score2)$;
11. call *update_scores* $(score, score1, score2, i)$;
12. end
13. end
14. end

align $(T, s, k, L, A, score)$ uses a slightly-generalized Smith-Waterman alignment to calculate the threading matrix between core T and sub-sequence $s[k, n]$ under the assumption that the open links L are assigned with A , and stores the last column of the threading matrix

in array $score$. *update_score* $(score, score1, score2, i)$ is a procedure that updates the array $score$ by comparing the current values of $score$ with the values of $score2$ incremented by the value of $score1[i]$. By using a search tree data structure [15], each call to this procedure takes $O(\log n)$ time.

Based on recurrence (5) and the pseudo code, we can show that *threading* $(T, s, 1, \emptyset, \emptyset, score)$ calculates the score of an optimum threading between s and (t, T) . If C is the optimum open link size of the template, this algorithm runs in $O((m + n^{1+0.5C} M \log(n))n^{C+1})$ time and $O(nm + n^{C+2})$ space. We omit implementation details and the analysis on the computational complexity in this abstract. Some bookkeeping is needed for recovering an optimum threading from the minimum threading score, which can be done within the above time and space bounds.

3.2 Construction of an optimum partition

We define a *partition tree* of an interaction graph as follows. The tree root represents the whole interaction graph, and each tree leaf represents a single vertex of the interaction graph. Each non-leaf tree node has two children, each of which represents a subgraph created by a cut as illustrated in Figure 1. For each tree node x , we use $C_x(A_x)$ to denote the effective open link size (of x 's representing sub-graph) when orientations of its open links are given by A_x . The goal is to find a partition and a set of consistent orientation assignments that minimize the following function:

$$\min_{\text{partition, orientation}} \{ \max_x \{ C_x(A_x) \} \}. \quad (6)$$

We now outline an algorithm for calculating the minimum value of Problem (6). Let $A_{i,j}$ be a set of particular in/out assignments to the open links of $T[i, j]$. For any bi-partition $(T[i, k], T[k+1, j])$ of $T[i, j]$, $A_{i,j}^{i,k}$

and $A_{i,j}^{k+1,j}$ denote the corresponding portions of $A_{i,j}$ for the two parts, respectively, $k \in [i, j-1]$. We use $A_{i,j}^k$ to represent a set of in/out assignments to the open links created by dividing $T[i, j]$ into $T[i, k]$ and $T[k+1, j]$, and $\overline{A_{i,j}^k}$ represents the inversed in/out assignments to the other half of the open links. $I_{i,j}(A_{i,j})$ denotes the effective open link size of $T[i, j]$ under the assignment $A_{i,j}$, and $C_{i,j}(A_{i,j})$ denotes the smallest maximum effective open link size among all partitions of $T[i, j]$. The following equations can be shown using an inductive proof, which we omit.

$$C_{i,j}(A_{i,j}) = \begin{cases} \min_{i \leq k < j, A_{i,j}^k} \{ \max\{C_{i,k}(A_{i,j}^{i,k} \cup A_{i,j}^k), \\ C_{k+1,j}(A_{i,j}^{k+1,j} \cup \overline{A_{i,j}^k}), \\ I_{i,j}(A_{i,j})\} \}, & i < j, \\ \text{effective_open_link_size}(T[i, j] | A_{i,j}), & i = j. \end{cases} \quad (7)$$

It is not difficult to show that $C_{1,M}(\emptyset)$ gives an optimum solution to problem (6). Our algorithm calculates the $C_{i,j}$ values in a bottom-up fashion using the equation (6). Though the algorithm runs in exponential time of 2, it is efficient enough for our practical purpose due to the small size of the open links (at most a few dozen).

4 Discussion

We are now in the final stage of completing the implementation of the threading algorithm and the calculation of the “energy” terms, and expect to fully test the performance of the threading system in the very near future. By our estimation, the threading algorithm, in its current form, can find an optimum threading within a few minutes to a few hours, between a sequence of a few hundred amino acids and a fold template of similar size with $C \leq 4$.

We have calculated the C values for the 296 unique fold templates in our database when restricting the maximum distance between two interacting amino acids to 7Å, as a case study. Our results show that over 75% of the 296 fold templates have C values less than or equal to 4, and the rest are all small numbers, as shown in Figure 3. This indicates the general applicability of the threading algorithm on fold templates when we restrict the maximum interaction distance to 7Å.

Research is currently under way to study how well the algorithm generalizes when we relax the maximum interaction distance to larger values. We have studied how the total number of interactions and the optimum open link size C change as the maximum interaction distance, D , increases, on a portion of the protein database. Figure 4 shows the result on a typical protein

fold template (PDB code: lbrd – 171 residues). For this example, C stays ≤ 4 when $D \leq 20\text{Å}$.

For the practical purpose of using our threading algorithm, we have also designed a simple algorithm to remove the minimum number of edges to keep the C value \leq a specified C^* (for our case study $C^* = 4$) for a given fold template, and to use this as a pre-processing step of the threading algorithm. In describing the algorithm, we use the same notations as in Section 3.2 except that now each edge has a *removed/kept* assignment, denoted by A . In addition, let $w_{i,j}(A_{i,j})$ denote the minimum number of edges that need to be removed to guarantee that $T[i, j]$'s optimum open link size $\leq C^*$, under the removed/kept assignments $A_{i,j}$. The following equation can be shown using an inductive proof.

$$w_{i,j}(A_{i,j}) = \begin{cases} \infty, & \|A_{i,j}\| > C^* \\ \min_{i \leq k < j, \|A_{i,j}^k\| \leq C^*} \{ w_{i,k}(A_{i,j}^{i,k} \cup A_{i,j}^k) + \\ \frac{w_{k+1,j}(A_{i,j}^{k+1,j} \cup \overline{A_{i,j}^k}) -}{\|A_{i,j}^k\|} \}, & \|A_{i,j}\| \leq C^* \end{cases} \quad (8)$$

where $\|A\|$ represents the number of “kept” edges in A , $\|\overline{A}\|$ the number of “removed” edges, and the boundary condition of recurrence (8) is given by $w_{i,i}(A_{i,i}) = \|\overline{A_{i,i}}\|$ or ∞ , depending on if $\|A_{i,i}\| \leq C^*$ or not. On a typical fold template, our algorithm, using recurrence (8), removes the minimum number of edges in seconds to minutes of time.

An alternative approach for threading fold templates with large C values is to divide the protein into several domains. Large proteins typically are composed of domains of a size in the range of 100 to 150 amino acids [16, 17]. The domains are relatively stable by themselves, and they are large enough to be recognized by threading [18]. Our preliminary study shows that basically all the protein folds with $C > 4$ in our database can be cut into smaller domain folds with $C \leq 4$.

A number of issues are being studied related to the implementation of the threading algorithm. One of the key issues is that the algorithm keeps a number of very large tables during the execution to avoid recalculations, and its (disk/memory) space requirement could be too large to be practical when the problem sizes are becoming large. Research is currently under way to develop an implementation that is time most-efficient for a fixed amount of allowed space.

In conclusion, we have developed an algorithm that finds an optimum threading between an amino acid sequence and a protein fold template, and have structured

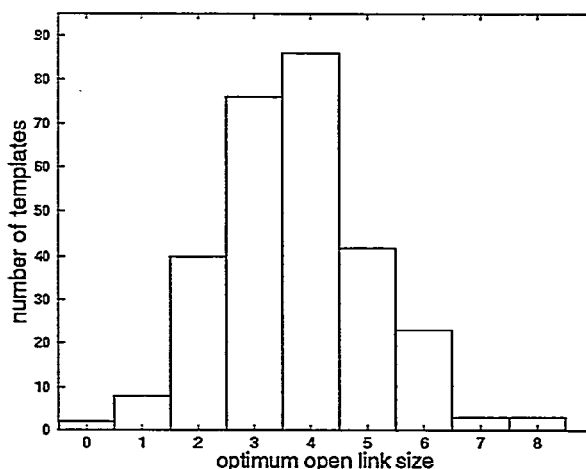


Figure 3: The distribution of the optimum open link size C among all 296 templates in our database, when the maximum interaction distance is restricted to 7\AA .

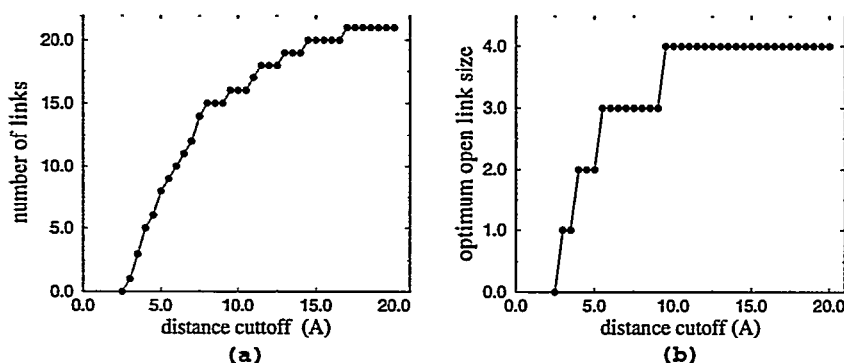


Figure 4: An example showing the relationship between maximum distance between interacting amino acids, and the total of number of links and the optimum open link size.

the algorithm in such a way which allows us to in-depth study how the maximum allowed interaction distance affects the computational complexity versus the fold recognition accuracy of the threading problem. Using this framework, we can determine what distance will be small enough for a practical solution to the threading problem, and large enough to guarantee an accurate fold recognition. We expect this study will lead to new insights into the computational protein threading problem, and possibly new and better ways to model the problem.

Acknowledgements

This research was supported by the United States Department of Energy, under contract DE-AC05-84OR21400 with Lockheed Martin Energy Systems, Inc.

References

- [1] T. F. Smith, L. Lo Conte, J. Bienkowska, C. Gaiatzes, R. G. Rogers Jr., and R. Lathrop, "Current Limitations to Protein Threading Approaches", *Journal of Computational Biology*, Vol. 3, 4, pp. 217 - 225, 1997.
- [2] R. H. Lathrop, "The Protein Threading Problem With Sequence Amino Acid Interaction Preferences is NP-complete", *Protein Engineering*, Vol. 7, No. 9, pp. 1059 - 1068, 1994.
- [3] S. H. Bryant and S. F. Altschul, "Statistics of Sequence-Structure Threading", *Curr. Opin. Struct. Biol.*, Vol. 5; pp. 236 - 244, pp. 1995.
- [4] T. Madej, J. F. Gibrat and S. H. Bryant, "Threading a Database of Protein Cores", *Proteins: Structure, Function, and Genetics*, Vol. 23, pp. 356 - 369, 1995.

- [5] R. H. Lathrop and T. F. Smith, "Global Optimal Protein Threading With Gapped Alignment and Empirical Pair Score Functions", *J. Mol. Biol.*, Vol. 255, pp. 641 - 665, 1996.
- [6] A. Godzik, A. Kolinski, and J. Skolnick, "Topology Fingerprint Approach to the Inverse Folding Problem", *J. Mol. Biol.*, Vol. 227, pp. 227 - 238, 1992.
- [7] D. Fischer, C. J. Tsai, R. Nussinov, and H. Wolfson, "A 3D Sequence-Independent Representation of the Protein Data Bank", *Protein Engng*, Vol. 8, pp. 981 - 997, 1995.
- [8] F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi, "The Protein Data Bank: A Computer Based Archival File for Macromolecular Structures", *J. Mol. Biol.*, Vol. 112, pp. 535 - 542, 1977.
- [9] M. S. Johnson, J. P. Overington, and T. L. Blundell, "Alignment and Searching for Common Protein Folds Using a Data Bank of Structural Templates", *J. Mol. Biol.*, Vol. 231, pp. 735 - 752, 1993.
- [10] W. Kabsch and C. Sander, "Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen Bonded and Geometrical Features", *Biopolymers*, Vol. 22, pp. 2577 - 2637, 1983.
- [11] S. Hubbard, *ACCESS*, University College, London, 1992.
- [12] D. T. Jones and J. M. Thornton, "A New Approach to Protein Fold Recognition", *Nature*, Vol. 358, pp. 86 - 89, 1992.
- [13] T. F. Smith and M. Waterman, "Comparison of Biosequences", *Advances in Applied Mathematics*, Vol. 2, pp. 482 - 489, 1982.
- [14] Y. Xu and E. C. Uberbacher, "A Polynomial-Time Algorithm for a Class of Protein Threading Problems", *Computer Applications in Biosciences*, Vol. 12, pp. 511 - 517, 1996.
- [15] R. E. Tarjan, *Data Structures and Network Algorithms*, Philadelphia, PA, Society for Industrial and Applied Mathematics Press, 1983.
- [16] S. A. Islam, J. Luo, and M. J. Sternberg, "Identification and Analysis of Domains in Proteins", *Biochemistry*, Vol. 24, pp. 1501-1509, 1985.
- [17] D. Xu, and R. Nussinov, "Favorable Domain Size in Proteins", *Folding & Design*, Vol. 3, 11-17, 1997.
- [18] W. P. Hu, A. Godzik and J. Skolnick, "Sequence-Structure Specificity—How Does an Inverse Folding Approach Work?", *Protein Eng.*, Vol. 10, 317-331, 1997.